Siebel

Deploying Siebel CRM Containers on Kubernetes using Siebel Cloud Manager

October 2026

Siebel
Deploying Siebel CRM Containers on Kubernetes using Siebel Cloud Manager

October 2026

F83038-37

Copyright © 2026, Oracle and/or its affiliates.

Author: Sukanya Mishra

Contents

Get Help i

1 What's New in This Release 1

What's New in This Release 1

2 Overview 25

About this Chapter 25
About Siebel Cloud Manager 25
About Siebel CRM Upgrade Factory 30

3 Deploying Siebel CRM on OCI using Siebel Cloud Manager 33

About this Chapter 33
Overview of Deploying Siebel CRM on OCI 34
Requirements and Limitations 34
High-Level Steps to Deploy Siebel CRM Using SCM 35
Creating a Compartment 37
Git Repositories for Siebel CRM Deployment 37
Using Vault for Managing Secrets 42
Downloading and Installing Siebel Cloud Manager 44
Upgrading Oracle Linux on the SCM VM 49
About URLs for Siebel CRM Deployments on OCI 55
Uploading Files to the SCM Container Using File Sync Utility 56
Mirroring Siebel Base Container Images 60
Reducing the Ingress Range for Siebel Cloud Manager 64
Using Advanced Network Configuration 65
Customizing Configurations Prior to Greenfield Deployment 67
Deploying Siebel CRM on OCI 70
Additional Administrative Tasks in Siebel Cloud Manager 113
Troubleshooting a Siebel Cloud Manager Instance or Requested Environment 121
Updating Siebel Cloud Manager with a New Container Image 129
Updating Ingress Controller 131

Migrating (Lift-And-Shift) Existing Siebel CRM Deployments 132
Removing a Siebel CRM Deployment on OCI 132
Making Incremental Changes to Your Siebel CRM Deployment on OCI 133
High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM 162
Enabling TLS 1.3 Support in Environments Prior to 23.11 162
Rotating Secrets 164
Assigning Pods to Nodes - Implementing Affinity and Anti-affinity on OKE using Siebel Cloud Manager 174
Adding and Overriding Environment Variables 178
Cleaning up the Siebel File System 181

4 Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer 185
About this Chapter 185
Overview 185
Moving Existing Siebel CRM on VM to a Kubernetes Orchestrated Deployment 186
Moving Existing Siebel CRM on VM to an OC3 Kubernetes Cluster 186
High-Level Steps for Deploying Siebel CRM on a Kubernetes Cluster 188
Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster 188
Downloading and Running Siebel Installer for SCM 191
Installing SCM using Helm 199
Migrating (Lift-And-Shift) Existing Siebel CRM Deployments 208
Deploying Siebel CRM using SCM 208
Updating SCM Configuration using Helm 213
Reinstalling SCM using Helm 214
Updating SCM using Helm 214
Uninstalling SCM using Helm 215
Troubleshooting Siebel CRM Deployment 215
Deploying Siebel CRM on OpenShift using SCM 216

5 Migrating Siebel CRM Deployments Using Siebel Lift Utility 227
About this chapter 227
Overview 227
Lifting Existing Siebel CRM Deployments 228
Shifting Existing Siebel CRM Deployments 240

6 Deploying Open Integration on a Kubernetes Cluster Using SCM 241
About this Chapter 241

Deploying Open Integration Using SCM 241
Rerunning the Open Integration Deployment 257
Deleting an Open Integration Deployment 258
Viewing all Open Integration Deployments 259
Performing Open Integration Incremental Updates 260
Troubleshooting Open Integration Deployment Issues 262

7 Monitoring Siebel CRM Deployments 265
Monitoring Siebel CRM Deployments 265
Metrics Information Categories 266
Siebel CRM Monitoring Architecture 268
Key Software Components for Monitoring 268
Visualization Components for Monitoring 269
Configuring the Siebel CRM Observability – Monitoring Solution 270
Dashboards for Siebel CRM Monitoring 280
Siebel CRM Observability - Data Visualization and Metrics Monitoring for Oracle Databases 282

8 Log Analytics in Siebel CRM Deployments 291
Log Analytics in Siebel CRM Deployments 291
Log Analytics Tooling Options 292
Solution Architecture and Components 294
Log Collection and Aggregation 295
Sample Dashboards 295
Pre-requisites for Enabling OCI Logging Analytics 295
Enabling Log Analytics in Siebel CRM Observability 296
Disabling Log Analytics in Siebel CRM 297
Accessing Log Analytics URLs 297
Oracle OpenSearch Usage in Siebel CRM Observability – Log Analytics 298
OCI Logging Analytics Configurations of Importance 298
OCI Logging Analytics Usage in Siebel CRM Observability – Log Analytics 299
Extending Siebel CRM Observability – Log Analytics 299

9 Siebel CRM Deployment Parameters 303
Overview 303
Parameter Usage Considerations 303
Payload Parameters for Siebel CRM Deployment 303

10 Appendix 333

Applying Siebel POC Patches in Siebel CRM Environment Deployed using SCM 333
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM 336
Post-Update Security Hardening for SCM and Siebel Pods 361
Migrating Siebel CRM Containers to OpenSSL Encryption 376
Configuring the Time Zone for SCM and Siebel Pods 380
Enabling Authentication for Siebel CRM Observability 383

Get Help

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at *https://docs.oracle.com/*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an email to: *oracle_fusion_applications_help_ww_grp@oracle.com*.

1 What's New in This Release

What's New in This Release

What's New in October 2026, CM_26.10

The following table lists the changes in this revision of the documentation to support this release (CM_26.10) of the software.

Topic	Description
Rotating the Load Balancer TLS Certificate for Ingress-Nginx	Added topic. This topic describes how you can replace the load balancer TLS certificate for an existing Siebel CRM environment that uses ingress-nginx in SCM 26.2 or earlier while preserving the existing Kubernetes Service, OCI load balancer IP, and external IP address.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Updated topic. Added the *Troubleshooting HelmRelease Failure* section, which explains how to check release status, roll back a failed Helm release to a stable revision, and increment the version in **`Chart.yaml`** for the next Helm upgrade.
Payload Parameters for Siebel CRM Deployment	Updated topic. Replaced **`smtp_auth_password_vault_ocid`** with **`smtp_auth_password`**. If Vault is enabled, this parameter accepts the OCID of the Vault secret that contains the SMTP authentication password. Otherwise, enter the password in plain text.

What's New in September 2026, CM_26.9

The following table lists the changes in this revision of the documentation to support this release (CM_26.9) of the software.

Topic	Description
Mirroring Siebel Base Container Images	Updated topic. Updated the topic with the *Signing Mirrored Images with a Customer Key* section. This section describes how you can optionally sign mirrored images with a customer key by using Cosign. It explains the supported signing keys, the **`customer_signing`** configuration, and how SCM verifies customer-signed images during Siebel CRM environment creation.
Downloading and Running Siebel Installer for SCM	Updated topic. Added support for optional customer image signing with Cosign for images pushed during Siebel Installer setup. The documentation describes supported signing keys, required inputs, generated image trust files, and Helm configuration for verifying customer-signed images.
Deploying Siebel CRM on OCI	Updated topic. • Updated the topic with the *Configuring Fixed NodePorts for Traefik* section. This section describes how to configure fixed Kubernetes NodePorts for the Traefik HTTP and HTTPS entry points. Fixed NodePorts let you configure external firewall, network security group, and security-list rules for specific ports. • Updated the topic with the *Using a Customer-Managed KMS Key for an SCM-Created Autonomous Database* section. This section describes how to change the encryption key for an

Topic	Description
	existing ATP database from an Oracle-managed key to the selected customer-managed key. It explains the required database, Vault, dynamic group, and IAM policy prerequisites.
Installing SCM using Helm	Updated topic. Updated the "Before Installing SCM" section with the following: • Added steps to configure image trust files and the **`imageTrust`** section in the SCM Helm chart when customer image signing is enabled during Siebel Installer setup. • Updated the **`service`** section to describe how you can assign a fixed Kubernetes NodePort to the SCM service when your network configuration requires a specific port for SCM access.
Git Repositories for Siebel CRM Deployment	Updated topic. • Updated the topic with the *Using Git Commit Signing* section. This section describes how to enable Git commit signing with trusted GPG keys so that Flux applies repository changes only when the latest commit is trusted. It also explains how to sign customer commits and rotate customer, SCM, and Flux signing keys after deployment. • Added SSH host-key verification for BYO Git deployments. You can now provide verified SHA-256 Git host-key fingerprints through the **`git_ssh_host_key_sha256`** parameter.
Troubleshooting a Siebel Cloud Manager Instance or Requested Environment	Updated topic. • Updated the *Troubleshooting a Siebel Cloud Manager Instance* section to describe safe SCM error responses. SCM now returns a general error message and a relative **`data.log_path`** value that directs administrators to the relevant diagnostic log without exposing sensitive details in the API response. • Updated the topic with the *Troubleshooting Stalled Resources Error* section. This section describes how to troubleshoot Stalled Resources errors by triggering a new deployment attempt, monitoring the deployment, and collecting Events, resource descriptions, and container logs to identify the root cause. • Updated the topic with the *Troubleshooting Customer-Managed KMS Key Issues for an SCM-Created Autonomous Database* section. This section provides troubleshooting steps for customer-managed KMS key-change work requests on SCM-created Autonomous Databases. It helps you verify database, Vault, key, dynamic group, and IAM policy settings before you retry the key-change operation.
Payload Parameters for Siebel CRM Deployment	Updated topic. • Added the **`commit_signing`** section, including the **`enabled`** and **`customer_public_key_file`** parameters, under the **`infrastructure > git`** section to configure Git commit signing for SCM Git repositories. • Added the **`node_ports`** section, including the **`web`** and **`websecure`** parameters, under **`infrastructure > ingress_controller`** to configure fixed Kubernetes NodePorts for the Traefik HTTP and HTTPS entry points. • Added the **`git_ssh_host_key_sha256`** parameter under the **`infrastructure > git > byo_git`** section to provide verified SHA-256 Git host-key fingerprints.

What's New in August 2026, CM_26.8

The following table lists the changes in this revision of the documentation to support this release (CM_26.8) of the software.

Topic	Description
Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM	Added topic. This topic describes how you can use the optional customer-managed OCI Vault Key Management Service (KMS) keys to encrypt supported infrastructure resources, such as OKE Kubernetes secrets, boot volumes, file systems, and Siebel environment file systems. Using customer-managed keys gives you greater control over the keys used to encrypt these resources.
Using Vault for Managing Secrets	Updated topic. Updated the topic to describe how to use OCI Vault KMS key OCIDs with the BYO Vault option. The topic now explains how to specify KMS key OCIDs for SCM storage and Siebel environment file storage, create the required dynamic groups, and grant SCM-created resources the permissions required to use the keys.
Downloading and Installing Siebel Cloud Manager	Updated topic. • Updated the SCM Instance Configuration step to explain how to configure the allowed source CIDR for SCM access during SCM stack creation. It explains when the field is required, how to allow access from a single IP address or subnet, which CIDR ranges SCM does not support, and how access is handled for private SCM deployments. • Updated the Key Management step. When "Use Existing Resources" is selected, you can now optionally specify a customer-managed KMS key OCID for SCM storage resources along with an existing OCI Vault. • Added a step that describes how to configure tags for SCM-created resources during SCM stack creation, including how to use both defined tags and free-form tags.
Example Payload to Deploy Siebel CRM	Updated topic. Updated the topic with the "Example for Using Customer-Managed KMS Keys for Infrastructure Resources Created by SCM" section. This section shows how to use customer-managed KMS keys for infrastructure resources, created by SCM, in the Siebel CRM deployment payload.
Enabling Authentication for Siebel CRM Observability	Added topic. This topic describes how to configure mandatory HTTP Basic Authentication for public Prometheus and OpenSearch Dashboards endpoints.
Downloading and Running the Siebel Lift Utility	Updated topic. Updated the topic to reflect that the Siebel Lift utility is now delivered through My Oracle Support (MOS) instead of the SCM download API. Added the "Downloading the Siebel Lift Utility" section, which describes how to download the Siebel Lift utility from MOS.
Using Network Policies	Added topic. This topic describes how SCM generates Kubernetes NetworkPolicy for SCM-managed CRM workloads. By default, SCM enables network policy generation unless you explicitly disable it in the Siebel CRM deployment payload.
Troubleshooting a Siebel Cloud Manager Instance or Requested Environment	Updated topic. Updated the topic with the "Troubleshooting Issues Related to Customer-Managed KMS Keys" section. This section helps you diagnose and resolve issues related to customer-managed KMS key in SCM.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the **`network_policy.enabled`** parameter to control Kubernetes ingress **`NetworkPolicy`** generation for SCM-managed CRM workloads. Added the **`basic_auth`** section, including the **`users`**, **`username`**, and **`password`** parameters, under **`prometheus_auth`**. Added the **`basic_auth`** section, including the **`users`**, **`username`**, and **`password`** parameters, under **`opensearch_dashboard_auth`**.

What's New in July 2026, CM_26.7

The following table lists the changes in this revision of the documentation to support this release (CM_26.7) of the software.

Topic	Description
About Siebel Cloud Manager	Updated topic. Introduced support for deploying AI Connectors on a Kubernetes cluster using SCM.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Updated topic. Revised the procedure for installing Siebel monthly updates to create backups and build custom Siebel images on an Oracle Linux VM using the mounted Siebel and SCM file systems. Also added the prerequisites and pre-installation tasks required for the revised procedure.
Payload Parameters for Siebel CRM Deployment	Updated topic. Updated the description of the **`to_email`** parameter under the **`observability > alertmanager_email_config`** section.

What's New in June 2026, CM_26.6

The following table lists the changes in this revision of the documentation to support this release (CM_26.6) of the software.

Topic	Description
About Siebel Cloud Manager	Updated topic. Added the *Running Workloads with Custom Non-Root User ID and Group ID* section, which introduces support for running SCM resources, Siebel CRM workloads, and observability workloads with custom non-root user ID and group ID.
Deploying Siebel CRM on OpenShift using SCM	Updated topic. Added guidance for configuring Security Context Constraints (SCCs) for enhanced security and default security deployments. Added instructions for running SCM and Siebel CRM workloads with custom user ID and group ID.
Installing SCM using Helm	Updated topic. Added guidance for configuring SCM to run with a custom non-root user ID and group ID by using the **`securityContext`** section in the SCM Helm chart **`values.yaml`** file.
Deploying Siebel CRM using SCM	Updated topic. Added prerequisite steps for preparing Siebel File System and migration file system directories when using custom non-root user ID security.
Monitoring Siebel CRM Deployments	Updated topic. Added guidance for configuring observability workloads to run with a custom non-root user ID and group ID by using **`observability.security_context`**. Added the *Observability: Local Storage Prerequisite for Custom Non-Root User ID Security* section to describe the Prometheus local storage ownership prerequisite for deployments that use local storage.
Deploying Siebel CRM on OCI	Added the *Preparing Siebel File System and Migration Directories* section that describes the procedure for preparing Siebel File System directories before deployment when using custom non-root user ID and group ID with bring-your-own file systems. Added the *Using Resource Tags* section, which describes support for configuring OCI resource tags by using the **`resource_tags`** section in the SCM provisioning payload.
Deploying Open Integration on a Kubernetes Cluster Using SCM	Updated chapter. Added Git-based deployment support, including customer Git registration, starter template management, Git-based configuration (**`git_id`**, branch, and directory), infrastructure provisioning, deployment validation, and rerun.
Example Payload to Deploy Siebel CRM	Updated topic. Added the *Example for Using Resource Tags* section to demonstrate how to configure the **`resource_tags`** section in the Siebel CRM deployment payload.
Post-Update Security Hardening for SCM and Siebel Pods	Added topic. This topic describes the post-update procedure to move an environment to the hardened-security model after the monthly update is complete. The hardened security model enables workloads to run with custom non-root user and group IDs, applies least-privilege security principles, and aligns with Kubernetes and OpenShift security requirements.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Added topic. This topic describes the procedure to install the latest monthly update in a Siebel CRM environment deployed on OpenShift using SCM.

Topic	Description
Applying Siebel POC Patches in Siebel CRM Environment Deployed using SCM	Updated topic. Updated the procedure for patching Siebel CRM containers with the POC provided by Oracle Support to resolve an issue affecting a specific Siebel CRM application version.
Installing Siebel Monthly Update in a Siebel CRM Environment Deployed on OpenShift with Security Hardening Using SCM	Added topic. This topic describes the procedure to install the latest monthly update in a Siebel CRM environment deployed on OpenShift with security hardening using SCM.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added **`security_context`**, **`run_as_user`**, **`run_as_group`**, **`fs_group`**, **`resource_tags`**, **`defined_tags`**, **`and freeform_tags`** parameters under the **`siebel`** and **`observability`** sections. Updated the descriptions for **`kubernetes_type`**, **`kubeconfig_path`**, and **`byo_ns`** under the **`infrastructure`** section.

What's New in May 2026, CM_26.5

The following table lists the changes in this revision of the documentation to support this release (CM_26.5) of the software.

Topic	Description
Updating SCM using Helm	Modified topic. Provided instructions on how to enable the newly introduced SCM features after updating SCM to a new version.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Modified topic. Provided reference to instructions on how to enable separate keystores for server authentication and client authentication.

What's New in April 2026, CM_26.4

The following table lists the changes in this revision of the documentation to support this release (CM_26.4) of the software.

Topic	Description
Example Payload to Deploy Siebel CRM	Updated topic. Added the "Example for BYO Database" section to demonstrate how to configure the **`database`** section in the Siebel CRM deployment payload when you bring your own database.

What's New in March 2026, CM_26.3

The following table lists the changes in this revision of the documentation to support this release (CM_26.3) of the software.

Topic	Description
About Siebel Cloud Manager	Modified topic. Added the *Managing Ingress using Traefik* section to introduce Traefik as the ingress controller for Siebel CRM deployments created using SCM.
Use Cases for Making Incremental Changes	Updated topics. Updated the steps and sample code blocks to reflect the migration from the NGINX Ingress Controller to Traefik.
Updating Ingress Controller	Added topic. This topic explains how to update the ingress controller to Traefik when you update from an SCM release earlier than 26.3.

Topic	Description
Using Custom Keystore	Updated topic. Updated the procedure to generate a custom keystore using two separate certificates—one for server authentication and another for client authentication. This approach improves security and supports compliance because, effective May 1, 2026, many Trusted Root Certification Authorities will no longer issue SSL certificates that include both the server authentication and client authentication Extended Key Usage (EKU).
Deploying Open Integration on a Kubernetes Cluster Using SCM	Updated chapter. Updated the Open Integration deployment payload to use two separate certificates—one for server authentication and another for client authentication.
Migrating Siebel CRM Containers to OpenSSL Encryption	Added topic to Appendix. This topic describes the steps to migrate Siebel CRM containers to OpenSSL encryption when you update existing container-based Siebel CRM deployments from a release earlier than 25.10 to 25.10 or later. This migration is mandatory to align with current Oracle security requirements and industry security standards.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the following parameters under the **`siebel > keystoresection`**: **`use_self_signed_certificate`**, **`siebel_server_keystore_path`**, **`siebel_client_keystore_path`** , **`siebel_server_keystore_password`**, **`siebel_client_keystore_password`**, **`siebel_server_certificate_alias`**, and **`siebel_client_certificate_alias`**.

What's New in February 2026, CM_26.2

The following table lists the changes in this revision of the documentation to support this release (CM_26.2) of the software.

Topic	Description
Using Custom Keystore	Updated topic. Updated the steps to enable secure communication between the Metacontroller and the Siebel Controller using HTTPS. This ensures encrypted data transmission and aligns with current security and compliance standards.
Siebel CRM Deployment Parameters	Added chapter. This chapter provides detailed information about the Siebel CRM deployment parameters. Note: In the previous releases, this information was found under the topic "Parameters in Payload Content."
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the following parameters to enable secure communication between the Metacontroller and the Siebel Controller using HTTPS, under the **`siebel > keystore`** section: **`siebel_certificate_alias`** and **`siebel_controller_certificate_alias`**.

What's New in January 2026, CM_26.1

The following table lists the changes in this revision of the documentation to support this release (CM_26.1) of the software.

Topic	Description
Troubleshooting Common Dashboard Issues	Added topic. This topic describes how to troubleshoot and resolve NGINX dashboard loading issues.
Troubleshooting Oracle DB Exporter Database Access Issues	Added topic. This topic describes how to troubleshoot and resolve Oracle Database access issues from the Oracle DB Exporter.

Topic	Description
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Updated topic. Updated the steps to install latest monthly updates in a Siebel CRM environment deployed using SCM.

What's New in December 2025, CM_25.12

The following table lists the changes in this revision of the documentation to support this release (CM_25.12) of the software.

Topic	Description
Siebel CRM Observability - Data Visualization and Metrics Monitoring for Oracle Databases	Added topic. This topic describes how to monitor your Siebel CRM database health, operational behavior, and business-state metrics from live queries in real time using the Oracle DB Exporter application and how to visualize metrics data in dashboards like Grafana. This supports faster troubleshooting, better decision-making, and a more resilient Siebel CRM environment. Oracle DB Exporter is integrated with SCM to provide unified visibility into Oracle databases deployed on Kubernetes – both on premises and in the cloud. Oracle DB Exporter: • Provides a comprehensive set of default metrics for performance monitoring and issue diagnosis across Oracle databases. • Supports custom metrics, which can be defined using a TOML (Tom's Obvious, Minimal Language) configuration file. You can enable Oracle database monitoring either during Siebel CRM deployment or on an existing Siebel CRM deployment, as follows: • Set the **`enable_oracle_db_monitoring`** parameter in Siebel deployment payload to **`true`**. • Configure the **`auth_info`** section based on the database type (ATP, DBCS_VM, or BYOD). You can access the metrics through the Prometheus or Grafana dashboards.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Updated topic. Updated the steps to install latest monthly updates in a Siebel CRM environment deployed using SCM to incorporate the following changes: • The **`busybox`** image has been rebuilt as **`base-builder`**. • The path for the **`dbutils`** repository has changed.
Deploying Open Integration on a Kubernetes Cluster Using SCM	Added chapter. This chapter describes how to deploy Open Integration using SCM 25.12 or later, on a Kubernetes cluster. Open Integration provides tooling for the creation and execution of REST APIs, enabling you to expose Siebel CRM processes externally. SCM simplifies the deployment of Open Integration on a Kubernetes cluster by automating key tasks such as version control, environment setup, and configuration changes.
Use Cases for Making Incremental Changes	Updated topic. Added the *Use Cases for Rerunning the PostInstallDB Job* topic. This topic describes the steps to rerun the PostInstallDB job using SCM.
Downloading and Running Siebel Installer for SCM	Updated topic. Updated the steps to download Siebel Installer from My Oracle Support (MOS) to reflect the recent changes in the MOS user interface.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the following parameters to enable Oracle Database monitoring under the observability section: **`enable_oracle_db_monitoring`**, **`db_metrics_exporter_username`**, **`db_metrics_exporter_password`**, **`wallet_path`**, **`tns_connection_name`**, **`data_as_metrics_exporter_username`**, and **`data_as_metrics_exporter_password`**.

What's New in November 2025, CM_25.11

The following table lists the changes in this revision of the documentation to support this release (CM_25.11) of the software.

Topic	Description
Downloading and Running Siebel Installer for SCM	Updated topic: • Added support for Siebel Web Tools in languages other than English (ENU) when deploying Siebel CRM on a Kubernetes cluster using Siebel Installer for SCM. • Removed the requirement for mirroring when deploying Siebel CRM on a Kubernetes cluster. Mirroring API calls are no longer required when deploying Siebel CRM on a Kubernetes cluster using Siebel Installer for SCM. You can download Siebel Installer, container images, and Safeboot DLLs from My Oracle Support (MOS) and proceed with the Siebel CRM deployment procedure without internet access.
Deploying Siebel CRM on OpenShift using SCM	Added topic. This topic describes how to deploy Siebel CRM on RedHat OpenShift using Siebel Installer for SCM.
Example Payload to Deploy Siebel CRM	Updated topic. Added the "Example for Configuring a Standalone Siebel Gateway" section to demonstrate how to deploy a single Siebel Gateway through the **`gateway_deployment_type`** parameter to in the Siebel CRM deployment payload.
Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers	Updated topic. Added configuration to enable access to Siebel CRM application user interfaces.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the **`gateway_deployment_type`** parameter to deploy a single Siebel Gateway under the **`siebel`** section.

What's New in October 2025, CM_25.10

The following table lists the changes in this revision of the documentation to support this release (CM_25.10) of the software.

Topic	Description
Tools for Deploying Siebel CRM using SCM	Updated topic. Added information about the Artifactory server.
High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM	Updated topic. Updated the steps to install latest monthly updates in a Siebel CRM environment deployed using SCM.
Migrating Siebel CRM Deployments	Added chapter. This chapter describes the procedure for migrating an existing Siebel CRM deployment to the cloud or any CNCF certified Kubernetes cluster.
Installing SCM using Helm	Introduced the **`affinity`** section in the **`values.yaml`** file to configure node affinity scheduling rules for the SCM pod.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the **`path`**, **`server`**, and **`storage`** parameters under the **`siebel > nfs`** section. Updated the description of the **`bucket_url`** parameter.

What's New in September 2025, CM_25.9

The following table lists the changes in this revision of the documentation to support this release (CM_25.9) of the software.

Topic	Description
Troubleshooting a Siebel Cloud Manager Instance	Updated topic. Added steps to troubleshoot the issue of SCM container start failure.

What's New in August 2025, CM_25.8

The following table lists the changes in this revision of the documentation to support this release (CM_25.8) of the software.

Topic	Description
Git Repositories for Siebel CRM Deployment	Updated topic. Added high-level steps to create an OCI DevOps project and repository. Updated the GitLab installation steps.
Upgrading Oracle Linux on the SCM VM	Updated topic. Updated the steps to upgrade the SCM VM instance from Oracle Linux 7 to Oracle Linux 8.
Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM	Added topic to Appendix. This topic describes the steps to install Siebel CRM monthly updates on an OKE environment deployed by SCM.
Example Payload to Deploy Siebel CRM	Modified topic. Added the "Example for BYO Namespace" section to demonstrate how to configure the **`kubernetes`** section for BYO namespace in the Siebel CRM deployment payload.
Troubleshooting a Siebel Cloud Manager Instance or Requested Environment	Modified topic. Added the "Troubleshooting BYO Git Failure" section to demonstrate how to resolve BYO Git **`process_cg_artifacts`** stage failure.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the **`git_selfsigned_cacert`** parameter under the **`infrastructure > git > byo_git`** section.

What's New in July 2025, CM_25.7

The following table lists the changes in this revision of the documentation to support this release (CM_25.7) of the software.

Topic	Description
High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM	Updated topic. Updated the steps to install latest monthly updates in a Siebel CRM environment deployed using SCM.
Using a BYO Namespace	Added topic. Added support for Bring Your Own (BYO) namespace.
Applying Siebel POC Patches in Siebel CRM Environment Deployed using SCM	Added topic to Appendix. This topic describes the procedure to patch your Siebel CRM containers with POC patches provided by Oracle Support.
Payload Parameters for Siebel CRM Deployment	Updated topic. Added the parameter **`byo_ns`** for BYO namespace.

What's New in June 2025, CM_25.6

The following table lists the changes in this revision of the documentation to support this release (CM_25.6) of the software.

Topic	Description
About Siebel Cloud Manager	Updated topic. Added information on the infrastructure provisioning options available for deploying Siebel CRM using SCM. Added the high-level steps for deploying Siebel CRM using SCM.
Use Cases for Updating Keystore File as Part of Incremental Changes	Updated topic. Added a note for the expected status of the pod and Helm release during flux reconciliation.
Terminating SSL/TLS at the Load Balancer (FrontEnd SSL) using SCM	Updated topic. Renamed the parameter **`lb-ssl-certificate`** to **`lb-tls-certificate`**.
Payload Parameters for Siebel CRM Deployment	Updated topic. Updated the description of the parameter **`cpu_cores`** under the **`database > atp`** section.

What's New in May 2025, CM_25.5

The following table lists the changes in this revision of the documentation to support this release (CM_25.5) of the software.

Topic	Description
Using Custom Keystore	Updated topic. Added the steps to generate and use custom certificate.
Downloading and Running Siebel Installer for SCM	Updated topic. Added the steps to run Siebel Installer in silent mode.
Downloading and Running the Siebel Lift Utility	Updated topic. Updated all Docker references to Podman as the Siebel Lift Utility (Container Mode) now uses Podman.

What's New in April 2025, CM_25.4

The following table lists the changes in this revision of the documentation to support this release (CM_25.4) of the software.

Topic	Description
Upgrading Oracle Linux on the SCM VM	Added topic. Describes the steps to upgrade the SCM VM instance from Oracle Linux 7 to Oracle Linux 8.
Adding and Overriding Environment Variables	Added topic. Describes the steps to add new environment variables and override existing environment variables in individual Siebel pods.
Installing SCM using Helm	Updated topic. Added the steps to configure SSL certificates for different service types in the **`values.yaml`** file.
Downloading the Siebel Lift Utility (for Container Mode)	Updated topic. Additional note added for using Podman as the container management tool.

Topic	Description
Payload Parameters for Siebel CRM Deployment	Modified topic. Updated the description of the **db_version** parameter under the **atp** and **dbcs_vm** sections.

What's New in March 2025, CM_25.3

The following table lists the changes in this revision of the documentation to support this release (CM_25.3) of the software.

Topic	Description
Updating Git CA Certificate	Added topic. Describes the procedure to rotate Git CA certificates.
Troubleshooting Proxy Server Settings Propagation Failure	Added topic. Describes how to resolve proxy server settings failure.
Payload Parameters for Siebel CRM Deployment	Modified topic. Updated the description of the following parameters: **siebel_admin_password**, **default_user_password**, and **anonymous_user_password**.

What's New in February 2025, CM_25.2

The following table lists the changes in this revision of the documentation to support this release (CM_25.2) of the software.

Topic	Description
Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer	Updated topic. Added support for deploying Siebel CRM on Oracle Compute Cloud @Customer (OC3) using SCM.
Git Repositories for Siebel CRM Deployment	Added topic. Added support for Bring Your Own (BYO) Git, in addition to GitLab, in the Siebel CRM deployment payload. Describes how Git repositories are used and managed in Siebel CRM deployment.
Example Payload to Deploy Siebel CRM	Modified topic. Added the "Example Git Section for BYO-Git" section, to demonstrate how to configure **byo_git** section in the Siebel CRM deployment payload.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added the following parameters specific to BYO Git: **git_type**, **git_user**, **git_protocol_type**, **git_accesstoken**, **git_ssh_private_key**, **git_scm_repo_url**, **git_scm_repo_branch**, **git_scm_flux_folder**, **git_helm_repo_url**, **git_helm_repo_branch**, **git_url**, **git_selfsigned_cacert**.

What's New in January 2025, CM_25.1

The following table lists the changes in this revision of the documentation to support this release (CM_25.1) of the software.

Topic	Description
Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer	Added chapter. Describes the steps to deploy Siebel CRM containers on a Kubernetes cluster using Siebel Installer for SCM.

What's New in December 2024, CM_24.12

The following table lists the changes in this revision of the documentation to support this release (CM_24.12) of the software.

Topic	Description
Rotating Secrets	Added topic. Describes the procedure to rotate secrets.

What's New in November 2024, CM_24.11

The following table lists the changes in this revision of the documentation to support this release (CM_24.11) of the software.

Topic	Description
Overview	Added chapter. Provides an overview of deploying Siebel CRM containers on Kubernetes using Siebel Cloud Manager.

What's New in November 2024, CM_24.10.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.10.1) of the software.

Topic	Description
Uploading Files to the SCM Container Using File Sync Utility	Added topic. Describes how to upload files to the SCM container using File Sync Utility.
Lifting a Siebel CRM Environment Running on Siebel CRM Compliant Operating System	Added topic. Describes the steps to lift a Siebel environment hosted on another machine that has a Siebel CRM compliant operating system running on it.

What's New in October 2024, CM_24.10

The following table lists the changes in this revision of the documentation to support this release (CM_24.10) of the software.

Topic	Description
Mirroring Siebel Base Container Images	Added topic. Describes how to mirror Siebel base container images and manage the user's container registry credentials.
Disabling OCI Monitoring for Siebel CRM	Added topic. Describes how to prevent sending metrics to OCI monitoring.
Disabling Log Analytics in Siebel CRM	Added topic. Describes how to stop streaming logs to OCI Logging Analytics.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added the **`enable_oci_monitoring`** parameter. Updated the description of the **`registry_prefix`** parameter.

Topic	Description

What's New in September 2024, CM_24.8.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.8.1) of the software.

Topic	Description
Downloading and Installing Siebel Cloud Manager	Modified topic. Made updates to the installation procedure.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added the **load_balancer_tls_secret_name** parameter.
Example Payload to Deploy Siebel CRM	Modified topic. Updated the examples in *Example Kubernetes Cluster Sections for BYO-Kubernetes*.

What's New in August 2024, CM_24.8

The following table lists the changes in this revision of the documentation to support this release (CM_24.8) of the software.

Topic	Description
Best Practices for Key Management	Added topic. Provides best practices for key management.
Key Points for Managing Secrets Using Secret Management Products	Added topic. Provides tips for managing secrets using secret management products.
Downloading and Installing Siebel Cloud Manager	Modified topic. Made updates to the installation procedure.
Using BYO File Storage	Added topic. Provides details about the BYO file system, which allows users to use an existing file system and mount target (exports for the file system) during the provisioning of Siebel environment.
Using BYO Kubernetes	Added topic, which contains the following subtopics: • *Notes on OKE (Oracle Container Engine for Kubernetes)* • *Notes on OCNE (Oracle Cloud Native Environment)* • *Notes on Other Kubernetes Cluster*
Payload Parameters for Siebel CRM Deployment	Modified topic. Updated these registry parameters: **registry_url**, **registry_user**, and **registry_password**. Added this registry parameter: **registry_prefix**. Added these infrastructure parameters: **kubernetes_type**, **oke_node_count**, **oke_node_shap**, **memory_in_gbs**, **ocpus**, **oke_cluster_id**, **oke_endpoint**, **oke_kubeconfig_path**, **kubeconfig_path**, **ingress_service_type**, and **ingress_controller_service_annotations**.

Topic	Description
	Added these observability parameters: **storage_class_name**, **local_storage**, and **kubernetes_node_hostname**. Added this database parameter: **whitelist_cidrs**. Added these additional parameters: **mount_target_ip** and **export_path**.
Example Payload to Deploy Siebel CRM	Added the following sections: • *Example Payload when "Do not use Vault" Checkbox is Selected* • *Example Kubernetes Cluster Sections for BYO-Kubernetes*
Use Cases for Enabling Component Groups or Components	Modified topic. Included sample data for sai_quantum.yaml and provided details on how to enable component groups and components.

What's New in July 2024, CM_24.7

The following table lists the changes in this revision of the documentation to support this release (CM_24.7) of the software.

Topic	Description
Running the Siebel Lift Utility in Silent Mode (for Container Mode)	Modified topic. Updated the command and provided flag descriptions.
Running the Siebel Lift Utility in Interactive Mode (for Container Mode)	Modified topic. Updated the command and provided flag descriptions.

What's New in July 2024, CM_24.6.2

The following table lists the changes in this revision of the documentation to support this release (CM_24.6.2) of the software.

Topic	Description
Downloading and Installing Siebel Cloud Manager	Modified topic. Introduces the option to use HTTPS protocol for Siebel Cloud Manager during the installation.
Custom Siebel CRM Metrics	Modified topic. Provides additional examples to use more than one server manager command at the same time to collect metrics.

What's New in July 2024, CM_24.6.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.6.1) of the software.

Topic	Description
Payload Parameters for Siebel CRM Deployment	Modified topic. Added these observability parameters: **oci_log_analytics**, **smc_log_group_id**, **sai_log_group_id**, **ses_log_group_id**, **gateway_log_group_id**, **node_logs_log_group_id**, and **log_source_name**.
Enabling Log Analytics in Siebel CRM Observability	Modified topic. Provided sample code that Siebel CRM deployment payload for SCM should contain, if OCI Log Analytics has to be enabled in a BYOR deployment.

What's New in June 2024, CM_24.6

The following table lists the changes in this revision of the documentation to support this release (CM_24.6) of the software.

Topic	Description
Monitoring Siebel CRM Deployments	New chapter. Describes how to configure the Siebel CRM Observability – Monitoring solution.
Log Analytics in Siebel CRM Deployments	New chapter. Describes how to configure the Siebel CRM Observability – Log Analytics solution.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added these observability parameters: **siebel_monitoring**, **send_alerts**, **siebel_logging**, **enable_oci_log_analytics**, **enable_oracle_opensearch**, **mount_target_private_ip**, **export_path**, **oci_config_path**, **oci_private_api_key_path**, **oci_config_profile_name**, **smtp_host**, **smtp_from_email**, **smtp_auth_username**, **smtp_auth_password_vault_ocid**, and **to_email**.
Troubleshooting Issues Related to Siebel CRM Observability – Monitoring Solution	Added topic. Describes how to debug issues related to the Siebel CRM Observability – Monitoring solution.
Troubleshooting Issues Related to Siebel CRM Observability – Log Analytics Solution	Added topic. Describes how to debug issues related to the Siebel CRM Observability – Log Analytics solution.

What's New in April 2024, CM_24.3.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.3.1) of the software.

Topic	Description
Resubmitting the Environment Creation Workflow	Modified topic. Introduced new functionality and parameters to allow any specific stage and all stages from any specific stage including that stage.
Updating Parameters During Rerun of Environment or Configuration APIs	Modified topic. Provided API example to update environment status as completed.

What's New in February 2024, CM_24.2

The following table lists the changes in this revision of the documentation to support this release (CM_24.2) of the software.

Topic	Description
Use Cases for Changing Log Level While Running PostInstallDB Setup	New topic. Provides information about changes to make to support use cases for changing log levels associated with the process of running PostInstallDBSetup.

What's New in January 2024, CM_24.1.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.1.1) of the software.

Topic	Description
High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM	Modified topic. Added instructions for: • Sourcing of virtual environment and k8sprofile • Tagging git repositories before moving to the latest Siebel CRM version

What's New in January 2024, CM_24.1

The following table lists the changes in this revision of the documentation to support this release (CM_24.1) of the software.

Topic	Description
Cleaning up the Siebel File System	New topic. Provides instructions for cleaning up orphan files in Siebel File System using APIs.

What's New in December 2023, CM_23.12

The following table lists the changes in this revision of the documentation to support this release (CM_23.12) of the software.

Topic	Description
Enabling TLS 1.3 Support in Environments Prior to 23.11	New topic. Provides instructions to enable TLS 1.3 communication from client to server and server tier to server tier.

What's New in November 2023, CM_23.10.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.10.1) of the software.

Topic	Description
Auto-enablement of Siebel Migration Application	New topic. Describes the details related to auto-deployed Siebel Migration application in Siebel CRM environments deployed using Siebel Cloud Manager.

Topic	Description
Troubleshooting Issues Related to Siebel Migration Application in an SCM Deployed Siebel CRM Environment	New topic. Provides troubleshooting tips for migration application usage in SCM deployed Siebel CRM environments.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added the **migration_package_mt_export_path** parameter.

What's New in September 2023, CM_23.8.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.8.1) of the software.

Topic	Description
Terminating SSL/TLS at the Load Balancer (FrontEnd SSL) using SCM	New topic. Describes mechanisms to enable frontend SSL (terminating SSL at the load balancer) with OKE provisioned Load balancer.
Assigning Pods to Nodes - Implementing Affinity and Anti-affinity on OKE using Siebel Cloud Manager	New topic. Describes how to use SCM to restrict Kubernetes pods to desired nodes using affinity/anti-affinity.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added these parameters: **load_balancer_ssl_cert_path**, **load_balancer_private_key_path**, and **load_balancer_private_key_password**.

What's New in August 2023, CM_23.8

The following table lists the changes in this revision of the documentation to support this release (CM_23.8) of the software.

Topic	Description
Using Security Adapters for Siebel CRM	Modified topic.

What's New in July 2023, CM_23.7.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.7.1) of the software.

Topic	Description
Using Security Adapters for Siebel CRM	Modified topic. LDAP over SSL is now supported.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added these parameters: **enable_ssl**, **ldap_wallet_path**, and **ldap_wallet_password**. Modified description for **application_password**.

What's New in July 2023, CM_23.7

The following table lists the changes in this revision of the documentation to support this release (CM_23.7) of the software.

Topic	Description
Using Security Adapters for Siebel CRM	New topic. Describes how to configure security adapters (security profile) provided with Siebel Business Applications.
Payload Parameters for Siebel CRM Deployment	Modified topic. Added these parameters: **security_adapter_type**, **ldap_host_name**, **ldap_port**, **application_user_dn**, **application_password**, **base_dn**, **credentials_attribute_type**, **password_attribute_type**, **roles_attribute_type**, **shared_db_credentials_dn**, **shared_db_username**, **shared_db_password**, **username_attribute_type**, **use_adapter_username**, **siebel_username_attribute_type**, **siebel_admin_username**, **siebel_admin_password**, **anonymous_username**, **anonymous_user_password**, **propagate_change**, **hash_db_password**, **hash_user_password**, **salt_attribute_type**, and **salt_user_password**.

What's New in July 2023, CM_23.6.2

The following table lists the changes in this revision of the documentation to support this release (CM_23.6.2) of the software.

Topic	Description
Downloading and Installing Siebel Cloud Manager	Modified topic. Provided steps to use an existing VCN (Bring Your Own VCN).
Payload Parameters for Siebel CRM Deployment	Modified topic. Added the following parameters for BYO-VCN: • **siebel_lb_subnet_ocid** • **siebel_private_subnet_ocid** • **siebel_db_subnet_ocid** • **siebel_cluster_subnet_ocid** • **vcn_ocid_of_db_subnet**
Example Payload to Deploy Siebel CRM	Modified topic. Added an example payload for a scenario when "Use existing VCN" checkbox is selected.
Using a BYO VCN	New topic: Describes how to use your own VCN in OCI.

What's New in June 2023, CM_23.6

The following table lists the changes in this revision of the documentation to support this release (CM_23.6) of the software.

Topic	Description
About Siebel CRM Upgrade Factory	New topic. Introduces Siebel CRM Upgrade Factory that delivers an automated development upgrade including the quick setup of a development environment and the efficient transition from Siebel CRM 8.0 and above to the latest Siebel CRM Release Update.
Customizing the Configuration	Modified topic. Provided steps to • Customize Siebel CRM configuration that require changes in helm charts repository • Customize Siebel CRM Kubernetes deployment parameters that require changes in the Cloud Manager repository
Payload Parameters for Siebel CRM Deployment	Modified topic. Updated the descriptions for **`cpu`** and **`memory`** payload parameters.
Example Payload to Deploy Siebel CRM	Modified topic. Updated payload examples with additional size parameters.
Updating Parameters During Rerun of Environment or Configuration APIs	Modified topic. Updated payload examples with additional size parameters.
Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers	Modified topic. Updated the use case for adding resources to individual Siebel servers.

What's New in May 2023, CM_23.5.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.5.1) of the software.

Topic	Description
High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM	New topic. Provides the steps required to install the latest monthly updates in a Siebel CRM on OKE environment deployed by Siebel Cloud Manager.
Payload Parameters for Siebel CRM Deployment	• Added the **`gateway_cluster_replica_count`** parameter. • Updated the description for the **`db_home_admin_password`** parameter.
Example Payload to Deploy Siebel CRM	Updated the description for the **`db_home_admin_password`** parameter.

What's New in May 2023, CM_23.5

The following table lists the changes in this revision of the documentation to support this release (CM_23.5) of the software.

Topic	Description
Using Vault for Managing Secrets	New topic. Introduces Vault integration for Siebel Deployment using Siebel Cloud Manager.
Downloading and Installing Siebel Cloud Manager	Modified topic. Details regarding Vault usage have been added.
Executing the Payload to Deploy Siebel CRM	Modified topic. Password location for basic authentication has been updated.

Topic	Description
Payload Parameters for Siebel CRM Deployment	Modified topic. Payload Parameters and Descriptions have changed. • New Additions: siebel_keystore_password, siebel_truststore_password, db_admin_username, db_admin_password • Renamed: siebel_admin_username(admin_user_name), siebel_admin_password(admin_user_password) • Updates: table_owner_user, table_owner_password, default_user_password, anonymous_user_password, admin_password
About Siebel Cloud Manager	Modified topic. Expands the list of third-party products and operators with brief descriptions.

What's New in April 2023, CM_23.3.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.3.1) of the software.

Topic	Description
Checklist for Creating a BYOR Deployment	New topic. Before deploying a BYOR environment, you need to go through this checklist consisting of various steps to ensure that you have a smooth deployment.
Downloading and Installing Siebel Cloud Manager	Modified topic. Describes how to use Cloud Manager behind a proxy.

What's New in March 2023, CM_23.2.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.2.1) of the software.

Topic	Description
Additional Administrative Tasks in Siebel Cloud Manager	Modified topic. Option for *Updating Parameters During Rerun of Environment or Configuration APIs*.
Payload Parameters for Siebel CRM Deployment	Modified topic. New parameters included for VCN traffic routing.

What's New in February 2023, CM_23.2

No new feature introduced, contains only bug fixes and minor internal enhancements.

What's New in February 2023, CM_23.1.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.1.1) of the software.

Topic	Description
Deploying Siebel CRM on OCI	Modified topic. Introduces the ability to use existing database available with user while with all other resources (such as OKE, File System, Mount Target etc.) are created by Siebel Cloud Manager during Siebel Deployment.

What's New in January 2023, CM_23.1

The following table lists the changes in this revision of the documentation to support this release (CM_23.1) of the software.

Topic	Description
Notes on OKE (Oracle Container Engine for Kubernetes)	Modified topic. Multiple Siebel environments can be provisioned in the same OKE cluster when the "Use Existing Resource" option is selected while creating the CM instance.

What's New in December 2022, CM_22.12.1

The following table lists the changes in this revision of the documentation to support this release (CM_22.12.1) of the software.

Topic	Description
Using Custom Keystore	New topic. Introduces the feature to use custom Keystore and Truststore.
Downloading and Installing Siebel Cloud Manager	Modified topic. Introduces feature to use existing resources (like VCN, OKE, Database, Mount Target etc) rather than have Cloud Manager must create these resources anew for Siebel Deployment.
Deploying Siebel CRM on OCI	Modified topic. Introduces feature to use existing resources (like VCN, OKE, Database, Mount Target etc) rather than have Cloud Manager must create these resources anew for Siebel Deployment.

What's New in July 2022, CM_22.7.0

The following table lists the changes in this revision of the documentation to support this release (CM_22.7.0) of the software.

Topic	Description
Downloading and Installing Siebel Cloud Manager	Modified topic. • A new Permissions step allows you to specify the permissions type for the Siebel Cloud Manager instance, either Instance Principal or User Principal. • Shape configuration for a Siebel Cloud Manager instance has been renamed as CloudManager Instance Configuration and includes specifying whether to use a private IP address (the default) or a public IP address.

Topic	Description
Troubleshooting a Siebel Cloud Manager Instance or Requested Environment	Renamed topic (from "Reviewing and Troubleshooting a Requested Environment") and added a new subtopic *Troubleshooting a Siebel Cloud Manager Instance*.
Updating Siebel Cloud Manager with a New Container Image	Modified topic. This procedure has changed due to the migration.sh script, which is new in this release.
Use Cases for Making Incremental Changes	Modified topic. Added use cases for adding or updating web artifacts or other Siebel artifact files.

What's New in June 2022, CM_22.5.2

The following table lists the changes in this revision of the documentation to support this release (CM_22.5.2) of the software.

Topic	Description
Downloading and Installing Siebel Cloud Manager	Modified topic. Configuring the Siebel Cloud Manager stack now supports specifying the node shape for the Cloud Manager instance, the number of OCPU cores, and the memory in gigabytes.
Payload Parameters for Siebel CRM Deployment *Example Payload to Deploy Siebel CRM*	Modified topics. For the DBCS_VM database type, the cpu_count parameter is now provided. This parameter is required where the shape is of flex type.

What's New in June 2022, CM_22.5.1

The following table lists the changes in this revision of the documentation to support this release (CM_22.5.1) of the software.

Topic	Description
Troubleshooting Siebel Lift Utility Execution	Modified topic. Information is provided about error codes for Siebel Lift utility introspection.
Using Advanced Network Configuration *Updating Siebel Cloud Manager with a New Container Image*	Modified topics. Information is provided about configuring the Siebel Cloud Manager private subnet and mount target. The mount target is no longer part of the subnet siebel_private_subnet_cidr and must be migrated in a one-time step when you update Cloud Manager to a new container image. The subnet siebel_atp_subnet_cidr has been renamed to siebel_db_subnet_cidr. This subnet applies to all database choices.
Payload Parameters for Siebel CRM Deployment *Example Payload to Deploy Siebel CRM* Multiple topics modified	Modified topics. Siebel CRM deployments on OCI now support the Database Service for Oracle Database (DBCS_VM), also referred to as Oracle Database Cloud Service, as well as Oracle Autonomous Database (ATP).

Topic	Description
	In the payload, use the db_type parameter to specify either ATP or DBCS_VM. Different database parameters are used, depending on your selection. Parameters for ATP are now under the atp section and have been renamed to remove the atp_ prefix.

What's New in May 2022, CM_22.4.1

The following table lists the changes in this revision of the documentation to support this release (CM_22.4.1) of the software.

Topic	Description
About URLs for Siebel CRM Deployments on OCI Multiple topics modified	New and modified topics. Some of the base URL elements have changed in this release of Siebel Cloud Manager. Specifically, api/v1/environments has changed to scm/api/v1.0.
Customizing Configurations Prior to Greenfield Deployment Multiple topics modified	New and modified topics. For a greenfield deployment, you can optionally decouple the configuration and provisioning stages, for the purpose of customizing the configuration before you provision the environment.

2 Overview

About this Chapter

This chapter provides an overview of how you can deploy Siebel CRM containers on Kubernetes using Siebel Cloud Manager (SCM). It includes the following topics:

- *About Siebel Cloud Manager*
- *About Siebel CRM Upgrade Factory*

Cloud-native is a paradigm for building, deploying, and managing applications that provides the benefits of containerization, dynamic scaling, automation via DevOps, and resiliency. Kubernetes is a key enabler of this modern approach. It is an open-source platform that helps you to automate the deployment and management of containerized applications by providing mechanisms for orchestrating containers, making it easier to manage large-scale applications such as Siebel CRM.

You can use SCM to deploy Siebel CRM containers on Kubernetes. SCM provides the following features to help you move toward a cloud native deployment of Siebel CRM:

- Modern observability with metrics monitoring and log analytics
- Declarative GitOps
- Modern tooling and innovation
- Deployment flexibility
- Retention of existing customizations and functionalities
- Run-time infrastructure optimization and scaling

This guide provides information about SCM and explains how you can:

- Set up SCM and, optionally, download the Siebel Lift utility.
- Deploy Siebel CRM using SCM.

About Siebel Cloud Manager

SCM is a platform for deploying and managing Siebel CRM containers on Kubernetes clusters, either on premises or in the cloud. It is built on modern cloud native tooling such as Terraform, Ansible, and so on. You can use SCM to:

- Automate the deployment and management of Siebel CRM:
  - On a variety of platforms such as:
    - Cloud Native Computing Foundation (CNCF) compliant Kubernetes cluster such as:
      - Oracle Kubernetes Engine (OKE) on Oracle Cloud
      - Oracle Cloud Native Environment (OCNE) on premises or on cloud
      - Red Hat OpenShift, and so on
    - Oracle Compute Cloud@Customer (OC3) in your own data center.
  - With the option to either:

    - Migrate existing Siebel CRM environments, on premises or on cloud, to CNCF compliant Kubernetes clusters using the Siebel Lift utility or
    - Create a new greenfield deployment of Siebel CRM. You can perform the greenfield deployment using the default configuration or you can customize the configuration before deployment.
- Run and maintain the Siebel CRM Enterprise.
- Deploy Siebel CRM Upgrade Factory which simplifies the Siebel CRM application upgrade process by allowing you to upload your customized, pre-IP2017 repository to run an upgrade and Incremental Repository Merge (IRM) process on Oracle Cloud Infrastructure (OCI).
- Deploy Open Integration on a Kubernetes cluster.
- Deploy AI Connectors on a Kubernetes cluster. For more information, refer to the "Deploying Siebel AI Connectors Using Siebel Cloud Manager" topic in the Siebel Installation Guide.

For more information on CNCF compliant Kubernetes, refer *CNCF online documentation*.

For more information on OC3, refer *Oracle Cloud Compute @Customer*.

> **Note:** Read this entire document and related documents and familiarize yourself with the concepts, tools, and methods for deploying Siebel CRM or other applications on CNCF compliant Kubernetes clusters mentioned above or on OC3. Many of the cloud computing principles and capabilities described by the CNCF apply to deploying Siebel CRM containers (referred as Siebel CRM in the document) using SCM. Apart from the Siebel-specific particulars, some such information is beyond the scope of this document.

This topic contains the following sections:

- *Siebel Lift Utility*
- *Tools for Deploying Siebel CRM using SCM*
- *Running Workloads with Custom Non-Root User ID and Group ID*
- *Managing Ingress using Traefik*
- *Deploying Siebel CRM Using SCM*
- *Operating Your Siebel CRM Environment*

Siebel Lift Utility

The Siebel Lift utility is an SCM-integrated tool that allows you to migrate existing on-premises Siebel CRM deployments to a CNCF compliant Kubernetes cluster or OC3 in your data center. The Siebel Lift utility:

- Creates deployment kits consisting of artifacts derived from an existing on-premises deployment of Siebel CRM.
- Reads the stored artifacts and uploads them to an object storage to populate the migration pipeline for Siebel CRM deployment.

After you create and upload Siebel CRM deployment kits, specify the deployment parameters in the deployment payload and submit it to SCM using a REST request. SCM accesses the Siebel CRM artifacts in the object storage and performs the tasks to deploy your applications.

> **Note:** Greenfield deployments of Siebel CRM don't use the Siebel Lift utility.

For information about downloading and running the Siebel Lift utility, see *Downloading and Running the Siebel Lift Utility*.

For information about downloading and installing SCM, see *Downloading and Installing Siebel Cloud Manager*.

Tools for Deploying Siebel CRM using SCM

SCM and the Siebel Lift utility use several third-party products and custom operators to support deployment pipeline operations. These products include the following:

- **Flux (Flux Operator)**: Flux is a continuous deployment tool and a CNCF Graduated project pioneered by Weaveworks. SCM uses Flux to synchronize the Git repository with the Kubernetes cluster and automate Siebel CRM deployment by using a GitOps approach. Flux helps ensure that the Siebel CRM deployment in the cluster matches the configuration defined in the Git repository. When Flux detects changes in the Git repository, it synchronizes the changes with the deployment.
- **Helm**: SCM uses Helm charts to deploy Siebel CRM and its supporting components.
- **kubectl**: You can use kubectl to interact with the Kubernetes cluster and perform administration tasks. For example, you can use `kubectl` to open a session in a Kubernetes pod and run commands.
- **Config Operator**: The Config Operator performs Siebel CRM configuration through Siebel Management Console (SMC) REST APIs. A Kubernetes job managed by Helm initiates this operator. This job is triggered when:
  - A successful Siebel CRM database connection is established.
  - SMC is running.

  The Helm Git repository contains the Siebel CRM configuration and deployment definition YAMLs. The `paramconfig` directory in the Helm Git repository contains the profile definition of the Siebel CRM infrastructure. The Config Operator uses this profile definition to perform Siebel CRM configuration.

  For more information on Git repositories, see *Git Repositories for Siebel CRM Deployment*.
- **Siebel Operator (An Incremental Operator)**: The Siebel Operator is a *Metacontroller* based custom controller that performs incremental changes in an existing Siebel CRM deployment created using SCM. It facilitates GitOps to create a powerful, automated, and declarative approach to managing the Siebel CRM resources. For each upgrade or Flux reconcile that's performed, configure job runs to validate the application configuration.

  The Siebel Operator monitors the `paramconfig` ConfigMaps, detects incremental changes, deploys runtime additions at the Enterprise, Siebel CRM Server, and component levels, and restarts the Siebel CRM Server when required. The Siebel Operator detects and synchronizes ConfigMap changes when you add the following incremental changes in `paramconfig`:

  - Enable a component group
  - Enable / disable a component
  - Enterprise parameter changes
  - Server parameter changes
  - Component parameter changes
  - Adding new named subsystem
  - Adding a new component definition
  - Adding new components to the server
  - Adding AI parameters

- Adding a new Siebel server or AI
- Changing log level

- **Artifactory server:** The Artifactory server is part of SCM and provides an artifact repository for Siebel CRM deployment artifacts. It stores and manages deployment artifacts and provides access to the files required for different deployment scenarios. It plays an important role in the deployment process:
  - In a greenfield or fresh Siebel CRM deployment, the Artifactory server acts as the source for deployment artifacts.
  - In lift-and-shift scenarios, the Artifactory server helps migrate existing Siebel CRM artifacts. The Siebel Lift utility extracts these artifacts from an existing Siebel CRM environment. You can optionally store the artifacts on an NFS path, and the Artifactory server mounts that NFS location to make the files available for deployment.

Running Workloads with Custom Non-Root User ID and Group ID

Starting with release 26.6, SCM supports running SCM resources, Siebel CRM workloads, and observability workloads with custom non-root user ID and group ID. This capability helps you meet Kubernetes security requirements and aligns with OpenShift security practices, where workloads are expected to run with the `restricted-v2` Security Context Constraint (SCC) or the least-privileged SCC that supports the workload.

Although most workloads can run with restricted permissions, some components require additional SCCs because they perform privileged operations, such as image building, host log collection, or access to OpenSearch storage. You must configure the required SCCs based on your deployment security model.

> **Note:** Oracle recommends adopting the enhanced security model with custom non-root user ID and group ID at the earliest opportunity. This approach provides improved security and aligns with future platform requirements. Support for the default security model is planned to be withdrawn in a future release for OpenShift clusters.

For more information, see *Deploying Siebel CRM on OpenShift using SCM*.

Managing Ingress using Traefik

SCM deploys ingress controller as part of a Siebel CRM deployment on Kubernetes. Before release 26.3, SCM deployed the NGINX Ingress Controller. Kubernetes SIG Network has retired NGINX Ingress, which reached end of life in March 2026.

Starting with release 26.3, SCM deploys Traefik as the default ingress controller out of the box for Siebel CRM on Kubernetes. If you run a SCM version earlier than 26.3, update to SCM 26.3 and use the migration option to move from NGINX Ingress to Traefik. This migration modernizes the ingress layer and better aligns with Oracle's cloud-native, GitOps-driven approach. It simplifies ongoing maintenance and improves integration with the Kubernetes ecosystem.

For more information, see the relevant sections in this document.

Deploying Siebel CRM Using SCM

To deploy a Siebel Enterprise:

- The Siebel CRM functionality is provided in the following base containers:
  - Siebel Gateway (CGW), sometimes called Cloud Gateway, that can be set up to operate in either standalone mode or cluster mode based on your scalability requirements.
  - Siebel Application Interface (SAI).
  - Siebel Server (SES), sometimes called Siebel Enterprise Server.
- The Siebel CRM database is migrated as an Autonomous Database for Transaction Processing in shared mode or Database Service for Oracle Database in virtual machine mode.

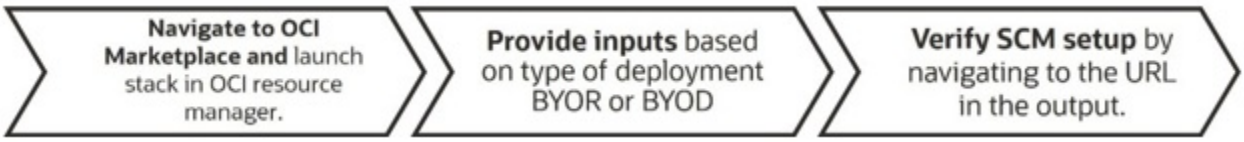
You can deploy Siebel CRM in different ways using SCM, depending on the following infrastructure provisioning options:

Infrastructure Provisioning Option	Applicable to Siebel CRM Deployment On	Description
Allow SCM to create the infrastructure resources	OKE on Oracle Cloud Infrastructure (OCI)	SCM creates and manages the infrastructure resources required for deploying Siebel CRM such as the database, OKE, mount target, file system, and so on. You can allow SCM to create: • All the required resources or • All resources except database (BYOD only)
Bring Your Own Resources (BYOR)	• CNCF certified Kubernetes clusters such as OKE on Oracle Cloud, OCNE cluster, Red Hat Open Shift and so on • Oracle Cloud Compute @Customer (OC3) in your data center	You must bring all resources such as an existing mount target, file system, OKE, and database, and provide the details of the resources in the Siebel CRM deployment payload.

The following are the high-level steps to deploy Siebel CRM using SCM (these are described in detail later in this document):

1. Set up Git repositories to store SCM templates, release YAMLs and Helm charts. You can bring your own existing Git repositories hosted on any standards-compliant Git distribution, such as OCI DevOps, GitHub, GitLab, and so on, or allow SCM to create the repositories as part of Siebel CRM provisioning.

2. Set up SCM:
   a. On OCI:



   b. On a Kubernetes cluster on premises or in the cloud or in your data center on OC3:



3. Deploy Siebel CRM:
   a. Create the deployment payload based on the infrastructure provisioning requirements, for a:
      i. Fully SCM provisioned environment for Siebel CRM deployment, provide the infrastructure resource details in the deployment payload by referring to the SCM guide.
      ii. BYOD deployment, provide the details of the existing database and the details of the infrastructure resources to be created by SCM.
      iii. BYOR, provide the details of all existing resources.
   b. Use the `/environment` API to create the Siebel CRM deployment using the POST method.
   c. Check the progress of the deployment and verify access to Siebel CRM on successful completion.

Operating Your Siebel CRM Environment

SCM uses pipelines to streamline the release cycle, reduce manual errors, and improve overall efficiency in the Siebel CRM application development and deployment process. The following pipelines are used in your environment:

- The migration pipeline, or continuous-integration pipeline, builds and stages software for making development changes or preparing for deployment. This pipeline is managed by the Siebel Migration application. (For more information about Siebel Migration, refer Siebel Database Upgrade Guide.)
- The deployment pipeline, or continuous-deployment pipeline, stages software for application deployment. This pipeline is managed by SCM, using the GitOps model of continuous delivery.

The overall flow of continuous integration and continuous delivery (CI/CD) is a multi-stage, cyclical process that's largely automated but includes various user interaction points, depending on your use case. In all cases, the migration pipeline is separated from the deployment pipeline.

About Siebel CRM Upgrade Factory

Siebel CRM Upgrade Factory provides an automated development upgrade process and helps you set up a development environment quickly. You can use this approach to simplify the Siebel CRM application upgrade process.

With Siebel CRM Upgrade Factory, you can upload your customized, pre-IP2017 repository and run the upgrade and IRM process on OCI. The development upgrade process creates a merged design repository. You can continue to use this

repository as your new Siebel CRM development environment in your OCI tenancy, or export and import it to your on-premises environment for continued development.

You can use Siebel CRM Upgrade Factory to:

- Perform Siebel CRM upgrades in a cost-efficient, low-risk manner within a shorter time frame.
- Dramatically reduce lengthy provisioning cycles.
- Easily repeat development upgrades multiple times using the DevOps pipeline.
- Flexibly move your application with business customizations intact to OCI, if needed by the business.
- Monitor and evaluate upgrade issues rapidly.

For more information about Siebel CRM Upgrade Factory, see *Database Upgrade Guide*.

For detailed information about Siebel CRM that might be relevant to your deployment tasks, see also *Siebel Update Guide* for your release and *Siebel Bookshelf* documents such as *Siebel Installation Guide* or *Siebel Database Upgrade Guide*.

You can also refer to the documentation for the following products or modules, where applicable:

- **Oracle products.** OCI (which includes OKE, Oracle Resource Manager, and many other modules and features), Oracle Enterprise Linux, Oracle Database, and other products.
- **Third-party products.** Products such as Ansible, Podman, Flux, GitLab, Helm, Kafka, Kubernetes, YUM, or other applicable products.

Oracle doesn't certify third-party container management or cloud deployment tools for the uses described here. For more information, see CNCF and other applicable resources.

3 Deploying Siebel CRM on OCI using Siebel Cloud Manager

About this Chapter

This chapter describes how to use Siebel Cloud Manager (SCM) to deploy Siebel CRM on Oracle Cloud Infrastructure (OCI). It includes the following topics:

- *Overview of Deploying Siebel CRM on OCI*
- *Requirements and Limitations*
- *High-Level Steps to Deploy Siebel CRM Using SCM*
- *Creating a Compartment*
- *Git Repositories for Siebel CRM Deployment*
- *Using Vault for Managing Secrets*
- *Downloading and Installing Siebel Cloud Manager*
- *Upgrading Oracle Linux on the SCM VM*
- *About URLs for Siebel CRM Deployments on OCI*
- *Uploading Files to the SCM Container Using File Sync Utility*
- *Mirroring Siebel Base Container Images*
- *Reducing the Ingress Range for Siebel Cloud Manager*
- *Using Advanced Network Configuration*
- *Customizing Configurations Prior to Greenfield Deployment*
- *Deploying Siebel CRM on OCI*
- *Additional Administrative Tasks in Siebel Cloud Manager*
- *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*
- *Updating Siebel Cloud Manager with a New Container Image*
- *Updating Ingress Controller*
- *Migrating (Lift-And-Shift) Existing Siebel CRM Deployments*
- *Removing a Siebel CRM Deployment on OCI*
- *Making Incremental Changes to Your Siebel CRM Deployment on OCI*
- *High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM*
- *Enabling TLS 1.3 Support in Environments Prior to 23.11*
- *Rotating Secrets*
- *Assigning Pods to Nodes - Implementing Affinity and Anti-affinity on OKE using Siebel Cloud Manager*
- *Adding and Overriding Environment Variables*
- *Cleaning up the Siebel File System*

Note: This document was first published in February 2022. Going forward, the information in this document is expected to be updated and expanded, as needed.

Overview of Deploying Siebel CRM on OCI

You can deploy Siebel CRM on OCI using SCM.

This chapter describes the steps required to do the following tasks:

- Set up SCM in an OCI tenancy.
- Deploy Siebel CRM on OCI.

You use SCM and other tools to deploy Siebel CRM on OCI. For more information, see *Overview* chapter.

Requirements and Limitations

The following requirements and limitations apply to SCM. This information may change in future updates.

General Requirements

Before you use SCM or deploy Siebel CRM, make sure that you meet the following requirements:

- You must run Siebel CRM 18.12 or later to migrate Siebel CRM to OCI.
- You must keep the on-premises Siebel CRM environment running when you run the Siebel Lift utility.
- You must have an OCI tenancy with Compute quota and manage privileges for Oracle Kubernetes Engine (OKE) and File Storage service (FSS).
- You must use virtual machines that run Oracle Enterprise Linux 7 or 8 for Siebel CRM deployments on OCI.
- You must have access to a standards-compliant Git distribution, such as OCI DevOps Service, GitHub, GitLab, or Bitbucket. You must use only one Git instance for the main OCI compartment where you use SCM. For more information, see see *Git Repositories for Siebel CRM Deployment*.
- You must create an OCI compartment that supports at least two child compartment levels. For more information, see *Creating a Compartment*.

Limitations

The following language limitations apply when you deploy Siebel CRM using SCM:

- You can use any language that Siebel CRM supports for lift-and-shift deployments.
- You can use only U.S. English (ENU) for greenfield deployment of Siebel CRM with SCM. SCM instructions currently are in U.S. English (ENU).

Oracle Database Requirements

Before you use SCM, make sure that you meet the following Oracle Database requirements:

- You must have Oracle Database 19c on OCI for Siebel CRM deployment. The Siebel CRM database is migrated into OCI as an Autonomous Database for Transaction Processing in shared mode or as an Oracle Database Cloud Service in virtual machine mode.
- You must mount the staging location for database artifacts, you create, on the computer where Oracle Database is installed.
- You must assign create directory privilege, database owner privilege, and read privilege for all dictionaries to the table owner user.
- You must have a compatible version of Oracle Database client installed where Siebel CRM is installed on premises to create database artifacts to migrate the Siebel CRM database when using the Siebel Lift Utility.
- You must ensure that Oracle Database client includes the data pump utilities (`expdp`) in the `bin` directory.

On the Oracle Database client computer, make sure that you meet the following requirements:

- You must use the Administrator installation type for Oracle Database client.
- You must configure the `tnsnames.ora` file with the required TNS settings for the installed Oracle Database.
- On Linux computers, you must set the `TNS_ADMIN` environment variable to the directory that contains the SQL*Net configuration files, including `sqlnet.ora` and `tnsnames.ora`.
- On Linux computers, you must add the user who runs the Siebel Lift utility to the Database Administrator (DBA) group.

High-Level Steps to Deploy Siebel CRM Using SCM

This topic describes the high-level steps to deploy Siebel CRM using SCM. Before you review the deployment steps, understand how SCM uses the term **BYO**.

The term BYO means Bring Your Own. In SCM, BYO refers to existing resources that you provide for deployment. For example, BYOD means Bring Your Own Database.

After you set up SCM, you can deploy Siebel CRM applications in different ways based on the infrastructure information that you provide in the Siebel CRM deployment payload. You can deploy Siebel CRM using one of the following options:

1. Brings your own resources (fully BYOR): Select **Use existing resource** when you provision SCM, and provide details of all required resources in the deployment payload, including the existing mount target, file system, OKE cluster, and database. SCM uses this payload information to create Siebel CRM deployments.
2. Let SCM create all resources: Do not select **Use existing resource** when you provision SCM. SCM creates and configures the infrastructure required for the Siebel CRM deployment, such as the database, OKE cluster, mount target, file system, and other required resources.
3. Bring your own database (BYOD) only: Do not select **Use existing resource** when you provision SCM but provide the existing database information in the deployment payload. SCM creates the remaining infrastructure resources, such as the mount target, file system, and OKE cluster. Make sure that SCM and OKE can connect to the database.

The following are the high-level steps to deploy Siebel CRM on OCI (these are described in detail later in this document):

1. Set up the Git repository:

- You can bring your own existing Git repositories hosted on any standards-compliant Git distribution, such as OCI DevOps, GitHub, GitLab, Bitbucket, and so on, or
- If you want SCM to create the repositories during Siebel CRM provisioning:
  - Set up a GitLab instance.
  - Generate a private key and an access token.

2. Set up SCM:

   Navigate to OCI and in marketplace applications, select SCM and launch a stack in OCI resource manager.

   If you use a local Git instance that is not a hosted service with a CA-signed certificate, copy the private key from the Git instance to a secure location in the SCM container.

   Provide inputs based on your deployment model, such as BYOR, BYOD, or fully SCM-provisioned deployment.

   For BYOR deployments, select **Use existing resource** and provide existing resource details, such as the VCN, subnet for launching the instance, policies and so on.

   After the stack job completes successfully, copy the URL from the job output and open it in a browser to verify the SCM setup.
3. Deploy Siebel CRM:

   a. Create a payload according to your chosen infrastructure provisioning option:
      - For a fully SCM-provisioned Siebel CRM environment, provide the required infrastructure details in the payload. For more information, see *Deploying Siebel CRM on OCI*.
      - For BYOD, provide the details of the existing database and infrastructure resource.
      - For BYOR, provide the details of all existing resources.

      > **Note:** In case of BYOR or BYOD make sure connection exists between:
      > - The database and the OKE cluster.
      > - The mount target and the OKE cluster.

   b. Use the `/environment` API with the POST method to create the Siebel CRM deployment.
      - Review the API response for validation errors or successful environment information.
      - Use the environment entity in the response to monitor deployment progress.
   c. Review the API response for validation errors or successful environment information. Use the environment entity in the response to monitor deployment progress.

Provisioning Status, Logs, and Retry Behavior

SCM provisions the Siebel CRM environment in a series of stages until the SMC and component URLs are published. If a stage fails, SCM marks the stage with a failed status.

You can use the log links in each stage to diagnose issues and take corrective action. Because all stages are idempotent, you can rerun the workflow by using the environment PUT method to restart all stages.

Creating a Compartment

Before you can install SCM, you must create a suitable compartment for your deployment within your OCI tenancy. The user creating the compartment must have the necessary access rights to be able to create the compartment. Hierarchically, the compartment you create in your OCI tenancy must support at least two child compartment levels. If necessary, the person who set up your tenancy can create the compartment for you.

After you have created the compartment, copy the OCID of the compartment for future reference. Now you can create the stacks for GitLab and SCM within this compartment.

For more information, see relevant documentation for OCI. For example, see *Overview of Oracle Cloud Infrastructure Identity and Access Management* for information about creating compartments, managing access rights to compartments, and more. See also *Requirements and Limitations*.

Git Repositories for Siebel CRM Deployment

GitOps is a paradigm that uses Git to declaratively automate deployments and improve operational efficiency by managing infrastructure and application code.

SCM uses Git repositories to store the configuration of each deployment that it performs. In case of lift and shift, Git repositories store the artifacts sourced from the source environment using the Siebel Lift utility, and then the configuration files are accessed from the Git repositories to perform the actual deployment. Git repositories also store the configuration artifacts for the greenfield use case, including the one described in *Customizing Configurations Prior to Greenfield Deployment*.

Each Siebel CRM deployment requires the following two Git repositories:

- SCM repository to store SCM templates and release YAMLs.
- Helm chart repository to store Helm charts. It also holds the details of the charts installed and upgrades.

You can configure Git repositories in one of the following ways:

- Use existing repositories. For more information, see *Using Existing Git Repositories*.
- Let SCM create repositories. For more information, see *Creating Repositories Through SCM*.

Additionally, to enable Git commit signing to add a cryptographic signature to each Git commit, see *Using Git Commit Signing*.

Using Existing Git Repositories

This topic contains the following sections:

- *Configuring SSH Host-Key Verification*
- *Creating Standards-Compliant Git Repositories*

If you've existing Git repositories hosted on any standards-compliant Git distribution, such as OCI DevOps, GitHub, GitLab, Bitbucket, and so on, you can use it to provision Siebel CRM through SCM. To use existing Git repositories, in the Siebel CRM deployment payload:

1. Set the `git_type` parameter to `byo_git`.
2. Configure the `byo_git` section. For details of the parameters to include in the `byo_git` section, see *Payload Parameters for Siebel CRM Deployment*.

   In this case, SCM will override the data in the Git repositories; that is, it will delete existing data in the repositories and then commit SCM data.

   For an example, see the payloads in the "Example Git Section for BYO-Git" section of the *Example Payload to Deploy Siebel CRM* topic.

Configuring SSH Host-Key Verification

If you use SSH for a BYO Git deployment, you must provide a verified SHA-256 host-key fingerprint for each Git host. SCM uses the fingerprint to verify the Git server identity and creates a resource-specific `known_hosts` file:

```
/home/opc/siebel/<ENV_ID>/known_hosts
```

This configuration prevents interactive SSH trust prompts. Git operations fail when SCM cannot verify the configured Git server identity.

> **Note:** You must obtain each fingerprint from the Git server administrator, or another trusted out-of-band source. Do not accept an unverified interactive SSH prompt.

When `git_protocol_type` is set to `ssh`, configure `git_ssh_host_key_sha256` under the `infrastructure.git.byo_git` section. For example:

```
{
    "git_protocol_type": "ssh",
    "git_ssh_private_key": "/home/opc/.ssh/id_rsa",
    "git_ssh_host_key_sha256": {
        "git.example.com": "SHA256:<verified-fingerprint>"
    }
}
```

Use the host name as the key when the Git server uses SSH port 22. For a nondefault port, use `host:port` as the key:

```
{
    "git_ssh_host_key_sha256": {
        "git.example.com:2222": "SHA256:<verified-fingerprint>"
    }
}
```

> **Note:** You don't need to configure `git_ssh_host_key_sha256` when `git_protocol_type` is set to `http`, including when you use HTTPS repository URLs.

Supported SSH Repository URL Formats

SCM supports the `ssh` and `http` Git protocol types. When you use SSH, SCM supports the following repository URL formats:

Git Service	Repository URL Format	User Handling
OCI DevOps	`ssh://host/path/to/repository`	SCM uses the `git_user` value from the payload when the URL does not include a user.
GitLab standard SSH	`ssh://git@gitlab.example.com/group/repository.git`	SCM uses the Git user specified in the URL.
GitLab SCP-style SSH	`git@gitlab.example.com:group/repository.git`	SCM uses the Git user specified in the URL.

For HTTPS Git URLs, set `git_protocol_type` to `http`. Provide the applicable token and, when required, `git_selfsigned_cacert`.

Configuring SSH for Manual Git Operations

SCM automatically applies the required SSH settings for Git operations that it performs. Before you manually run a remote Git command from an SCM-managed working copy, you must set the `GIT_SSH_COMMAND` environment variable when `git_protocol_type` is set to `ssh`.

To set `GIT_SSH_COMMAND` in the active shell:

```
export GIT_SSH_COMMAND="ssh -F /home/opc/.ssh/config \
    -o IdentityFile=<git_ssh_private_key_path> \
    -o IdentitiesOnly=yes \
    -o StrictHostKeyChecking=yes \
    -o UserKnownHostsFile=/home/opc/siebel/<ENV_ID>/known_hosts \
    -o User=<git_user>"
```

In the above example:

- Replace `<ENV_ID>`, `<git_ssh_private_key_path>`, and `<git_user>` with the values for the deployed environment.
- Add the `-p <port>` option for a nondefault SSH port.

> **Note:** Do not set `GIT_SSH_COMMAND` when `git_protocol_type` is set to `http`, including when the repository URL starts with `https://`. HTTP and HTTPS Git repositories continue to use the configured access token or credentials. When applicable, they also use `git_selfsigned_cacert`. SSH keys, SSH fingerprints, and `known_hosts` files do not apply.

Creating Standards-Compliant Git Repositories

Optionally, you can also create Git repositories on any standards-compliant Git distribution, such as OCI DevOps, GitHub, GitLab, Bitbucket, and so on before using them to provision Siebel CRM through SCM. For example, you can create Git repositories in OCI DevOps as follows:

1. Log in to OCI Console.
2. Create an OCI DevOps project. For detailed steps to create an OCI DevOps project, refer *Creating a Project*.
3. Create repositories under the OCI DevOps project. For detailed steps to create repositories, refer *Creating a Repository*.

Creating Repositories Through SCM

If you want SCM to create Helm chart and SCM repositories during Siebel CRM provisioning, you must install GitLab Community Edition (CE) and configure the Siebel CRM deployment payload to use GitLab as the Git repository, as follows:

1. Download, install, and configure GitLab CE as follows:
   a. Enable and start the OpenSSH server daemon.
   b. Add the GitLab package repository.
   c. Install GitLab CE.
   d. Configure GitLab CE.

   For detailed steps to install GitLab CE, refer *Install the Linux package on AlmaLinux and RHEL-compatible distributions*.
2. Configure HTTPS for secure communication. For steps to configure HTTPS, refer the "Configure HTTPS manually" section under *Configure SSL for a Linux package installation*.
3. Set the `git_type` parameter to `gitlab`.
4. Configure the `gitlab` section. For details of the parameters to include in the `gitlab` section, see *Payload Parameters for Siebel CRM Deployment*.

   You must ensure that GitLab is accessible through HTTP/HTTPS for API access to create or delete repositories.

   For an example payload, see the payloads in the *Example Payload to Deploy Siebel CRM* topic.

Using Git Commit Signing

This topic contains the following sections:

- *Signing Customer Git Commit*
- *Rotating Git Signing Keys*

You can enable Git commit signing to add a cryptographic signature to each Git commit. The signature proves that the change was made by a trusted customer or SCM signing identity.

SCM uses the signature so that Flux applies repository changes only when the latest commit is signed by a trusted GPG key.

SCM Git commit signing supports GPG (GNU Privacy Guard) keys. Use an RSA GPG key with a minimum key size of 3072 bits. You can use 4096 bits if your customer security policy allows it.

The following table describes the signing keys that SCM uses:

Key Type	Used For	Customer Input
CUSTOMER	Customer or customer automation commits.	GPG public key file required.
SCM	SCM-generated commits during install, migration, upgrade, or maintenance.	No input required. SCM manages this key.

Key Type	Used For	Customer Input
FLUX	Flux image automation commits, when image automation is enabled.	No input required. SCM manages this key.

To enable Git commit signing, add the `commit_signing` section under `infrastructure.git` in the environment or configuration request. For example:

```
"commit_signing": {
    "enabled": true,
    "customer_public_key_file": "/home/opc/customer-signing-public-key.asc"
}
```

By default, the Git commit signing is disabled. When signing is disabled, you don't need to provide a customer GPG public key file. For more information on the parameters, see *Payload Parameters for Siebel CRM Deployment*.

Signing Customer Git Commit

To sign customer Git commit, you must run the following commands in the Git repository where customer commits are made:

```
git config user.signingkey <CUSTOMER_GPG_KEY_ID>
git config commit.gpgsign true
```

After you run the above commands, normal Git commit commands are signed automatically.

To sign one commit manually, you run the following:

```
git commit -S -m "commit message"
```

To verify the signature for the last commit, you can run:

```
git log --show-signature -1
```

You must use the customer private key that matches the public key file that you provide to SCM.

Rotating Git Signing Keys

You can rotate Git signing keys when required by your organization's security policy or when a trusted signing key must be replaced. You can use the key rotation `git_sign_key_rotate` API to change a trusted Git signing key after deployment.

To rotate a Git signing key:

1. Create a request body as follows:
   a. For customer signing key:
   ```
   {
       "env_id": "<environment-id>",
       "type_of_key": "CUSTOMER",
       "customer_public_key_file": "/home/opc/new-customer-signing-public-key.asc"
   }
   ```
   b. For SCM and Flux signing keys:
   ```
   {
       "env_id": "<environment-id>",
       "type_of_key": "<SCM or Flux>"
   ```

```
}
```

The `type_of_key` value can be CUSTOMER, SCM, or FLUX. The `customer_public_key_file` value is required only for `CUSTOMER` rotation.

2. Submit request body using the POST method:
   `POST /scm/api/v1.0/git_sign_key_rotate`

You can also enable mandatory signed commits in your Git server or branch settings if your Git provider supports this option. You configure this setting outside SCM.

Using Vault for Managing Secrets

OCI Vault is a key management service that stores and manages master encryption keys and secrets for secure access to resources. When you provision a Siebel CRM environment using SCM, you must provide sensitive information in several places. Instead of entering sensitive values directly in the environment payload, you can store them as secrets in OCI Vault and provide their Oracle Cloud Identifier (OCID) values in the payload. SCM uses the OCIDs to fetch the actual secret values when required.

For more information about OCI Vault services, refer *Overview of Vault*.

For use with SCM, you can provision Vault in one of the following ways:

- Bring your own OCI Vault: In this case, you must provide your existing vault's OCID. Optionally, specify the OCID of a customer-managed KMS key during SCM stack creation. If you do not specify a key, SCM uses Oracle-managed encryption keys to encrypt SCM and Siebel file storage.

  SCM uses separate KMS key OCID inputs for SCM storage and Siebel environment file storage:

  - BYO KMS key OCID: Use this key to provide the KMS key OCID for SCM storage. SCM uses this key to encrypt the SCM boot volume and SCM file storage.
  - `siebel_storage_kms_key_ocid`: Use this parameter in the `infrastructure` section of the Siebel environment payload to encrypt Siebel environment file storage.

  To use customer-managed KMS keys, you must create the required dynamic groups and IAM policies for the applicable resources. For more information on configuring dynamic groups and policies, see *Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM*.

  > **Note:** You must use the correct OCID type when you configure Vault and storage encryption:
  > - A Vault OCID starts with `ocid1.vault` and identifies an OCI Vault.
  > - A Vault Secret OCID starts with `ocid1.vaultsecret` and identifies a stored secret, such as a password.
  > - A KMS key OCID starts with `ocid1.key` and identifies the customer-managed KMS key used by OCI storage resources.

- Have SCM provision a new Vault: You do not need to provide Vault information. SCM provisions a new Vault during stack creation. You can choose to create a default Vault or a virtual private Vault.

If you bring your own Vault, make sure SCM has the required access to fetch secrets from the Vault. For more information about the required policies, see *Common Policies*.

After SCM stack creation is complete, the Vault is available for use. You must complete the following steps before you provision a new Siebel CRM environment:

1. Create a Master Encryption Key (MEK) in the Vault.
2. Create secrets using the MEKs for the necessary fields in the payload section.
3. Copy the OCIDs of the secrets created in step 2 and provide them as input in the payload section.

Best Practices for Key Management

- **No "big secret"**: Ensure that secrets in your system are not long-term, have a limited blast radius, and are not of high value. Avoid shared secrets, such as using a single password for all administrative users.
- **As is / To be**: Maintain a clear overview of which users can view or modify the secrets. Often, maintainers of a project can access or extract its secrets. Reduce the number of individuals who can perform administrative tasks to limit exposure.
- **Log & Alert**: Collect all logs related to secrets and implement rules to detect secret extraction or misuse, whether accessed through a web interface or through methods like double base64 encoding or encryption with OpenSSL.
- **Rotation**: Regularly rotate secrets.
- **Forking should not leak**: Ensure that a repository fork or copy of job definitions does not inadvertently expose secrets.
- **Document**: Document the secrets you store and the reasons for their storage to facilitate easy migration when necessary.

Key Points for Managing Secrets Using Secret Management Products

- **Rotation/Temporality**: Ensure that the credentials used to authenticate with the secrets management system are rotated frequently and expire after their intended use.
- **Scope of Authorization**: Limit the scope of credentials to only those secrets and services necessary for their intended function.
- **Attribution of the Caller**: Maintain the ability to attribute actions to the individual or service that made requests to the secrets management solution. If this isn't supported by default, implement a correlation mechanism to track requests.
- **Compliance**: Adhere to the best practices listed in *Best Practices for Key Management*, including logging, alerting, and other essential measures.
- **Backup**: Store backups of critical secrets, such as encryption keys, in separate, secure storage solutions (e.g., cold storage).

Downloading and Installing Siebel Cloud Manager

Use this task to create and deploy the SCM stack (that is, to install the SCM instance in a virtual machine instance on OCI).

Before you perform this task, if you want SCM to create the Git repository, install GitLab CE (if it isn't already installed) into the same compartment where you install SCM.

During stack creation, review all default values displayed. Confirm each value or enter a new value as appropriate for your task. Steps for verifying SCM are also included.

To download and install SCM

1. Start the OCI console and log in.
2. Navigate to Marketplace, All applications.
3. Search for Siebel Cloud Manager.
4. Drill down on the Siebel Cloud Manager link.
5. Select the version and compartment (which you created in *Creating a Compartment*), check review terms and conditions.
6. Click **Launch Stack**.
7. Navigate to the Stack Variables page.
8. Under **General**, provide the following details:
    - Use existing resources: Specify whether you want to use existing resources (such as Compartment, VCN, mount target, database, and OKE) for the SCM instance. If you:
        - Select "Use existing resources", you can choose your existing resources (such as Compartment, VCN, mount target, database and OKE) for SCM configuration and Siebel environment provisioning.
        - Don't select "Use existing resources", SCM creates all the above-mentioned resources.
    - Root compartment OCID for your SCM instance (the compartment you created in Step 5)
    - SCM public ssh keys for accessing the SCM instance.
    - Resource prefix to name the OCI resources (all the resources created through this stack have this prefix added).

9. Under **Permissions**, specify one of the following permission types for the SCM instance:
   - Instance Principal to provide secure access and permissions to the SCM instance. When configuring Instance Principal, you have the option to use existing dynamic group and policy. By default, the "Use Existing Dynamic Group and Policy" checkbox is unchecked, that is, the system will automatically:
     - Create a new dynamic group.
     - Generate an OCI CLI policy.
     - Assign the created policy to the SCM instance.

     If you select the "Use Existing Dynamic Group and Policy" checkbox then, after the Apply stack job is completed, you must manually:

     - Add a new matching rule `"instance.id=<cm_instance_id>"` in an existing dynamic group - `<dynamic_group_name>`.
     - Add a new policy statement `"Allow dynamic-group <dynamic_group_name> to manage all-resources in compartment id <cm_compartment_ocid>"` in an existing policy. This policy allows you to access and perform various CRUD operations in SCM compartment from SCM instance.
   - User Principal for user-specific authentication. When you use User Principal, the SCM instance does not use dynamic groups or automatically generated policies. The OCI configuration is done manually.

     To set up User Principal, you will receive necessary details such as the user's private key, OCI fingerprint, and OCI passphrase. You can generate the private key and obtain the fingerprint from the OCI Console by navigating to **Users > Resources > API Keys**. All the permissions that apply for this user are available to the SCM instance.
10. Under **VCN**, specify whether you want to use existing VCN resource. This option enables you to use your existing network component resources and allows SCM to create and manage other resources such as mount target, file system, database, and OKE.
    - Network component for SCM Instance: Locate the compartment where the desired VCN is present for creating the SCM instance and in the following drop-down field select an existing VCN and a subnet.

      > **Note:**
      > - Allow TCP port 22 from your client network to establish SSH connection to the SCM instance.
      > - Allow TCP port 16690 from your client network to access the SCM application.
      > - Ensure appropriate egress rules are created for two-way traffic.

    - Network component for mount target: Locate the compartment where the desired VCN is present for creating the mount target and in the following drop-down field select an existing VCN and a subnet.

      > **Note:** Allow TCP ports 111, 2048, 2049, 2050 and UDP ports 111, 2048 from the SCM instance subnet.

    - "Use existing File system and Mount Target" option is provided to allow the user to bring existing resources instead of SCM to create the mount target and file system service. When this option is chosen user has to provide value for the IP address of the mount target.

      > **Note:** The existing file system export is to be provided in the subsequent section as below when "Use existing File system and Mount Target" is chosen.

11. When the "Use Existing Resources" or "Use existing File System and Mount Target" option is not selected earlier, then under **Storage**, select the availability domain for storage in which the shared mount target and file storage is created. The options are 1, 2, or 3.

    When the "Use Existing Resources" or "Use existing File system and Mount Target" is chosen, provide value of Export path for the desired file storage which will be used as persistence storage for SCM application.
12. Under **CloudManager Instance Configuration** section, configure the following:
    - **Select the Cloud Manager Instance Type**: Select the shape for the SCM instance. For example, select `VM.Standard.E5.Flex` or `VM.Standard.E6.Flex`.
    - **No of OCPU's**: Enter the number of OCPU cores. For example, enter `1`. This field is available only for SCM instance types that support configurable OCPU values.
    - **Amount of Memory(GB)**: Enter the memory in gigabytes. For example, enter `4 GB`. This field is available only for SCM instance types that support configurable memory values.
    - **Assign Public IP Address**: Select this option to assign a public IP address to SCM. If you do not select this option, SCM uses a private IP address.

      > **Note:** Assigning a public IP address configures SCM for public access. Not assigning a public IP address configures SCM for private access only. Switching between public and private access is not supported.

    - **Allowed source CIDR for Cloud Manager access**: Enter the IPv4 Classless Inter-Domain Routing (CIDR) block that can connect to SCM over SSH and the SCM API/UI port. This field is available and mandatory only when **Assign Public IP Address** is selected. You must provide the narrowest CIDR that still allows administrators to access SCM:
      - To allow one specific public IP address, enter the IP address with `/32`. For example, enter `203.0.113.10/32` when only one administrator machine, VPN egress IP, NAT gateway IP, or corporate proxy egress IP must be allowed.
      - To allow an entire network range, enter the subnet in CIDR notation. For example, enter `203.0.113.0/24` when administrators connect from a known corporate subnet or controlled network range.

      The value must be a valid IPv4 CIDR with prefix length `/1` through `/32`. SCM does not allow the following values:

Blocked Value or Range	Reason
Any `/0` CIDR, for example `0.0.0.0/0` or `1.1.1.1/0`	Allows the entire IPv4 internet.
`0.x.x.x`	Special-use address range.
`127.x.x.x`	Loopback address range.
`224.0.0.0/4` through `239.255.255.255`	Multicast range.

Blocked Value or Range	Reason
`240.0.0.0/4` through `255.255.255.255`	Reserved or broadcast range.

> **Note:** For private SCM deployments, this value is not required because SCM does not expose a public IP address. Private access must come through the customer's VCN, VPN, FastConnect, bastion, or other private access path.

- **HTTP Proxy**: Enter the HTTP proxy server URL for HTTP requests. For example, enter yourhttpproxyserver.com:80.
- **HTTPS Proxy**: Enter the HTTPS proxy server URL for HTTPS requests. For example, enter yourhttpsproxyserver.com:80.
- **URL(s) to bypass proxy**: Enter the URLs that must bypass the proxy server (no_proxy). For example, enter externalurl1.com,externalurl2.com.

  > **Note:** You must consider all URLs that might require proxy bypass during SCM and Siebel CRM environment provisioning. The `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` variables are applied only to the SCM container as environment variables. They are not applied to the container management configuration.

- **Select security protocol for Siebel Cloud Manager**: Optionally, select the security protocol to use: HTTP or HTTPS.
- **Enter port number where Siebel Cloud Manager API will be accessed**: Specify the port number to use when accessing SCM through its APIs. If you select HTTPS, choose whether to provide your own PEM-format SSL/TLS certificate, such as a CA-signed or self-signed certificate. If you do not provide a certificate, SCM provisions and uses a self-signed certificate. You can change the certificate later.

  If you do not select a security protocol, SCM uses HTTPS by default and automatically provisions a self-signed certificate.

> **Note:** The example values specified are indicative only and are not mandatory. Review the vendor documentation to determine the recommended compute shape, CPU, memory, storage, and operating system requirements for your deployment. Select a configuration that meets your workload, performance, and availability requirements.

13. Under **Network Configuration**:

    - If "Use Existing Resources" is selected, then you will be prompted to provide existing VCN details, such as the VCN Compartment OCID where VCN resides, and the VCN Name and Subnet where the SCM instance should be created.
    - If "Use Existing Resources" is not selected, then you need to specify whether you want to use Advanced Network Configuration to manage subnet IP address ranges for the SCM instance and Siebel CRM deployments.

      Use this option only to override the default subnet CIDR ranges: /16 for the VCN and /24 for each subnet. If you specify Advanced Network Configuration, then you can modify the default settings of 10.0.0.0/16 for the IP range for the VCN CIDR block, 10.0.0.0/24 for the IP range for the SCM subnet CIDR block, and 10.0.255.0/24 for the IP range for the SCM private subnet CIDR block.

      For details, see *Using Advanced Network Configuration*.

14. Under **Key Management**, choose how you want SCM to use OCI Vault for key management. The available options depend on whether you selected **Use existing resources**:
    - If "Use existing resources" is not selected, you can choose one of the following options:
        - Select **Create a new OCI Vault** to allow SCM to create a new Vault.
        - Select **Enter OCID of your existing OCI Vault** to use an existing OCI Vault.
        - Select **Do not use Vault** to skip Vault usage for key management.
    - If "Use existing resources" is selected, you can choose one of the following options:
        - Select **Enter OCID of your existing OCI Vault** and provide the OCID of the existing OCI Vault. You can also enter the OCID of the customer-managed KMS key. The OCID of the key must start with `ocid1.key`. If you omit this value, SCM uses Oracle-managed keys.

          For more information about the best practices for secrets management, see *Using Vault for Managing Secrets*.
        - Select **Do not use Vault** to skip Vault usage for key management.

> **Note:** Oracle recommends using OCI Vaults to conform to best practices regarding managing secrets. For more information about the best practices for secrets management, see *Using Vault for Managing Secrets*.

15. Under **Tags**, you can optionally add tags for SCM-created resources. You can use defined tags and free-form tags to organize the OCI resources created by the SCM stack.

    For defined tags, select the tag namespace and tag key, and enter the tag value. You must ensure the defined tag namespace and key already exist, and that the stack principal has permission to use the selected tag namespace.

    If SCM creates file systems and you enter a customer-managed key, you must select at least one defined tag that the dynamic group and policy use to grant OCI File Storage access to the key.

    You can use the following matrix to determine when the SCM storage KMS key and tags are required.

Use existing resources	Use BYO FSS	Customer-managed key provided	SCM creates FSS	Defined tag required	Encryption behavior
false	false	No	Yes	No	Oracle-managed encryption
false	false	Yes	Yes	Yes, at least one defined tag	SCM boot volume and SCM-created FSS
false	true	Yes	No	Yes, at least one defined tag	SCM boot volume only; BYO FSS is not changed
true	Hidden / N/A	Yes	No	Yes, at least one defined tag	SCM boot volume only; existing resources are not changed

16. Choose **Run Apply** and then click **Create** to create the stack. Terraform scripts run which define the configuration for the new stack.

17. Wait for the completion of the Apply job. If an error such as `authorization failed` or `requested resource not found` appears, then choose **Run Apply** again.
18. Make a note of the following URLs provided at the end of the run log:
    - **CloudManagerApplication.** The URL for running SCM, which uses the public or private IP address and port number of the newly created instance. You will use this URL to run SCM, as described in *Reducing the Ingress Range for Siebel Cloud Manager*. For example:
      `https://<CM_instance_IP>:<port_num>/`
      - **Siebel Lift utility**: The Siebel Lift utility is available on My Oracle Support (MOS) as the **Siebel CRM 26.8.0.0 Lift Utility (Patch)** media file. For more information, see *Downloading and Running the Siebel Lift Utility*.
19. To verify the running status of the application, run `ssh` in the SCM VM instance and check the `systemctl` status for siebel-cloud-manager, as follows:

    ```
    ssh opc@[CM_instance_IP]
    ```
20. To verify the SCM application is running, run the following commands:

    ```
    sudo podman ps
    sudo podman logs -t cloudmanager -f
    ```
21. To check the response, launch the following URL:

    ```
    https://<CM_instance_IP>:<port_num>/
    ```

    If you are performing a greenfield deployment, then you are now ready to create an environment using SCM. Otherwise, you must first download and run the Siebel Lift utility, as described in *Downloading and Running the Siebel Lift Utility*, before you can create an environment using the lifted artifacts in the OCI Object Store.

Upgrading Oracle Linux on the SCM VM

This topic describes the steps to upgrade the VM instance, on which SCM is deployed, from Oracle Linux 7 to Oracle Linux 8. This upgrade will also update the SCM container image to 25.4 or later.

Oracle Linux 7 has reached the end of its Premier Support; hence you should upgrade the VM on which SCM is deployed from Oracle Linux 7 to Oracle Linux 8. Upgrading the VM instance from Oracle Linux 7 to Oracle Linux 8 will install Podman, set up SCM as a Podman container, and make Podman the default tool to manage the SCM container registry. Additionally, the Podman SCM container is set up as a service using Quadlet that enables it to run declaratively under the `systemd` service.

Note: After the upgrade, you must execute all Docker commands with their Podman equivalents.

This topic includes the following information:

- *Prerequisite Tasks for Upgrading Oracle Linux on the VM*
- *Upgrading Oracle Linux on the VM*
- *Troubleshooting Oracle Linux Upgrade on VM*
- *Handling Additional FSS created during OL8 Upgrade*

Prerequisite Tasks for Upgrading Oracle Linux on the VM

You must verify the SCM version and health of the SCM application before you upgrade Oracle Linux on the VM:

1. Verify the version of SCM running on the VM.
   ```
   sudo docker ps
   ```
2. If you're running SCM 24.7 or lower, then you must migrate to SCM 24.8, as follows:
   a. Upgrade the SCM server to 24.8, as follows:
      ```
      cd /home/opc
      bash start_cmserver.sh CM_24.8.0
      ```
   b. Verify the version of the SCM application, as follows:
      ```
      sudo docker ps
      ```

      The version of the running SCM container must be 24.8.0.
   c. Migrate the new SCM features, as follows:
      ```
      docker exec -it cloudmanager bash
      cd /home/opc
      bash siebel-cloud-manager/scripts/cmapp/migration.sh
      ```

      Choose "Update CM instance script" when the options are presented by the `migration.sh` script.
   d. Exit from the SCM container and restart the container, as follows:
      ```
      cd /home/opc/cm_app/{CM_RESOURCE_PREFIX}/bash start_cmserver.sh CM_24.8.0
      ```
3. Ensure that the SCM application is functioning by executing the following API request:
   ```
   curl --location --request GET 'https://<CM_Instance_IP>:<Port>/scm/api/v1.0/environment/<env_id>' \
        --header 'Content-Type: application/json' \
        --header 'Authorization: Basic <Base 64 encoded user:api_key>'
   ```

   The SCM service is functioning correctly if the response includes valid environment details.
4. Create a backup of the SCM file system mounted at `/home/opc/cmapp` on the current OL7 instance to ensure data recovery in the event of any unexpected issue during the upgrade process.

   > **Note:** When taking a backup, it is important that you:
   > - Do not store the backup on the same SCM instance, as this instance will be destroyed and recreated during the upgrade process. Use an alternate and secure location to store the backup.
   > - Do not solely depend on FSS snapshots for backup purposes.

Upgrading Oracle Linux on the VM

To upgrade Oracle Linux from Oracle Linux 7 to Oracle Linux 8 on the VM:

1. Create a dummy Oracle Resource Manager stack as follows:
   a. Sign in to OCI Console.
   b. Open the navigation menu and click **Marketplace**.
   c. Under Marketplace, click **All Applications**.
   d. Enter "Siebel" in the search bar.
   e. Click **Siebel Cloud Manager (SCM)**.

f. Select SCM version 25.4 or later.
g. Select the compartment to create the stack in.
h. Select the **Oracle Standard Terms and Restrictions** checkbox.
i. Click **Launch Stack**.
j. Enter the stack name, description, and tags (you can retain the default values) and click **Next**.
k. Configure the variables for the infrastructure resources. Use the same values as the existing stack and click **Next**.
l. Deselect the **Run apply** checkbox.
m. Click **Create**.

2. Create the execution plan for the stack resources and download the Terraform configuration as follows:
   a. Go to the stack created in step 1.
   b. On the Stack Details page, click **Plan**.
   c. Optionally, update the plan job name.
   d. Click **Plan**. The Plan job takes a few minutes to complete.
   e. Once it succeeds, click the plan job name under the Jobs table. The plan job page appears.
   f. Click **Download Terraform Configuration** to download the configuration ZIP file.

3. Edit the existing stack either through OCI Console or through OCI CLI:
   - Through OCI Console as follows:
     i. On the **Stacks** list page, click the existing stack that was used to deploy SCM. The stack page appears.
     ii. Scroll down to the bottom left of the stack page, click the **Variables** tab under the **Resources** section, and make a note of all the variables and their values.
     iii. Click **Edit**.
     iv. Click **Edit Stack**. The Edit Stack page appears.
     v. Click **Browse** under the **Terraform configuration source** section.
     vi. Browse and select the configuration ZIP file downloaded in Step 2.
     vii. Click **Open**.
     viii. Click **Next**.
     ix. Click **Browse** under the **OCI User private key** section.
     x. Browse and upload the OCI private key file.
     xi. Review all other variables for the infrastructure resources and ensure that their values match the values noted earlier in step 3.b of OCI console.
     xii. Click **Next**.
     xiii. Do not select **Run Apply**.
     xiv. Click **Save Changes**.
     xv. On the Stack Details page, click **Plan**. Wait for the Plan job to complete successfully.
     xvi. Verify the Plan job status in OCI Console. After the Plan job succeeds, review the Plan job logs to confirm that only 1 resource, the OCI instance, is marked for destruction. Ensure that no other resources are marked for deletion.
     xvii. On the Stack Details page, click **Apply**. Wait for the Apply job to complete successfully.
   - Through OCI CLI:
     i. Update the stack with the new terraform configuration ZIP file as follows:

     ```
     oci resource-manager stack update --stack-id <Stack OCID> --config-source <zipFilePath> --force
     ```

     The variables in the example have the following values:

     - `<Stack OCID>` is the OCID of the existing stack to update.
     - `<zipFilePath>` is the path of the Terraform configuration zip file downloaded in the previous step.

     ii. Create the Plan job as follows:

     ```
     oci resource-manager job create-plan-job --stack-id <Stack OCID>
     ```

     The variables `<Stack OCID>` is the OCID of the existing stack to update.

     Wait for Plan job to complete successfully.

     iii. Verify the Plan job status in OCI Console.
     iv. Review the logs to confirm that only 1 resource, OL7 instance, is getting destroyed.
     v. Create Apply job:

     ```
     oci resource-manager job create-apply-job --execution-plan-strategy FROM_PLAN_JOB_ID --stack-id <Stack OCID> --execution-plan-job-id <Plan Job OCID>e
     ```

     vi. Wait for few minutes for Apply job to complete.
     vii. Verify status of Apply job in OCI Console.

> **Note:** The Oracle Linux 7 VM instance is destroyed and a new Oracle Linux 8 VM instance is created. SCM will be installed as a Pod man container service on the new Oracle Linux 8 VM instance.

4. Verify the deployment as follows:
   a. SSH into the SCM instance:
      ```
      ssh -i <ssh Private Key> opc@<SCM Host IP>
      ```
   b. Verify the status of the SCM container:
      ```
      sudo podman ps
      ```
   c. Verify the status of `cloudmanager` container service:
      ```
      sudo systemctl status cloudmanager
      ```
      Ensure that the `cloudmanager` service is Active: active (running).
   d. Retrieve the provisioned environment details:
      ```
      curl --location --request GET 'https://<CM_Instance_IP>:<Port>/scm/api/v1.0/environment/<env_id>'
       \
          --header 'Content-Type: application/json' \
          --header 'Authorization: Basic <Base 64 encoded user name and api_key>'
      ```
      The SCM service is functioning correctly if the response includes the valid environment details.

      > **Note:** If you are unable to retrieve the environment details, and the Apply job logs from the ORM stack indicate that a new File Storage System (FSS) is being created, see the troubleshooting section *Handling Additional FSS created during OL8 Upgrade*.

5. Migrate the new SCM features as follows:
   ```
   sudo podman exec -it cloudmanager bash
   cd /home/opc
   bash siebel-cloud-manager/scripts/cmapp/migration.sh
   ```
   Choose one of the options presented by the `migration.sh` script. Run the script multiple times for the required options.
6. Restart the SCM container as follows:
   ```
   cd /home/opc/cm_app/{CM_RESOURCE_PREFIX}/bash start_cmserver.sh <SCM VERSION>
   ```

Troubleshooting Oracle Linux Upgrade on VM

After upgrading, if SCM is inaccessible and you receive the following response:

```
ERR_CONNECTION_REFUSED - This site can't be reached
```

Then wait for a few minutes after the "Run apply" job completes, as SCM is deployed through the cloud-init script it takes time to complete after the "Run apply" job completes. If the error persists even after this period, follow these steps to diagnose and resolve the issue:

1. SSH in to the SCM instance.
2. Check the status of the SCM container:
   ```
   sudo podman ps
   ```

Troubleshoot based on the result of the command:

- If this command throws the following error:

```
bash: podman: command not found
```

It implies that the cloud-init script is still running and the Podman installation hasn't started. Check the `cloud-init-output.log` for progress details, as follows:

```
vi /var/log/cloud-init-output.log
```

If the log indicates that the cloud-init script was skipped, that is, the Podman installation did not occur yet the script was marked as completed, then run the cloud-init script manually as follows:

```
ssh -i <ssh Private Key> opc@<SCM Host IP>
curl --fail -H "Authorization: Bearer Oracle" -L0 http://169.254.169.254/opc/v2/instance/metadata/
user_data | base64 --decode > /tmp/user-init.sh
sudo bash -x /tmp/user-init.sh
```

Enter 'n' when prompted for overwriting the ssh key.

> **Note:** Running this script multiple times creates duplicate entries for the "DEFAULT" profile in the `~/.oci/config` file, resulting in SCM container startup failure. To resolve this issue:
>
> a. Edit the `~/.oci/config` file manually to delete the duplicate "DEFAULT" profile entries.
> b. Reload `systemd`:
>
> ```
> sudo systemctl daemon-reload
> ```
>
> c. Restart the SCM container:
>
> ```
> sudo systemctl restart cloudmanager
> ```

- If the status of the SCM container indicates that the SCM container is stopped or restarting, check the logs:

```
sudo podman logs -f cloudmanager
```

If the container logs shows the `NotAuthenticated 401` error, it implies that the OCI config or fingerprint passed during SCM stack deployment is incorrect. To resolve the issue:

a. Update the OCI config in the `/home/opc/.oci/config` file with the correct parameter values.
b. Restart the SCM container:

```
sudo systemctl restart cloudmanager
```

- If it does not list the SCM container, check the status of the container service:

```
sudo systemctl status cloudmanager
```

If the command returns the following message:

```
cloudmanager.service: Start request repeated too quickly.
cloudmanager.service: Failed with result 'exit-code'.
Failed to start Podman cloudmanager.service.
```

The error indicates that the SCM container is repeatedly restarting. To diagnose and resolve the issue, check the container logs and analyze the issue. If the error is `NotAuthenticated 401`, follow the above steps to resolve.

For more information on any other issues, see *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

Handling Additional FSS created during OL8 Upgrade

If you are unable to retrieve the provisioned environment details after upgrade, check the Terraform Apply job log. If the log indicates that the `oci_file_storage_file_system`, `oci_file_storage_export`, `oci_file_storage_export_set` resources were created, it indicates that a new file system has been provisioned. To restore the environment data from the old SCM file system, execute the following steps on the newly created OL8 instance:

1. Verify that the mount is empty:
   ```
   ls -lrt /home/opc/cmapp
   ```
   If the directory has no files or directories, continue with the subsequent steps.
2. Create a temporary directory to mount the old file system:
   ```
   mkdir /home/opc/tmp_fs
   ```
3. Mount the old file system:
   ```
   sudo mount -t nfs <Mount_Target_IP>:/<Old_Export_Path> /home/opc/tmp_fs -o nolock
   ```
4. Copy data from the old file system to the new file system:
   ```
   sudo cp -r /home/opc/tmp_fs/<SCM_RESOURCE_PREFIX>/ /home/opc/cmapp/
   ```
5. Retrieve the provisioned environment details:
   ```
   curl --location --request GET 'https://<CM_Instance_IP>:<port_num>/scm/api/v1.0/environment/<env_id>' \
    --header 'Content-Type: application/json' \
    --header 'Authorization: Basic <Base 64 encoded user name and api_key>'
   ```
   If the command output displays valid environment details, it indicates that the SCM service is operating as expected.
6. Unmount and delete the old file system that is mounted on `tmp_fs` if the data has been retrieved.

About URLs for Siebel CRM Deployments on OCI

All Siebel CRM deployments on OCI use the following base URL for their resource endpoints, where the application deployments and configurations are located:

```
https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/
```

In the example above:

- `<SCM_instance_IP>` is the hostname IP address for the SCM instance.
- `<port_num>` is the port number on the hostname.

Note: Where security has been configured, `https` is used instead of `http`.

The following are two main uses of the base URL:

- It serves as the location for Siebel CRM deployments:

  ```
  https://<CM_instance_IP>:<port_num>/scm/api/v1.0/environment
  ```

  You can access each Siebel CRM deployment by appending the environment ID to the end of the URL as follows:

  ```
  https://<CM_instance_IP>:<port_num>/scm/api/v1.0/environment/4QVRX5
  ```

- It holds the Siebel CRM configurations that you can create for one or more greenfield deployments:

  ```
  https://<CM_instance_IP>:<port_num>/scm/api/v1.0/configuration
  ```

  You can access each Siebel CRM configuration by appending the configuration ID at the end of the URL as follows:

  ```
  https://<CM_instance_IP>:<port_num>/scm/api/v1.0/configuration/MZM3RJ
  ```

Note: For Siebel CRM applications deployed in some earlier releases, the base URLs have this form, and are still valid: `https://<CM_instance_IP>:<port_num>/api/v1/environments/`. For more information, see earlier versions of this document.

Uploading Files to the SCM Container Using File Sync Utility

You can upload files, related to Siebel CRM deployment, to the SCM container using File Sync Utility (FSU). File Sync Utility is a collection of REST APIs that makes it convenient to upload files to the SCM container. When you run File Sync Utility, it creates a folder, referred to as sync folder, and uploads files to it. You can create multiple sync folders using File Sync Utility.

The following constraints apply when uploading files through File Sync Utility:

- You can only upload files with the following extensions:
  `.jks, .crt, .txt, .ini, .key, .sso, .p12, .properties, .pem, .ora.`
- You can only upload files of size up to 1 MB.

This topic covers how you can use the File Sync Utility for:

- *Creating a Sync Folder and Uploading Files*
- *Uploading Files to an Existing Sync Folder*
- *Retrieving Files and Directories in a Sync Folder*
- *Retrieving the List of Sync Folders*
- *Deleting a Sync Folder*

Creating a Sync Folder and Uploading Files

You can create a sync folder and upload files to it using the `syncutilities/upload` API.

To create a sync folder and upload files to it using curl, call the `syncutilities/upload` API as follows:

```
curl -X POST \
 --user "<username>:<password>" \
 --header 'Content-Type: multipart/form-data' \
 --form "file[]=@/path/to/xxxxxxxxxx.jpg" \
 --form "file[]=@/path/to/xxxxxxxxx.crt" \
 --form "file[]=@/path/to/xxxxxxx.ini" \
 --url https:// <CM_instance_IP>:<port_num>/scm/api/v1.0/syncutilities/upload
```

In the example:

- `user` is the credentials (user name and password) for basic authentication.
- `Content-Type` is the type of request content. The value of `Content-Type` must be `multipart/form-data`.
- `form` is the file to upload to the sync folder through the "`file[]`" key.
- `url` is the POST endpoint to create a sync folder and upload files to it.

This command generates a new sync ID and creates the sync folder with the same name as the sync ID. It then uploads the files in the command to this folder. It also creates a log file in the sync folder that records the activities (files added and files overridden) performed on the sync folder.

Sample response:

```
{
     "data": {
          rejected_files": [],
          "sync_id": "SCM_FileSync_2024_07_04_07_18_20_2ZH7QY",
          "synced_files": [
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.jpg",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.crt",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.ini",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/
SCM_FileSync_2024_07_04_07_18_20_2ZH7QY_synclogs.txt"
          ]
     },
     "message": "All files successfully synced.",
     "status": "success",
}
```

Uploading Files to an Existing Sync Folder

You can upload new files to an existing sync folder or override existing files in the sync folder using the `syncutilities/upload` API.

To upload new files or override existing files in an existing sync folder using curl, call the `syncutilities/upload` API as follows:

```
curl -X PUT \
 --user "<username>:<password>" \
 --header 'Content-Type: multipart/form-data' \
 --form "file[]=@/path/to/xxxxxxxxx.crt" \
 --form "file[]=@/path/to/xxxxxxx.ini" \
 --url https:// <CM_instance_IP>:<port_num>/scm/api/v1.0/syncutilities/upload/<sync_id>
```

In the example:

- `user` is the credentials (user name and password) for basic authentication.
- `Content-Type` is the type of request content. The value of `Content-Type` must be `multipart/form-data`.
- `form` is the file to add or override in an existing sync folder through the "`file[]`" key.
- `url` is the PUT endpoint to upload new files or override existing files in an existing sync folder. Here, `<sync_id>` is the name of the sync folder that you want to update.

Sample response:

```
{
     "data": {
          "sync_id": "SCM_FileSync_2024_07_04_07_18_20_2ZH7QY",
          "synced_files": [
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.crt",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.ini",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/
SCM_FileSync_2024_07_04_07_18_20_2ZH7QY_synclogs.txt"
          ]
     },
     "message": "Sync folder updated successfully",
     "status": "success",
}
```

Retrieving Files and Directories in a Sync Folder

You can retrieve the list of files and directories in a sync folder using the `syncutilities` API.

To retrieve the list of files and directories in a sync folder using curl, call the `syncutilities` API as follows:

```
curl -X GET \
--user "<username>:<password>" \
--url https://<CM_instance_IP>:<port_num>/scm/api/v1.0/syncutilities/<sync_id>
```

In the example:

- `user` is the credentials (user name and password) for basic authentication.
- `url` is the GET endpoint to retrieve the list of file and directories in a sync folder. Here, `<sync_id>` is the name of the sync folder of which you want to retrieve the list of files and directories.

Sample response:

```
{
     "data": {
          "rejected_files": [],
          "sync_id": "SCM_FileSync_2024_07_04_07_18_20_2ZH7QY",
          "synced_files": [
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/
SCM_FileSync_2024_07_04_07_18_20_2ZH7QY_synclogs.txt",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.jpg",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.crt",
               "/home/opc/syncUtility/SCM_FileSync_2024_07_04_07_18_20_2ZH7QY/xxxxxxx.ini"
          ]
     },
     "message": "Synced Files successfully fetched",
     "status": "success"
}
```

Retrieving the List of Sync Folders

You can retrieve the list of all the sync folders using the `syncutilities` API.

To retrieve the list of all sync folders using curl, call the `syncutilities` API as follows:

```
curl -X GET \
--user "<username>:<password>" \
--url https://<CM_instance_IP>:<port_num>/scm/api/v1.0/syncutilities
```

Note: Don't add a forward slash (/) at the end, it will make the endpoint invalid.

In the example:

- `user` is the credentials (user name and password) for basic authentication.
- `url` is the GET endpoint to retrieve the list all sync folders.

Sample response:

```
{
     "data": {
          "sync_ids": [
               "SCM_FileSync_2024_07_04_07_18_20_2ZH7QY",
               "SCM_FileSync_2024_07_05_04_17_07_4CXU2Q",
               "SCM_FileSync_2024_07_08_11_39_57_2ZQE3A"
          ]
     },
     "message": "Sync Folders successfully fetched.",
     "status": "success"
}
```

Deleting a Sync Folder

You can delete a sync folder using the `syncutilities` API. Deleting a sync folder permanently deletes the sync folder and all the files and directories in it.

To delete a sync folder with the files and directories in it, call the `syncutilities` API as follows:

```
curl -X DELETE \
--user "<username>:<password>" \
 --url https://<CM_instance_IP>:<port_num>/scm/api/v1.0/syncutilities/<sync_id>
```

In the example:

- `user` is the credentials (user name and password) for basic authentication.
- `url` is the DELETE endpoint to delete a sync folder. Here, `<sync_id>` is the name of the sync folder that you want to delete.

Sample response:

```
{
     "data": {},
     "message": "Sync folder deleted.",
     "status": "success"
}
```

Mirroring Siebel Base Container Images

This topic describes how to mirror Siebel CRM base container images using REST APIs. Base images are the default Siebel CRM container images provided by Oracle and are also referred to as vanilla images.

The APIs allow you to mirror images and update the user container registry credentials.

This topic contains the following sections:

- *Mirroring Siebel CRM Base Images*
- *Signing Mirrored Images with a Customer Key*
- *Managing User Container Registry Credentials*

Mirroring Siebel CRM Base Images

You can mirror the Siebel CRM base images to a user defined destination registry through the `base/images/mirror` API. The API call pulls the container images and pushes them to the user defined destination registry.

To mirror the base images, call the `base/images/mirror` API as follows:

POST endpoint:

```
https://<CM_Instance_IP>:<port_num>/scm/api/v1.0/base/images/mirror
```

Sample payload:

```
{
     "destination_registry": {
          "registry_url": "iad.ocir.io",
          "registry_user": "deploygroup/user.name@example.com",
          "registry_password": "aDgFFg123",
          "registry_prefix": "deploygroup"
     },
     "update_global_config": "true"
}
```

Note: Specify the value of payload parameters suitable for your circumstances by referring to the payload parameters in the *Payload Parameters for Siebel CRM Deployment* table.

For every image mirroring process job triggered via the POST API, a unique 6 character identifier "RUN_ID" is generated through which you can check status of the job. To get the details of the current job, execute a GET method API call as follows:

GET endpoint:

```
https://<CM_Instance_IP>:<port_num>/scm/api/v1.0/base/images/mirror/<RUN_ID>
```

Sample response:

```
{
     "data": {
          "image_mirror_details": {
               "end_time": null,
               "images": {
                    "iad.ocir.io/deploygroup/base-builder:latest": {
```

```
                        "end_time": "Fri, 13 Sep 2024 09:57:49 +0000",
                        "message": "Successfully uploaded to destination registry.",
                        "start_time": "Fri, 13 Sep 2024 09:57:45 +0000",
                        "status": "completed"
                    },
                    "iad.ocir.io/deploygroup/curlimages/curl:latest": {
                        "end_time": "Fri, 13 Sep 2024 09:57:54 +0000",
                        "message": "Successfully uploaded to destination registry.",
                        "start_time": "Fri, 13 Sep 2024 09:57:49 +0000",
                        "status": "completed"
                    },
                    "iad.ocir.io/deploygroup/cm/dbutils:23.1": {
                        "end_time": null,
                        "message": "Currently being uploaded.",
                        "start_time": "Fri, 13 Sep 2024 09:57:54 +0000",
                        "status": "in-progress"
                    }
                },
                "images_processed": 2,
                "run_id": "XXXXX",
                "start_time": "Fri, 13 Sep 2024 09:57:45 +0000",
                "status": "in-progress",
                "total_images": 31,
                "update_global_config": true
            }
        },
        "message": "Current status of the image mirroring process",
        "status": "success"
    }
```


Parameters in Response Definition

The following table describes the parameters in the response:

Payload Parameter	Section	Definition
image_mirror_details	Top level	Provides the details of the image mirroring process.
start_time	image_mirror_details	The time when the image mirroring process started.
end_time	image_mirror_details	The time when the image mirroring process finished.
images	image_mirror_details	A dictionary containing individual images and their respective upload statuses.
start_time	images	The time when this specific image started uploading.
end_time	images	The time when this specific image finished uploading.
message	images	Status message for the specific image upload process.
images_processed	image_mirror_details	The number of images processed so far during the mirroring process.
run_id	image_mirror_details	Unique identifier for tracking the current image mirroring process.
status	image_mirror_details	The overall status of the image mirroring process (in-progress, completed, failed).
total_images	image_mirror_details	The total number of images scheduled for mirroring in the process.
update_global_config	image_mirror_details	Indicates whether the destination registry details were saved to the global configuration for future use.
message	Top level	A general message describing the current status of the image mirroring operation.

Payload Parameter	Section	Definition
status	Top level	Indicates the success or failure of the API request (success, failed).

Note: If you're using Oracle Cloud Infrastructure Registry (OCIR) as the destination registry, ensure that `registry_user` has permission to create the repository name (for example, `project01/acme-web-app/component1`) in the root compartment, or you should create the repositories before pushing images. The `/home/opc/siebel-cloud-manager/scripts/cmapp/yaml/siebel_images.yaml` file in the SCM container will contain a list of images, and the corresponding repository paths need to be created in OCIR. For more information, see the *Creating a Repository* topic in the OCI documentation.

Signing Mirrored Images with a Customer Key

You can sign mirrored images with a customer key. SCM uses Cosign to sign and verify mirrored image digests in the container registry. SCM supports Cosign-native keys and unencrypted RSA or ECDSA PEM private keys with matching public keys. SCM imports RSA or ECDSA PEM keys into Cosign format.

By default, customer image signing is optional.

If you omit `customer_signing` and no customer signing profile exists, SCM:

- Verifies Oracle-signed source images before mirroring.
- Mirrors the images to the destination registry.
- Does not sign the mirrored images with a customer key.
- Does not enforce customer-signature verification during later environment creation.

If you omit `customer_signing` and a customer signing profile already exists, SCM reuses the existing profile. If the profile has customer signing enabled, SCM continues to use customer signing.

Example: Mirror Images Without Customer Signing

You can use the following payload when you want SCM to mirror images without signing the mirrored images with a customer key:

```
{
    "destination_registry": {
        "registry_url": "{customer-registry}",
        "registry_user": "{registry-user}",
        "registry_password": "{registry-password}",
        "registry_prefix": "{registry-prefix}"
    },
    "update_global_config": "true",
    "customer_signing": {
        "enabled": false
    }
}
```

When customer signing is disabled, SCM verifies Oracle source images before mirroring but does not sign the mirrored registry images with a customer key.

Example: Mirror Images with Customer Signing

To sign mirrored images with a customer key:

1. Copy the customer signing files to the SCM container:
   ```
   /home/opc/config/image_trust/customer-cosign.key
   /home/opc/config/image_trust/customer-cosign.pub
   ```
2. Call the mirror API with customer signing enabled:
   ```
   {
       "destination_registry": {
           "registry_url": "<customer registry URL>",
           "registry_user": "<registry user>",
           "registry_password": "<registry password>",
           "registry_prefix": "<registry prefix>"
       },
       "update_global_config": "true",
       "customer_signing": {
           "enabled": true,
           "private_key_path": "/home/opc/config/image_trust/customer-cosign.key",
           "public_key_path": "/home/opc/config/image_trust/customer-cosign.pub",
           "private_key_password": "<signing key password>"
       }
   }
   ```

SCM signs the mirrored destination images and creates the following manifest:

```
/home/opc/config/image_trust/mirror-image-manifest.yaml
```

SCM uses this manifest and the customer public key during Siebel CRM environment creation.

Managing User Container Registry Credentials

You can update and retrieve the user container registry credentials through the `registry` API. This ensures that the Siebel CRM base image can be pulled from the user's container registry for Siebel CRM environment creations in the future.

This topic contains the following sections:

- *Updating User Container Registry Credentials*
- *Retrieving User Container Registry Credentials*

Updating User Container Registry Credentials

To update the user container registry credentials, call the `registry` API as follows:

PUT endpoint:

```
https://<CM_Instance_IP>:<port_num>/scm/api/v1.0/registry
```

Sample payload:

```
{
     "registry_url": "iad.ocir.io",
     "registry_user": "deploygroup/user.name@example.com",
     "registry_password": "aDgFFg123",
     "registry_prefix": "deploygroup"
}
```

Specify payload parameters suitable for your circumstances by referring to the following payload parameters:

Payload Parameter	Section	Definition
registry_user	Top Level	(Required) Specifies the user ID to connect to the container registry. This user must have access to push and pull images from container registry.
registry_url	Top Level	(Required) Specifies the URL of the Open Container Initiative compliant container registry. For example, for the OCI container registry in the Ashburn region, you might use **`iad.ocir.io`**. For more information, see the *Preparing for Container Registry* topic in the Oracle Cloud Infrastructure documentation.
registry_password	Top Level	(Required) Specifies the password or authentication token for **`registry_user`**.
registry_prefix	Top Level	(Optional) Specifies a prefix that's appended after registry_url. For OCI container registry, this should be the tenancy namespace, if needed, you can add a suffix to it. As it's an optional field, it can be left blank.

Retrieving User Container Registry Credentials

To retrieve the user container registry credentials, call the **`registry`** API as follows:

GET endpoint:

```
https://<CM_Instance_IP>:<port_num>/scm/api/v1.0/registry
```

Sample response:

```
{
     "data": {
          "registry_password": "********",
          "registry_prefix": "deploygroup",
          "registry_url": "iad.ocir.io",
          "registry_user": "deploygroup/user.name@example.com"
     },
     "message": "Registry details have been fetched successfully.",
     "status": "success"
}
```

Note: To ensure security, sensitive data such as the registry password is masked in the response.

Reducing the Ingress Range for Siebel Cloud Manager

Before you run SCM to create an environment, you can optionally reduce the ingress range to tighten network security for SCM. The procedure below uses **`siebel-cm`** as the example compartment name.

Related Topics

Downloading and Installing Siebel Cloud Manager

To reduce the ingress range for SCM

1. Navigate to Instances.
2. Select the *prefix* `-siebel-cm` compartment in the left side of screen to list the newly created instances.
3. Drill down to the *prefix* `-siebel-cm` instance with public IP specified.
4. Drill down to the link *prefix* `-siebel-cm-subnet` given as subnet under Primary VNIC header.
5. Drill down on the link `security-list-for-cm`.
6. Click **Edit Ingress Rules**.
7. (Optional) Change the 0.0.0.0/0 for Source CIDR for Destination Port 16690 to the desired IP Addresses/Range.

> **Note:** Ingress port 16690 is by default added to the security list with 0.0.0.0/0 to open for internet. It is recommended to change this CIDR block to the required limited IP addresses or CIDR blocks.

Stack apply job logs have the URL endpoint for the REST application, or you can get the IP address info from the Compute instance of SCM (*prefix* `-siebel-cm`) inside *prefix* `-siebel-cm` compartment, where *prefix* is the input given during stack creation.

Using Advanced Network Configuration

SCM and the Siebel CRM environments you deploy on OCI use several subnets. Each subnet uses a range of IP host addresses. When you install SCM, for complete network IP range customization, you can specify to use Advanced Network Configuration.

If you choose this option, then you can specify the IP host address range for the SCM subnet and private subnet. Later, when you deploy Siebel CRM, you specify the minimum IP range for each subnet in this deployment's environment. The IP ranges are specified using CIDR notation.

- You configure the SCM subnet and private subnet one time only, during SCM stack creation. This subnet applies in the compartment in which you installed SCM. See also *Downloading and Installing Siebel Cloud Manager*.
- All other subnets apply per Siebel CRM environment, in the compartment for that environment. You specify these additional subnet CIDR ranges as payload parameters for the Siebel CRM deployment, as described in *Payload Parameters for Siebel CRM Deployment*.

  The default CIDR range values for the payload parameters below are those for VCN 10.0.0.0/16, SCM subnet 10.0.0.0/24, and private subnet 10.0.255.0/24 ranges. When using Advanced Network Configuration, specify different values as appropriate.

```
"infrastructure": {
  "siebel_lb_subnet_cidr": "10.0.1.0/24",
  "siebel_private_subnet_cidr": "10.0.2.0/24",
  "siebel_db_subnet_cidr": "10.0.3.0/24",
  "siebel_cluster_subnet_cidr": "10.0.4.0/24"
}
```

When giving Siebel CRM subnet IP ranges using Advanced Network Configuration, review the following information:

- The minimum IP range required for one Siebel CRM environment is /26 for VCN, providing 64 IP host addresses. The minimum IP range required for each subnet is either /29 or /30, as shown in the following table. /29 provides eight host addresses and /30 provides four host addresses.

  > **Note:** In each subnet's CIDR range, the OCI Networking service reserves the first two addresses and the last address for use by Oracle. So, if a subnet requires one host address, then the minimum IP range for that subnet is /30. If a subnet requires two or more host addresses, then the minimum IP subnet range is /29. You must use a valid CIDR for each subnet's IP range.

- Each subnet range must be inside the parent VCN IP range given.
- In case of multiple Siebel CRM environments in the same SCM instance, you must use nonoverlapping IP ranges for each environment's subnets.

Subnets for SCM and Siebel CRM on OCI

Subnet	Usage	Minimum Subnet Range / Description
Cloud Manager subnet	SCM public endpoint	/30 SCM uses one host address in this subnet for its public end point.
Cloud Manager private subnet	SCM mount target	/29 SCM uses one host address in this subnet for a shared mount target resource.
siebel_lb_subnet_cidr	Load Balancer subnet	/29 The OCI Load Balancers use two host addresses in the subnet, one each for the primary and secondary load balancers.
siebel_private_subnet_cidr	Kubernetes worker nodes private subnet	/29 By default, SCM configures three Kubernetes nodes. Each Siebel CRM environment requires three host addresses in this private subnet for Kubernetes nodes.
siebel_db_subnet_cidr	Database private subnet	/30 A minimum of one host address is needed for the database private subnet for Autonomous Database (ATP) or Oracle Database Cloud Service (DBCS).
siebel_cluster_subnet_cidr	OKE cluster subnet (Kubernetes API server)	/30 A minimum of one host address is needed for the Kubernetes API Server.

Related Topics

Downloading and Installing Siebel Cloud Manager

Reducing the Ingress Range for Siebel Cloud Manager

Payload Parameters for Siebel CRM Deployment

https://docs.oracle.com/en-us/iaas/Content/Network/Concepts/overview.htm

Customizing Configurations Prior to Greenfield Deployment

In a greenfield deployment scenario, when you deploy a Siebel CRM environment as described in *Deploying Siebel CRM on OCI*, by default the environment is automatically configured and provisioned as a single step. This topic describes how you can optionally decouple the configuration and provisioning stages, for the purpose of customizing the configuration before you provision the environment.

Configurations are maintained in Git repositories for your subsequent customization and use. The configurations present in Git repositories are the source for environment provisioning in this use case.

Once you've created a configuration, you can customize it according to your requirements. Configuration customizations can include any of the types of changes described for deployed environments in *Making Incremental Changes to Your Siebel CRM Deployment on OCI*. Examples include adding or deleting components on a server, adding new profiles, or adding or deleting parameters for an enterprise, server, or component.

You can use your customized configuration as a base for provisioning multiple environments, such as for test or production purposes. To do this, you must pass the configuration ID during the provisioning step.

Configuration options for a Siebel CRM greenfield deployment on OCI have the following use cases:

- **Greenfield deployment use case 1 (default configuration).** In this use case, you use SCM to deploy a new Siebel CRM environment on OCI with a default configuration.
- **Greenfield deployment use case 2 (customized configuration).** In this use case, you use SCM to deploy a new Siebel CRM environment on OCI with a customized configuration. The configuration is created first (for which a configuration ID is generated), then you customize the configuration, and then you deploy it for one or more environments. This is the use case described in this topic.

For an example payload and usage guidelines, see *Example Payload to Deploy Siebel CRM*.

> **Note:** After deployment, any configuration changes must be made in the deployed environment, as described in *Making Incremental Changes to Your Siebel CRM Deployment on OCI*. If you require the same changes in the original customized configuration that you created using greenfield configuration use case 2, then you must make the same changes in both locations.

This topic contains the following information:

- *Creating the Configuration and Obtaining the Configuration ID*
- *Customizing the Configuration*

Related Topics

Deploying Siebel CRM on OCI

Checking the Status of a Requested Configuration

Making Incremental Changes to Your Siebel CRM Deployment on OCI

Creating the Configuration and Obtaining the Configuration ID

This topic is part of *Customizing Configurations Prior to Greenfield Deployment*.

If you plan to customize the configuration prior to provisioning the environment for greenfield deployment use case 2, then the first step is to create the configuration and to obtain the six-character ID for this configuration.

To create the configuration and obtain the configuration ID

- Do a `POST` API like the following:

  ```
  POST https://<CM_Instance_IP>:16690/scm/api/v1.0/configuration
  ```

  > **Note:** Specify a payload appropriate for greenfield deployment use case 2. For an example payload and for usage guidelines, see *Example Payload to Deploy Siebel CRM*.

  The configuration is created and its configuration ID is returned, such as MZM3RJ.

  As a result of the above `POST` API, if the `git_type` is set to `gitlab`, the following two Git repositories are created:

  - `config_<namespace>_<config_id>-helmcharts`

    (for example: config_stage_MZM3RJ-helmcharts)
  - `config_<namespace>_<config_id>-cloud-manager`

    (for example: config_stage_MZM3RJ-cloud-manager)

  You can access this configuration in the SCM container as follows:

  a. SSH into the SCM instance.
  b. Access the configuration in the `configuration/<config_id>` directory. For example:
     ```
     sudo podman exec -it cloudmanager bash
     /home/opc/siebel/configuration/MZM3RJ
     ```

  You can use a selfLink for monitoring purposes. For example:
  `https://<CM_Instance_IP>:16690/scm/api/v1.0/configuration/MZM3RJ`

Customizing the Configuration

This topic is part of *Customizing Configurations Prior to Greenfield Deployment*.

After creating a configuration and obtaining its configuration ID, you can customize this configuration prior to provisioning the environment (greenfield deployment use case 2).

To customize Siebel CRM configuration that require changes in Helm charts repository

For example, for adding a custom component, changing parameter values in a component, enabling/disabling components, named subsystem changes, component definition changes and so on.

1. SSH into the SCM instance.
2. Enter the SCM container:
   ```
   sudo podman exec -it cloudmanager bash
   ```
3. Go to the `paramconfig` directory:
   ```
   cd /home/opc/siebel/configuration/<config_id>/config_<namespace>_<config_id>-helmcharts/siebel-config/paramconfig
   ```
   The `paramconfig` folder has files supporting this Siebel CRM configuration. For more information about customization use cases and the YAML or other configuration files that you can modify, see *Making Incremental Changes to Your Siebel CRM Deployment on OCI*. (For existing deployments, the `paramconfig` folder is in a different location.)
4. Make all changes necessary to customize the configuration files.
5. Commit your customization in the Helm charts Git repository. Make sure to add all modified files:
   ```
   cd /home/opc/siebel/configuration/<config_id>/config_<namespace>_<config_id>-helmcharts/siebel-config
   git add .
   git commit -m "<customization note>"
   git push
   ```
   The above changes will be included in the initial environment provisioning, where you specify the configuration ID.
6. Check the status of a requested configuration, as described in *Checking the Status of a Requested Configuration*.
7. Deploy the environment with the customized configuration, as described in *Deploying Siebel CRM on OCI*. In this step (for greenfield deployment use case 2), you specify only the configuration ID and the deployment name.

To customize Siebel CRM Kubernetes deployment parameters that require changes in the SCM repository

For example, for changing the number of replicas for SES or SAI, adding a new SiebServer Profile, setting resources like CPU and memory specific to individual Siebel Server and so on.

1. SSH into the SCM instance.
2. Enter the SCM container:
   ```
   sudo podman exec -it cloudmanager bash
   ```
3. Edit the `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel/siebel.yaml` file to add "sesResources" for each Siebel server as:
   ```
   siebelServer:
    - profile: siebel
    replicas: 1
    sesResources:
   ```

```
limits:
cpu: 4
memory: 24Gi
requests:
cpu: 1
memory: 8Gi
siebsrvr_prefix: edge
- profile: siebel
replicas: 1
sesResources:
limits:
cpu: 4
memory: 24Gi
requests:
cpu: 1
memory: 8Gi
siebsrvr_prefix: tibus
```

> **Note:** sesResources defined at the profile level for individual Siebel server takes higher precedence over the generic sesResources overridden in payload.

4. Make all changes necessary to customize the configuration files.
5. Commit your customization in the SCM Git repository. Make sure to add all modified files. The above changes will be included in the initial environment provisioning, where you specify the configuration ID.
6. Check the status of a requested configuration. For more information, see *Checking the Status of a Requested Configuration*.
7. Deploy the environment with the customized configuration, as described in *About this Chapter*. In this step (for greenfield deployment use case 2), you specify only the configuration ID and the deployment name.

Deploying Siebel CRM on OCI

After you have performed all the prerequisite tasks, you can use SCM to deploy Siebel CRM on OCI. To do this, you prepare a suitable payload and then you execute this payload on SCM.

This topic contains the following information:

- *Overview of Siebel CRM Deployment Steps using Siebel Cloud Manager*
- *Authorization Information for Siebel CRM Deployment*
- *Using a BYO VCN*
- *Using BYO File Storage*
- *Using BYO Kubernetes*
- *Configuring Fixed NodePorts for Traefik*
- *Using a BYO Namespace*
- *Using a BYO Database*
- *Using Resource Tags*
- *Using Network Policies*
- *Checklist for Creating a BYOR Deployment*
- *Using Security Adapters for Siebel CRM*
- *Using Custom Keystore*
- *Terminating SSL/TLS at the Load Balancer (FrontEnd SSL) using SCM*

- *Auto-enablement of Siebel Migration Application*
- *Preparing Siebel File System and Migration Directories*
- *Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM*
- *Executing the Payload to Deploy Siebel CRM*
- *Example Payload to Deploy Siebel CRM*
- *Using a Customer-Managed KMS Key for an SCM-Created Autonomous Database*

Related Topics

Customizing Configurations Prior to Greenfield Deployment

Making Incremental Changes to Your Siebel CRM Deployment on OCI

Overview of Siebel CRM Deployment Steps using Siebel Cloud Manager

You can deploy Siebel CRM using SCM by creating a deployment payload and submitting it using the `environment` API. Payload execution results in the execution of various stages necessary for Siebel CRM deployment by SCM. The execution flow depends on whether the "Use existing resources" option was selected or not during SCM stack creation.

The term "BYO" stands for "Bring Your Own" and is indicative of existing resources at the disposal of the user. For example, BYOD stands for "Bring Your Own Database".

Siebel CRM applications can be deployed using SCM in different ways based on the type of infrastructure information provided in the payload:

1. You bring all resources (Fully BYOR): If you select "Use existing resource" during SCM provisioning, you must provide the details of all the resources such as existing mount target, file system, OKE, database in the payload that the SCM instance will use to create Siebel CRM deployment(s).
2. SCM creates resources:
    a. All infra resources: If you do not select "Use existing resource" during SCM provisioning, SCM creates and configures all the required infrastructure resources for a Siebel CRM deployment, such as database, OKE, mount target, file system, and so on.
    b. All infra resources except database: If you do not select "Use existing resource" during SCM provisioning, user can still provide information of an existing database in the payload. SCM creates all other infra resources (mount target, file system, OKE etc.) for Siebel CRM deployment. You must make sure that the database can be connected from SCM and OKE.

After the SCM installation is completed, you can invoke the payload with the necessary information to start the deployment process of Siebel CRM and monitor the deployment stages using the REST API calls. For more details, see *Checking the Status of a Requested Environment*.

> **Note:** When deploying Siebel CRM using SCM, ensure that the Siebel CRM version matches the SCM version. Deploying an earlier version of Siebel CRM with a later version of SCM is not supported. For example, you cannot deploy Siebel CRM 25.7 using SCM 25.10.

Authorization Information for Siebel CRM Deployment

The `auth_info` section in the payload is:

- Mandatory when you select "Use existing resources" during SCM stack creation, as SCM doesn't modify anything in the database when you choose the BYOD option.
- Optional for an SCM provisioned environment, which means you didn't select the "Use existing resources" option during SCM stack creation. If values are not provided, default values are assigned.

The required details in `auth_info` are:

- `admin_user_name` and `admin_user_password`:

  The Siebel CRM administrator username and password is required for configuring Siebel CRM topology.

- `default_user_password`:

  The default user password which is used for logging in the rest of the users, when user info is exported, and made available in the lifted artifacts.

- `table_owner_user` and `table_owner_password`:

  The schema name and password which is the owner for all Siebel CRM tables, the password is required to execute `postinstalldb` process during update deployments.

- `anonymous_user_password`:

  The password used for connecting the anonymous user to the database and provided in Siebel CRM configuration.

Using a BYO VCN

The BYO VCN feature allows you to use your own VCN in OCI. This provides significant flexibility in setting up networking components like VCN, subnet, Internet Gateway, Service Gateway, NAT gateway and so on to launch the SCM instance and subsequently the Siebel CRM environment. This helps you ensure compliance with your specific network topology and security requirements.

Existing VCN can be used for SCM provisioning and/or during Siebel CRM environment provisioning.

During SCM provisioning, if the "Use existing VCN" option is:

- Selected, SCM will not create a VCN. This option allows:
  - Selection of subnets for the SCM instance, mount target, resources (OKE, file system, database) for Siebel CRM environment provisioning.
  - Optionally, using an existing database (see Notes on BYOD section) which can be in the same or different user specified VCN where other resources like OKE, file system are created.
- Not selected, then:
  - SCM will create a VCN. SCM instance, mount target, other resources (OKE, file system, database) for Siebel CRM deployment will be created in that VCN (that is, in this case, SCM stack and Siebel CRM will be in the same VCN).

- Optionally, an existing VCN can still be used only for Siebel CRM environment provisioning (which means, the SCM instance and Mount Target can be in different VCNs where Siebel CRM environment will be created in).
- You can still use an existing database (see Notes on BYOD section) which can be in the same or different user specified VCNs where other resources like OKE, file system for Siebel CRM deployment are created in.

Effectively, regardless of "Use existing VCN" option chosen, there is flexibility regarding using existing VCNs for Siebel CRM deployment resources (OKE, file system) and database.

You must ensure that the following permissions/access OCI policy requirements are met:

- ATP Database: Allow dynamic group {namespace}-instance-principal-group to manage autonomous-database in compartment id {siebel-compartment-id}.

  For more information, refer *Policy Details for Autonomous Database on Serverless*.
- DBCS Database: Allow dynamic group {namespace}-instance-principal-group to manage database-family in compartment id {siebel-compartment-id}.

  For more information, refer *Policy Details for Base Database Service*.
- OKE: Allow dynamic group {namespace}-instance-principal-group to manage cluster-family in compartment id {siebel-compartment-id}.

  For more information, refer *Policy Configuration for Cluster Creation and Deployment*.
- File system: Allow dynamic group {namespace}-instance-principal-group to manage file-family in compartment id {siebel-compartment-id}.

  For more information, refer *Create, manage, and delete file systems*.

Using BYO File Storage

The BYO file system option lets you use an existing file system and mount target (exports for the file system) during Siebel CRM environment provisioning. This will enable you to use your existing Siebel CRM file systems on NFS shares with Siebel CRM deployed in Kubernetes using SCM, without shifting the file system content. Alternatively, if you are not relying on OCI file system services, you can use an existing NFS share to migrate the lifted Siebel CRM file system.

Scenarios of shifting the file system:

1. When you set the `shift_siebel_fs` key to false, make sure the NFS share specified in the payload contains a valid Siebel CRM file system. The system checks that the required Siebel CRM directories exist, but it does not perform any other validation. You must ensure that you use the correct NFS shares.
   - att
   - atttmp
   - cms
   - eim
   - Marketing
   - red
   - ssp

2. When you do not set the `shift_siebel_fs` key, then the value defaults to `true` and shifts the file systems. For lift-and-shift use case, make sure that the lift bucket includes all required file system artifacts.

   The provided file system must have a directory structure as follows:

   ```
   MOUNT_TARGET_IP:/EXPORT_PATH e.g. 10.0.0.1:/siebfs0
   ```

   In the `siebfs0` path it is expected to contain a valid Siebel CRM file system as per the directory structure given above.

   You can even provide multiple mount targets for different file systems. The parameters involved are: `mounttarget_exports`, `siebfs_mt_export_paths`, and `zookeeper_mt_export_path`. For more information, see *Payload Parameters for Siebel CRM Deployment*.

Note: SCM does not re-encrypt BYO existing file systems. BYO existing file systems do not use `siebel_storage_kms_key_ocid`.

Using BYO Kubernetes

The concept of BYO refers to using your own Kubernetes clusters for Siebel CRM provisioning rather than SCM creating managed OKE cluster.

Choosing your own Kubernetes can provide significant benefits in terms of customization, cost management, security, performance, avoiding vendor lock-in, innovation and skill development. However, it also requires higher levels of expertise and operational overhead compared to utilizing SCM creating managed OKE cluster. Organizations should weigh these factors carefully based on specific needs and capabilities before deciding to go with bring your own Kubernetes.

Note: Currently, SCM supports only managed nodes for OKE. You are responsible for upgrading Kubernetes on managed nodes and for managing cluster capacity.

- **Customization and Control**: With Bring your own Kubernetes, you have full control over your Kubernetes cluster, including control planes and worker nodes. This allows for more granular customization as well as optimization tailored to specific requirements.
- **Cost Management**: BYO Kubernetes can be more cost-effective in certain scenarios, especially if you have an existing infrastructure setup with all network configurations or need to run a large number of clusters. Managed OKE often come up with additional costs for convenience and support they provide.

  You can optimize resource allocation and scaling policies to match workload needs, potentially reducing unnecessary expenses.
- **Security and Compliance**: For organizations with strict data sovereignty requirements, managing own Kubernetes clusters ensures that data remains within your control and complies with company's regulations.

  You can implement custom security measures to your cluster, such as network policies, access controls, encryption standards that meet organization's specific compliance and security needs.
- **Performance and Reliability**: You can design and implement your own high availability and disaster recovery strategies tailored to your infrastructure and business requirements.

- **Avoiding vendor lock-in**: BYO Kubernetes allows you to avoid vendor lock-in by maintaining the flexibility to move your workloads across different cloud providers or on-premises environments without being tied to specific vendor's managed Kubernetes service.

When you select the "Use existing resource" option during SCM provisioning, SCM uses the cluster you provide instead of creating a new one. Therefore, if you want SCM to use your Kubernetes cluster during the Siebel CRM environment setup through REST API POST, use one the following cluster options:

- BYO OKE - Bring your own Oracle Kubernetes Engine (OKE) option allows you to use an existing OKE cluster for Siebel CRM deployment.
- BYO OCNE - Bring your own Oracle Cloud Native Environment (OCNE) option lets you to leverage your own OCNE cluster for Siebel CRM deployment.
- BYO Other - Bring your own Other option enables you to use any other CNCF compliant Kubernetes cluster for Siebel CRM deployment.

You must ensure that your cluster adheres to the following rules else the execution workflow will fail during resource state validation stage:

- The Kubernetes cluster should not contain namespaces such as `<env_name>` before deployment, as these namespaces will be used during Siebel CRM environment provisioning.
- At least one node should be in Active state.
- The Kubernetes cluster should be accessible from the SCM instance with required polices and VCN peering, if necessary, should be configured before deployment.

Notes on OKE (Oracle Container Engine for Kubernetes)

To use your own OKE cluster during Siebel CRM environment provisioning through REST API POST invocation, you must:

- Set the payload parameter `kubernetes_type` to `BYO_OKE`
- Configure `oke_cluster_id` and `oke_endpoint` together or `oke_kubeconfig_path` alone under the `kubernetes > byo_oke` section.

For more information, see *Payload Parameters for Siebel CRM Deployment*.

You can provision multiple Siebel CRM environments in the same OKE cluster when the "Use Existing Resource" option is selected.

> **Note:** From CM_23.1.0, users can deploy multiple Siebel CRM environments using the same OKE cluster by provisioning each environment in their own namespace. When the "Use Existing Resource" option is selected, we recommend you to use OKE without any existing flux setup. If there is any existing flux setup, you can:
>
> - Either uninstall using the command:
>
>   ```
>   flux uninstall all --namespace=<namespace>
>   ```
>
> - Or upgrade existing flux setup with `flag --watch-all-namespaces=false` to restrict the scope to watch the namespace where the toolkit is installed.

Notes on OCNE (Oracle Cloud Native Environment)

OCNE is an integrated suite of open-source software tools and platform designed to facilitate the development, deployment, and management of cloud-native applications.

OCNE is build around Kubernetes, the leading orchestration platform and includes additional tools and components to enhance Kubernetes capabilities making it easier for organizations to adopt cloud-native technologies in a secure, scalable, and reliable manner.

To use your own OCNE cluster during Siebel CRM environment provisioning through REST API POST invocation, you must:

- Set the payload parameter `kubernetes_type` to `BYO_OCNE`
- Configure `kubeconfig_path` under the `kubernetes > byo_ocne` section.

For more information, see *Payload Parameters for Siebel CRM Deployment*.

Notes on Other Kubernetes Cluster

To use any other Kubernetes cluster of your choice during Siebel CRM environment provisioning through REST API POST invocation, you must:

- Set the payload parameter `kubernetes_type` to `BYO_OTHER`
- Configure `kubeconfig_path` under the `kubernetes > byo_other` section.

For more information, see *Payload Parameters for Siebel CRM Deployment*.

> **Note:** We support deployment of Siebel CRM environment on Kubernetes clusters that adhere to Cloud Native Computing Foundation (CNCF) standards.

Configuring Fixed NodePorts for Traefik

You can assign fixed Kubernetes NodePorts to expose the Traefik ingress service through specific ports. Fixed ports let you limit external firewall, network security group, or security-list rules to the required ports.

To use fixed NodePorts, in the Siebel CRM deployment payload:

- Set `infrastructure.ingress_controller.ingress_service_type` to `NodePort`.
- Configure the optional `infrastructure.ingress_controller.node_ports` section.

For example, the following configuration assigns fixed NodePorts to the Traefik HTTP and HTTPS entry points:

```
"ingress_controller": {
    "ingress_service_type": "NodePort",
    "node_ports": {
        "web": <valid-unused-port>,
        "websecure": <valid-unused-port>
    }
}
```

For more information about the deployment payload parameters, see *Payload Parameters for Siebel CRM Deployment*.

> **Note:** Your cluster administrator can configure the NodePort range at the cluster level. Kubernetes uses the default range of `30000-32767`. Self-managed clusters can use a different range with the `kube-apiserver --service-node-port-range` option. For more information:
>
> - See the *service-node-port-range* description in the Kubernetes *kube-apiserver* documentation
> - See the *Configuring the node port service range* chapter in the RedHat documentation, for OpenShift.

If your cluster uses a custom NodePort range, specify unused values within that range. You must use different values for the `web` and `websecure` services. Kubernetes rejects a NodePort value that is already allocated or outside the configured range.

Using a BYO Namespace

Bring your own (BYO) namespace refers to the configuration approach of using your own namespace in Kubernetes for Siebel CRM deployment. In Kubernetes, a namespace provides a mechanism for isolating and organizing cluster resources. Here, namespace is the logical division within the Kubernetes cluster in which you want to install Siebel CRM.

You can use BYO namespace in the following scenarios:

- You have an existing namespace that is managed externally. In this case, ensure that the name of the existing namespace matches the `name` field in the Siebel CRM deployment payload.
- You don't want the namespace to be cleaned up during reruns.

BYO namespace provides flexibility in managing namespaces while leveraging the power of Kubernetes for Siebel CRM provisioning.

To use an existing namespace, you must set the `byo_ns` parameter to `true` under the `infrastructure > Kubernetes > byo_oke` or `byo_ocne` or `byo_others` section. When you set the `byo_ns` parameter to `true`:

- SCM will not create the namespace during Siebel CRM deployment.
- SCM rerun will perform resource cleanup within the namespace. However, it will not delete the namespace itself.
- SCM will not delete the namespace when the DELETE operation is triggered through the DELETE API.

For more information, see *Payload Parameters for Siebel CRM Deployment*.

Using a BYO Database

SCM can deploy Siebel CRM with Oracle Database only.

When you select the "Use existing resources" option during SCM stack creation, you can use an existing Oracle database, along with other existing resources such as VCN, OKE, mount target, and so on, for your Siebel CRM deployment on OCI. In this case, you must provide the details of the resources as SCM will not create the resources.

However, if you provision an SCM instance without selecting the "Use existing resources" option, BYOD is still supported. In this scenario, you can use your existing database and SCM will provision the rest of the resources, such as OKE, mount target and so on.

Before deploying Siebel CRM using an existing database, you must ensure that you establish connectivity between the existing Siebel CRM database and the:

- SCM instance and
- Pods in the Kubernetes cluster

Connecting to an empty (without any data) existing database is not supported.

For more information on the parameters required for using an existing database, see *Payload Parameters for Siebel CRM Deployment*.

For a sample `database` section that shows how to use an existing database, see *Example for BYO Database*.

Connectivity Information

The database connection details are used by Siebel CRM applications to establish a connection with the database. You must provide connectivity information, such as wallet credentials and the connection identifier, to ensure secure and successful database access.

The required details in the "BYOD" are:

- `wallet_path`: The absolute path of the Oracle net services configuration files or Oracle client credentials (wallet) is required for connecting to the database. The wallet files have to be copied inside the SCM container. The wallet should contain at least the `tnsnames.ora` for a valid folder. During environment provisioning the wallet will be validated if it contains the `tnsnames.ora`. TLS enabled wallets are also supported. The provided wallet path will be copied inside the environment directory for usage.
- `tns_connection_name`: The connection identifier provided in the field will be validated whether it is present in the tnsnames.ora file or not. If it is not available, a client side validation (400) will be raised.

The provided connection string will be used by the Siebel applications to connect with the database.

If your existing database is on OCI, whether in the same region as your SCM VCN or a different one, you can use private routing to avoid connections through the internet. You must establish a connection from your existing SCM VCN to the VCN where the database resides. The different scenarios where the database can reside and how to establish connection are:

- Present in the same region:

  If the database is present in the same VCN as the SCM VCN, you must establish local peering between the two VCNs.

  For more information, refer to *Local VCN Peering using Local Peering Gateways*.
- Present in different region:

  If your database is present in different region than your SCM VCN, then you must establish Remote peering connection to establish connectivity between both the VCN.

  For more information, refer to *Remote VCN Peering using a Legacy DRG*.

In the above cases, you will be required to add a route in the route table to allow traffic to the database or vice versa.

Every Siebel CRM deployment requires connectivity from the following subnets:

- SCM subnet:

  This will be required to run administrative tasks such as verifying if the database is in right shape, are the provided credentials valid etc. This is done prior to creating a Siebel CRM deployment.
- OKE Nodes subnet:

  This will be required by all Siebel CRM applications to establish connection to the database starting from user authentication to querying tables. For SCM subnet mentioned above, it can be done prior to the deployment, but in case of OKE nodes subnet, they are not yet created at the stage. So, users can provide the OCID of the Dynamic Routing Gateway (DRG) (using the field drg_ocid) which needs to be attached to the OKE Nodes subnet and the destination CIDR block of the DB's subnet or VCN (using the field `destination_db_cidr_block`) where the traffic has routed through the DRG.

Provided the above are done, the traffic which is controlled by security list also should allow traffic through these ranges. The traffic going outside of SCM instance subnet and OKE nodes subnet are already taken care by the deployment. It will allow traffic to go out. From the database subnet's security rules, similar rules have to be written to allow traffic to come in. In case, the traffic is only controlled through security List, ATP will still require a Network Security Group (NSG) to allow the traffic through it.

For more information, refer to *Private Endpoints Configuration Examples on Autonomous Database*.

Connectivity Tests

Before the provisioning of the environment, you must ensure that the database is accessible from the:

- SCM instance:
  - Admin User/Password based access
  - Table Owner User/Password based access
  - Guest User access
- Kubernetes nodes in which the Siebel CRM application lives.

Issues with theses connectivity requirements will be reported in stage "validate-connectivity" and the provisioning activities in OCI for Siebel CRM deployment will be stopped here. The deployment can be re-run after fixing issues related to connectivity.

Workflow Continuation

In the BYOD flow, database import is skipped, and the "import-db-stage" is marked as "Passed."

Debugging Methods

The individual stage logs will record all connection test information and provide detailed results. Logs for connectivity-related tests can be found in the "validate-connectivity" stage. When the tests are passing they leave trail of the events, such as:

- """
- admin user validation in progress
- admin user validation completed
- tblo user validation in progress
- tblo user validation completed
- """

You can perform validations manually using SQL Plus to check. After the issue is fixed, you can rerun the workflow by submitting the payload as before. Common reasons for which the connections might fail are:

- Host provided in the `tnsnames.ora` is not reachable.

  Proper connection has to be established to validate this. Incase of OCI, the VCN in which the database resides should have proper security rules to the SCM instance.

  In case of any other externally hosted Oracle database, the guidelines for those providers needs to be followed and whitelisted to provide access to the SCM instance.
- Invalid info in wallet.

The data provided in the wallet has to be valid to establish the connection.

- Invalid authorization information.

  The data provided in the `auth_info` section has to be valid in order to establish the connection.

Other scenarios which cause failure of connectivity are caught and the details are provided in the stage logs.

Using Resource Tags

OCI tagging enables you to add metadata to resources by using key-value pairs. Resource tags help you organize, identify, and manage resources according to your business requirements, such as cost tracking, ownership, and environment classification. You can use the optional `resource_tags` section in the SCM provisioning payload to apply OCI tags to supported resources that SCM creates. For more information, refer *Overview of Tagging* in Oracle online documentation.

You can configure the following tag types:

- `defined_tags`
- `freeform_tags`

For more information, see *Payload Parameters for Siebel CRM Deployment*.

You can define resource tags for the following resources:

- Siebel-related OCI resources under the `siebel` section. SCM applies `siebel.resource_tags` to the following OCI resources:
  - Object Storage buckets created during greenfield bucket preparation.
  - Oracle Cloud Infrastructure Registry (OCIR) repositories created during artifact migration or repository setup.
- Observability-related OCI resources under the `observability` section. SCM applies `observability.resource_tags` to the following OCI resources:
  - OCI Monitoring and Application Performance Monitoring (APM) dashboard imports.
  - Saved searches created during dashboard imports. For dashboard imports, `observability.resource_tags` acts as an override:
    - If non-empty tag values are provided, SCM applies those values to the imported dashboard resources.
    - If tags are omitted or empty, SCM preserves any tags already defined in the dashboard template.
  - OCI Log Analytics log groups.
  - Log Analytics dashboards.
  - Prometheus OCI Block Volumes.

Using Network Policies

You can use network policies to control which network traffic is allowed to reach SCM-managed CRM workloads. You can enable network policies through the `network_policy.enabled` parameter. For example:

```
{
```

```
    "network_policy": {
       "enabled": true
    }
}
```

You can use this parameter to enable or disable SCM-generated `NetworkPolicy` resources.

For new CRM deployments, `NetworkPolicy` generation is enabled by default. You do not need to add the `network_policy` section unless you want to explicitly document the default.

You can set `network_policy.enabled` to `false` to temporarily disable the generated `NetworkPolicy` resources for compatibility or troubleshooting. This is not recommended.

Existing environments that omit `network_policy` continue to load successfully. When SCM regenerates CRM deployment content for those environments, SCM treats the missing value as enabled by default.

For more information on the `network_policy.enabled` parameter, see *Payload Parameters for Siebel CRM Deployment*.

Checklist for Creating a BYOR Deployment

Before deploying a BYOR environment, you need to go through a checklist consisting of various steps to ensure that you have a smooth deployment. If the steps or resources are used directly by your environment, without any validation, the application may fail. Here is a resource-wise list of the different steps which need to be performed/validated.

- *OKE*
- *Mount Target*
- *Database*
- *Git Repository*
- *Debugging Common URL Connectivity Issues*

OKE

When using an existing OKE, verify the following:

- Check if the API server URL is accessible from the SCM instance. If the `kubeconfig` path is provided, then make sure that the API server is accessible. It can be validated by running basic `kubectl` commands. If the URL is not accessible, see *Debugging Common URL Connectivity Issues* to debug.
- If the Cluster ID is provided while creating a deployment, ensure if the SCM instance's principal (User or Instance principal) has the required access to download `kube-config`. This can be validated by running the following OCI command:

  ```
  oci ce cluster create-kubeconfig --cluster-id <SOME_CLUSTER_ID> --file $HOME/.kube/config --region us-phoenix-1 --token-version 2.0.0 --kube-endpoint PRIVATE_ENDPOINT
  ```

- If the SCM instance uses Instance Principal, verify if the following policy exists:

  ```
  'Allow <subject> to manage cluster-family in compartment id <oke_compartment_ocid>'.
  ```

  Here `<subject>` can be group/dynamic_group etc.

  For more information, refer to *Policy Syntax* documentation.

  Once you have saved the configuration, you need to set the variables for KUBECONFIG and OCI_CLI_AUTH.

  ```
  # set the required env variables
  ```

```
# Possible values are - api_key or instance_principal based on your OCI principal configuration.
export OCI_CLI_AUTH=api_key
# Path to your OKE kube config
export KUBECONFIG=/path/to/your/oke/config
# fetch the cluster info
kubectl cluster-info --request-timeout 5s
# Get the nodes info
kubectl get nodes
```

- If it is not accessible, then the instance might not have access to read/use the resource. Either contact your tenancy administrator for proper permissions or check for your policies.
- If you are behind a proxy, make sure that either the API server is accessible through the Proxy server or if it can be bypassed. Provide it in `no_proxy` (Contact your administrator for appropriate choice).
- Login to any one of the nodes and validate if the Git server is accessible by either making a request or cloning a repository etc.

Mount Target

Mount targets' access endpoints are always private endpoints. So inter-network/VCN validations must be done.

- These are accessed internally. If you are running behind a proxy, chances are your proxy settings might route it to the proxy server. So if it requires to be bypassed, pass it in the `no_proxy` settings.
- NFS uses port 2049 by default. One can configure a different port as well. Ensure that your Security List/NSG's have rules to allow the traffic. If the URL is not accessible, see *Debugging Common URL Connectivity Issues* to debug.
- NFS client exports are also controlled in mount target and are configured to read or read/write. Ensure that you have read/write permissions on it.
- Use NFS mount commands to mount a local directory to verify if the SCM instance can communicate. You can unmount the directory after verifying. If you are unable to mount, it is likely that the mount target URL is not reachable. In this case, see *Debugging Common URL Connectivity Issues* to debug.

  ```
  nfs mount <args>
  nfs umount <args>
  ```

- Ensure that your mount targets' endpoints are accessible from your OKE nodes. You can verify it by logging in to the OKE's node and checking if the endpoint is accessible. This is a mandatory requirement as the applications need to access the file systems.

Database

You need to validate database endpoints, SID, Listener, and Credentials before creating an environment.

- Ensure that the endpoints are accessible from both SCM container and the OKE node.
- To check if the database endpoints are accessible from SCM, connect to the database endpoint using tools such as `telnet` from the SCM container. You also need to verify if the listener is valid.
- To check if the database endpoints are accessible from OKE, connect to any one of the OKE nodes and use commands such as `telnet` to check if they can be reached from there.
- If the database endpoints are private endpoints (DB endpoints), then there is a chance that your OKE node might not be able to resolve the Host name URL. In such case, verify it with the IP address of the host. If nodes can be setup with an option to resolve the DB hostnames, then IP address will not be required.
- Verify all the credentials such as: Siebel Admin, Table Owner, anonymous user credentials etc. You can use `sqlplus` available in the SCM instance to login and validate the credentials.

- To connect to a database using sqlplus, set the following variables.

```
export ORACLE_HOME=/usr/lib/oracle/19.29/client64
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=/home/opc/siebel/IIG8L6/wallet
```

- After setting the variables, connect to sqlplus CLI.

```
sqlplus username/password@connection_identifier
# username - db user whom you would like to authenticate
# password - password of that db user.
# connection string which you would like to establish connection and verify with. Can be found in
 tnsnames.ora
```

For more information, see the *Using a BYO Database* topic.

Git Repository

The Git instance, where the Helm charts and SCM must be created, should be accessible from both the SCM instance and the OKE nodes. SCM access is required to ensure any changes in release yamls and terraform configuration is tracked.

To verify Git access from the SCM container, log in to the SCM container and run the following commands. If required, to verify from OKE, list down all the OKE Nodes and `ssh` into any one of the node and run the following steps to verify it:

```
# Check if you are able to ping to the gitlab IP/URL and access it.
ping gitlaburl.com

OR
# Hit any of the existing gitlab API with the token to verify
curl https://gitlaburl.com/api/v4/users --header 'Authorization: Basic token'
# even a 40x response also makes it clear that the URL is accessible

# Try to clone an existing repo to verify if SSH access is available
git clone git@gitlaburl.com/repo-name.git # using SSH
```

Note: The Git instance or repositories should be accessible from both the SCM instance and the OKE nodes.

Debugging Common URL Connectivity Issues

If any URL is not reachable or not able to communicate, you can debug the issue using the following steps.

You can use utilities such as telnet, traceroute, ping, curl etc. Install these utilities using `yum/dnf`. If you are behind a proxy server and not able to reach the repo, then you need to configure proxy details in `/etc/yum.conf`

```
sudo yum install ping curl traceroute telnet
```

1. To verify if an URL is accessible or not, verify the security rules/NSG rules of the corresponding resource and the host from which you are connecting.

   For more information, refer to the *Network Security Groups* documentation.
2. There is a possibility that there is a secondary barrier also added from resource side, that is ACL for DB's, NFS client export options for mount targets etc. Check if they are whitelisted.

   For more information, refer to the *Configure Access Control Lists for an Existing Autonomous Database Instance* documentation.
3. If you are connecting from your on-prem through a Fast connect coupled with a DRG, make sure you have matching rules for your DRG. This is applicable if two or more VCN's are also connected (even with Local Peering Gateway (LPG)).

For more information, refer to the *FastConnect with Multiple DRGs and VCNs* documentation.

4. Check if you are behind a proxy server and your proxy server allows connection to the URL. You can verify this by disabling proxy or adding in no_proxy to test it.

```
# verify the list of values set currently
printenv | grep 'PROXY\|proxy'
# update the required var - HTTP_PROXY, HTTPS_PROXY, NO_PROXY
export HTTP_PROXY=myproxyserver.com
```

5. Use `telnet` to see if you are able to reach a URL on a particular URL. Some tools such as sqlplus get hung when a connection does not happen.

```
telnet someurl.com 22
Connected to someurl.com
# Port - 22 on someurl.com is reachable from the current host.

telnet notreachableurl.com 1522
...
# 1522 on notreachableurl.com is not reachable
```

6. Use `traceroute` to see where exactly the hopping stopped. that is it might go out of an instance but may not go out to the public internet because an IG/NAT was not connected. In this case, the last hop would have been only the VCN.

```
traceroute someurl.com
1 hop1.com (10.0.35.153) 292.885 ms 289.622 ms 376.783 ms
2 hop2.com (10.0.32.130) 250.955 ms 250.505 ms 289.326 ms
3 hop3.com (10.0.29.42) 250.155 ms 250.227 ms 290.869 ms
4 hop4.com (10.76.13.210) 271.508 ms 268.169 ms 309.374 ms
5 hop5.com (10.76.13.209) 276.570 ms 273.716 ms 277.106 ms
6 hop6.com (10.76.27.10) 272.482 ms 272.206 ms 269.685 ms
7 hop7.com (10.76.27.68) 269.659 ms 269.013 ms 268.582 ms
8 hop8.com (10.196.6.42) 272.557 ms 273.320 ms 279.004 ms
9 * * *
10 final.destination.com (100.10.14.9) 272.173 ms !Z 271.058 ms !Z 318.078 ms !Z

# if it gets hung at ***, then possibly the packet is not able to proceed further from there to the next
 hop/router.
```

7. Ping the URL to verify if the server is up or not. Internet Control Message Protocol (ICMP) has to be enabled).

```
ping google.com
PING google.com (172.217.14.78): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
^C
--- google.com ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
# Not able to connect/ping and there is a 100% loss.

ping someworkingurl.com
PING someworkingurl.com (100.10.14.1): 56 data bytes
64 bytes from 100.10.14.1: icmp_seq=0 ttl=46 time=284.899 ms
64 bytes from 100.10.14.9: icmp_seq=1 ttl=46 time=271.194 ms
^C
--- someworkingurl.com ping statistics ---
3 packets transmitted, 2 packets received, 33.3% packet loss
round-trip min/avg/max/stddev = 271.194/278.047/284.899/6.852 ms
# packets are transmitted, so it can be reached and also provides additional diagnostics info.
```

8. cURL a URL to verify if the URL is accessible. Check the response headers for the response code to see what has gone missing. Based on the response headers, validate what could have gone wrong. Here are some sample responses: 400 - Bad request(client side validation), 401 - Bad authorization, 302 - Redirect found.

```
curl -I https://oracle.com:443
HTTP/1.0 200 Connection established
```

```
HTTP/1.1 301 Moved Permanently
Date: Tue, 04 Apr 2023 15:07:52 GMT
Content-Type: text/html
Content-Length: 175
Connection: keep-alive
Location: https://www.oracle.com/

# Connection established to oracle.com which means the URL is accessible.
```

Using Security Adapters for Siebel CRM

This topic is part of *Deploying Siebel CRM on OCI*.

This section describes how to configure security adapters (security profile) provided with Siebel Business Applications.

SCM supports the configuration of the following security adapter types: DB and LDAP.

The SCM application sets authentication-related configuration parameters for Siebel Business Applications and Siebel Gateway authentication, but does not make changes to the LDAP directory. Make sure the configuration information you enter is compatible with your directory server.

When you specify LDAP as the security adapter type in the payload during environment provisioning, the setting you specify provides the value for the Enterprise Security Authentication Profile (Security Adapter Mode) parameter.

The Security Adapter Mode and Security Adapter Name (named subsystem) parameters can be set for:

- Siebel Gateway
- Siebel Enterprise Server
- All interactive Application Object Manager components

For more information, see the "Security Adapter Authentication" chapter in the Siebel Security Guide.

Use payload parameter `security_adapter_type` to specifiy the security adapter type. For more information, see *Payload Parameters for Siebel CRM Deployment*.

If you set the `security_adapter_type` to:

- DB, the details in the database payload section are used for configuring security adapter during environment provisioning.
- LDAP, you must pass the details in the `ldap` subsection under the `siebel` section.

Note:

- For greenfield environment or any lift bucket lifted from SCM version prior to CM_23.7.0, parameters under `siebel>ldap` sub-section of Payload Elements of SCM under *Payload Parameters for Siebel CRM Deployment* will be applicable.
- For lift bucket lifted using SCM version CM_23.7.0 or above and source environment is of security adapter type LDAP, then during shifting (using REST API for deployment), only below user credential parameters will be mandatory (since these information cannot be 'lifted') and rest are optional and it will be taken from lifted data if not passed in payload.
  - application_password
  - siebel_admin_username
  - siebel_admin_password
  - anonymous_username
  - anonymous_user_password

For greenfield environment, the value of `siebel_admin_username` must be `SADMIN` and value of `anonymous_username` must be `GUESTCST` since database will be having only greenfield data.

Example payload section specific to the case when `security_adapter_type` is `LDAP`. For complete sample payload structure, see *Payload Parameters for Siebel CRM Deployment*.

```
{
     "name": "test173",
     "siebel": {
     ....
     ....
     "security_adapter_type": "ldap",
     "ldap":
          {
          "ldap_host_name": <ldap_FQDN>,
          "ldap_port": "389",
          "application_user_dn": "cn=Directory Manager",
          "application_password": "ocid1.vaultsecret.oc1.uk-london-1.iaheyoqdfpc33khmp42wec",
          "base_dn": "ou=people,o=siebel.com",
          "shared_db_credentials_dn": "uid=sadmin,ou=people,o=siebel.com",
          "shared_db_username": "sadmn",
          "shared_db_password": "ocid1.vaultsecret.oc1.uk-london-1.tkyyppq733brnkhmp42wec",
          "password_attribute_type": "userPassword",
          "siebel_admin_username": "sadmin",
          "username_attribute_type": "uid",
          "credentials_attribute_type": "mail",
          "siebel_admin_password": "ocid1.vaultsecret.oc1.uk-london-1.amaaaaaa4n2rr5ia2wcc",
          "anonymous_username": "GUESTCST",
          "anonymous_user_password": "ocid1.vaultsecret.oc1.uk-london-1.amaaaaaa4n2rnkhmp42was",
          "use_adapter_username": "true",
          "siebel_username_attribute_type" : "uid",
          "propagate_change": "true",
          "hash_db_password": "true",
          "hash_user_password": "true",
          "salt_user_password": "true",
          "salt_attribute_type": "title"
          }
```

```
        }
        "infrastructure": {
        ....
        ....
```

Example payload section specific to the case when `security_adapter_type` is `LDAP` and `enable_ssl` is set to true (that is, for LDAPs). Note the change in `ldap_port` value. For complete sample payload structure, see *Payload Parameters for Siebel CRM Deployment*.

```
{
        "name": "test173",
        "siebel": {
        ....
        ....
        "security_adapter_type": "ldap",
        "ldap":
             {
             "ldap_host_name": <ldap_FQDN>,
             "ldap_port": "636",
             "application_user_dn": "cn=Directory Manager",
             "application_password": "ocid1.vaultsecret.oc1.uk-london-1.iaheyoqdfpc33khmp42wec",
             "base_dn": "ou=people,o=siebel.com",
             "shared_db_credentials_dn": "uid=sadmin,ou=people,o=siebel.com",
             "shared_db_username": "sadmn",
             "shared_db_password": "ocid1.vaultsecret.oc1.uk-london-1.tkyyppq733brnkhmp42wec",
             "password_attribute_type": "userPassword",
             "siebel_admin_username": "sadmin",
             "username_attribute_type": "uid",
             "credentials_attribute_type": "mail",
             "siebel_admin_password": "ocid1.vaultsecret.oc1.uk-london-1.amaaaaaa4n2rr5ia2wcc",
             "anonymous_username": "GUESTCST",
             "anonymous_user_password": "ocid1.vaultsecret.oc1.uk-london-1.amaaaaaa4n2rnkhmp42was",
             "use_adapter_username": "true",
             "siebel_username_attribute_type" : "uid",
             "propagate_change": "true",
             "hash_db_password": "true",
             "hash_user_password": "true",
             "salt_user_password": "true",
             "salt_attribute_type": "title"
             "enable_ssl": "true",
             "ldap_wallet_path": "/home/opc/siebel/ewallet.p12",
             "ldap_wallet_password": "ocid1.vaultsecret.oc1.uk-london-1.aaa4noqkyyppq7lf4oamvb7f2cxx"
             }
        }
        "infrastructure": {
        ....
        ....
```

Using Custom Keystore

SCM uses a custom keystore so you can manage and standardize the TLS certificates and private keys used by Siebel components (and any supporting services). When you deploy Siebel CRM with SCM, you must use two separate certificates: one for server authentication and another for client authentication. The keystore and truststore files are JKS files that store the certificates required for the Siebel TLS configuration:

- `keystore.jks`: Contains the Siebel server and Siebel Controller certificates.
- `keystore_client.jks`: Contains the Siebel client certificate.
- `truststore.jks`: Contains the root certificate.

The JKS files are necessary for the SCM container to use secure two-way communication when connecting with other Siebel CRM components. During Siebel CRM environment provisioning, SCM creates a self-signed certificate and propagates it to all application containers (SES, SAI, and CGW) and controller containers (Metacontroller and Siebel Controller).

You can also bring your own custom certificate. This option lets you keep your private keys within your organization's control, which helps you meet your compliance and security requirements.

> **Note:** We strongly recommend using a Certificate Authority (CA)-signed certificate. Using a CA-signed certificate provides greater control, flexibility, and security over certificate lifecycle management, and is considered a best practice in SCM.

To use a custom certificate:

1. Generate the custom certificate as follows:
   a. SSH into the SCM instance:
   ```
   ssh -i <ssh Private Key> opc@<SCM Host IP>
   ```
   b. Enter the container:
   ```
   sudo podman exec -it cloudmanager bash
   ```
   c. Create a directory to store the custom certificate and set the environment variable `ENV_DIR` to this directory. For example:
   ```
   ENV_DIR=/home/opc/customcerts
   ```
   d. Set the following environment variables:
   ```
   export HOSTNAME=<SCM_hostname>
   export NAMESPACE=<namespace>
   export PASSWORD=xxxxx
   export SUBDOMAIN=svc.cluster.local
   export KEYSTORE=${ENV_DIR}/ca/siebelcerts/siebelkeystore.jks
   export CLIENT_KEYSTORE=${ENV_DIR}/ca/siebelcerts/siebelkeystore_client.jks
   export SIEBEL_SERVER_KEYSTORE=${ENV_DIR}/ca/siebelcerts/keystore.jks
   export SIEBEL_CLIENT_KEYSTORE=${ENV_DIR}/ca/siebelcerts/keystore_client.jks
   export SIEBEL_TRUSTSTORE=${ENV_DIR}/ca/siebelcerts/truststore.jks
   export JRE_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
   export ALIAS=siebel
   export CONTROLLER_ALIAS='siebel-controller'
   export CN=${ENV_DIR##*/}-${NAMESPACE}
   ```

   The variables in the example have the following values:

   - `<SCM_hostname>` is the fully qualified domain name (FQDN) of the SCM host.
   - `<namespace>` is the value of the:
     - `namespace` field in the `<env_id>_environment.yaml` file for an existing environment.
     - `name` field in the Siebel CRM deployment payload for a new environment.

   e. Copy the `openssl.cnf.template` file as `openssl.cnf`, the `openssl_controller.cnf.template` file as `openssl_controller.cnf`, and the `ca` directory to ENV_DIR:
   ```
   cd ${ENV_DIR}
   cp -r /home/opc/siebel-cloud-manager/scripts/cmapp/initca/ca .
   cp ca/openssl.cnf.template ca/openssl.cnf
   cp ca/openssl_controller.cnf.template ca/openssl_controller.cnf
   ```
   f. Update the `openssl.cnf` and `openssl_controller.cnf` files:
   ```
   sed -i "s|{{NAMESPACE}}|${NAMESPACE}|g" ca/openssl.cnf
   sed -i "s|{{SUBDOMAIN}}|${SUBDOMAIN}|g" ca/openssl.cnf
   ```

```
sed -i "/{{EXTERNALDOMAIN}}/d" ca/openssl.cnf
sed -i "/{{IPADDRESS}}/d" ca/openssl.cnf
sed -i "s|{{NAMESPACE}}|${NAMESPACE}|g" ca/openssl_controller.cnf
```

g. Create a folder to store the certificate. For example:

```
mkdir -p ${ENV_DIR}/ca/siebelcerts
```

h. Generate the certificate as follows:

i. Generate a private key:

```
cd ${ENV_DIR}
openssl genrsa -out ca/private/ca.key.pem 2048
```

ii. Generate a CA file for the private key:

```
openssl req -config ca/openssl.cnf -x509 -sha256 -new -nodes -key ca/private/ca.key.pem -
days 1827 -out ca/certs/ca.cert.pem -subj "/CN=${CN}/C=US/ST=California/O=Oracle/L=Redwood
 Shores/OU=Siebel"
```

iii. Generate the JAVA keystore and keypair for Siebel server certificate:

```
${JRE_HOME}/bin/keytool -genkeypair -alias ${ALIAS} -validity 365 -keystore
 ${SIEBEL_SERVER_KEYSTORE} -storetype JKS -keyalg RSA -keysize 2048 -storepass ${PASSWORD}
 -keypass ${PASSWORD} -sigalg SHA256withRSA -dname "cn=${CN},ou=Siebel,o=Oracle,l=Redwood
 Shores,ST=California,c=US"
```

iv. Generate the Siebel server certificate:

```
${JRE_HOME}/bin/keytool -certreq -alias ${ALIAS} -keystore ${SIEBEL_SERVER_KEYSTORE} -file
 ca/csr/${ALIAS}.csr -storepass ${PASSWORD} -keypass ${PASSWORD}
openssl ca -batch -config ca/openssl.cnf -extensions server_cert -days 375 -notext -md
 sha256 -in ca/csr/${ALIAS}.csr -out ca/certs/${ALIAS}.cert.pem -passin pass:${PASSWORD}
```

v. Generate the keypair for Siebel Controller certificate:

```
${JRE_HOME}/bin/keytool -genkeypair -alias ${CONTROLLER_ALIAS} -validity 365 -keystore
 ${SIEBEL_SERVER_KEYSTORE} -storetype JKS -keyalg RSA -keysize 2048 -storepass ${PASSWORD}
 -keypass ${PASSWORD} -sigalg SHA256withRSA -dname "cn=${CN},ou=Siebel,o=Oracle,l=Redwood
 Shores,ST=California,c=US"
```

vi. Generate the Siebel Controller certificate:

```
${JRE_HOME}/bin/keytool -certreq -alias ${CONTROLLER_ALIAS} -keystore
 ${SIEBEL_SERVER_KEYSTORE} -file ca/csr/${CONTROLLER_ALIAS}.csr -storepass ${PASSWORD} -
keypass ${PASSWORD}
openssl ca -batch -config ca/openssl_controller.cnf -extensions server_cert -
days 375 -notext -md sha256 -in ca/csr/${CONTROLLER_ALIAS}.csr -out ca/certs/
${CONTROLLER_ALIAS}.cert.pem -passin pass:${PASSWORD}
```

vii. Generate the JAVA keystore and keypair for the client certificate:

```
${JRE_HOME}/bin/keytool -genkeypair -alias ${ALIAS} -validity 365 -keystore
 ${SIEBEL_CLIENT_KEYSTORE} -storetype JKS -keyalg RSA -keysize 2048 -storepass ${PASSWORD}
 -keypass ${PASSWORD} -sigalg SHA256withRSA -dname "cn=${CN},ou=Siebel,o=Oracle,l=Redwood
 Shores,ST=California,c=US"
```

viii. Generate the Siebel client certificate:

```
${JRE_HOME}/bin/keytool -certreq -alias ${ALIAS} -keystore ${SIEBEL_CLIENT_KEYSTORE} -file
 ca/csr/${ALIAS}_client.csr -storepass ${PASSWORD} -keypass ${PASSWORD}
openssl ca -batch -config ca/openssl.cnf -extensions usr_cert -days 375 -notext -md sha256
 -in ca/csr/${ALIAS}_client.csr -out ca/certs/${ALIAS}_client.cert.pem -passin pass:
${PASSWORD}
```

ix. Verify the certificate details:

```
openssl x509 -noout -text -in ca/certs/${ALIAS}.cert.pem
openssl verify -CAfile ca/certs/ca.cert.pem ca/certs/${ALIAS}.cert.pem
openssl x509 -noout -text -in ca/certs/${ALIAS}_client.cert.pem
openssl verify -CAfile ca/certs/ca.cert.pem ca/certs/${ALIAS}_client.cert.pem
openssl x509 -noout -text -in ca/certs/${CONTROLLER_ALIAS}.cert.pem
```

```
openssl verify -CAfile ca/certs/ca.cert.pem ca/certs/${CONTROLLER_ALIAS}.cert.pem
```

x. Import the certificate PEM files (`ca.cert.pem`, `siebel.cert.pem`, and `siebel-controller.cert.pem` to `keystore.jks`):

```
${JRE_HOME}/bin/keytool -import -trustcacerts -keystore ${SIEBEL_SERVER_KEYSTORE} -alias
 root -file ca/certs/ca.cert.pem -storepass ${PASSWORD} -keypass ${PASSWORD} -noprompt
${JRE_HOME}/bin/keytool -importcert -alias ${ALIAS} -file ca/certs/${ALIAS}.cert.pem -
keystore ${SIEBEL_SERVER_KEYSTORE} -storepass ${PASSWORD} -keypass ${PASSWORD} -noprompt
${JRE_HOME}/bin/keytool -importcert -alias ${CONTROLLER_ALIAS} -file ca/certs/
${CONTROLLER_ALIAS}.cert.pem -keystore ${SIEBEL_SERVER_KEYSTORE} -storepass ${PASSWORD} -
keypass ${PASSWORD} -noprompt
```

xi. Import the certificate PEM files (`ca.cert.pem` and `siebel_client.cert.pem`) to `keystore_client.jks`:

```
${JRE_HOME}/bin/keytool -import -trustcacerts -keystore ${SIEBEL_CLIENT_KEYSTORE} -alias
 root -file ca/certs/ca.cert.pem -storepass ${PASSWORD} -keypass ${PASSWORD} -noprompt
${JRE_HOME}/bin/keytool -importcert -alias ${ALIAS} -file ca/certs/${ALIAS}_client.cert.pem
 -keystore ${SIEBEL_CLIENT_KEYSTORE} -storepass ${PASSWORD} -keypass ${PASSWORD} -noprompt
```

xii. Import `ca.cert.pem` and generate `truststore.jks`:

```
${JRE_HOME}/bin/keytool -import -trustcacerts -keystore ${SIEBEL_TRUSTSTORE} -alias root -
file ca/certs/ca.cert.pem -storepass ${PASSWORD} -keypass ${PASSWORD} -noprompt
```

xiii. Copy `keystore.jks` to `siebelkeystore.jks`:

```
cp ${SIEBEL_SERVER_KEYSTORE} ${KEYSTORE}
cp ${SIEBEL_CLIENT_KEYSTORE} ${CLIENT_KEYSTORE}
cat ca/certs/ca.cert.pem ca/certs/${ALIAS}.cert.pem > ${ENV_DIR}/ca/siebelcerts/ca-
chain.cert.pem
cat ca/certs/ca.cert.pem ca/certs/${ALIAS}_client.cert.pem > ${ENV_DIR}/ca/siebelcerts/ca-
chain_client.cert.pem
```

2. Configure the keystore section under the `siebel` section in the Siebel CRM deployment payload to reference the `keystore.jks`, `keystore_client.jks`, and `truststore.jks` files.

   Here's a sample of the `keystore` section in the initial deployment payload:

```
"keystore":
{
 "siebel_server_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore.jks",
 "siebel_server_keystore_password": "xxxxxx",
 "siebel_client_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore_client.jks",
 "siebel_client_keystore_password": "xxxxxx",
 "siebel_truststore_path": "/home/opc/customcerts/ca/siebelcerts/truststore.jks",
 "siebel_truststore_password": "xxxxx",
 "siebel_server_certificate_alias": "siebel",
 "siebel_client_certificate_alias": "siebel",
 "siebel_controller_certificate_alias": "siebctrlcert"
}
```

> **Note:** If the `keystore` section isn't configured in the Siebel CRM deployment payload, SCM generates a self-signed certificate and uses it.

During environment provisioning, the JKS files are pushed to the Helm charts Git repository in the `siebel- config/keystore` directory, which is used by Siebel CRM applications. Additionally, Siebel Controller certificates from the JKS files are pushed to the `metacontroller/tls` directory in Helm charts Git repository and are used by Metacontroller and Siebel Controller.

You need to follow these rules when creating custom keystore and truststore files:

- Use the .jks file extension for the Siebel server keystore, Siebel client keystore, and truststore files and set the storeType to JKS.

- Configure the keystore certificate with the DNS as "*.svc.cluster.local" along with other DNS entries.
- Import the Siebel certificate into `keystore.jks` using the alias `siebel`.
- Import the Siebel client certificate into `keystore_client.jks` using the alias `siebel`.
- Configure `keystore.jks` for the Siebel Controller certificate with the DNS name as `siebel-controller.<namespace>.svc.cluster.local`. Here, `<namespace>` is the environment name.
- Sign the Siebel Controller certificate with the same root certificate used to sign Siebel server and client certificates.
- Import the Siebel Controller certificate into `keystore.jks` using an alias other than `siebel`.
- Create the certificates with a password.
- Include the CA certificate, any intermediate certificates, and CSR certificate information in `keystore.jks`.
- Include the Siebel client CA certificate, any intermediate certificates, and CSR certificate information in `keystore_client.jks`.
- Ensure the Siebel server, Siebel client, and Siebel Controller certificates have only the serverAuth Extended_Key_Usage.
- Include the CA certificate information in the truststore file.

For more information about updating Keystore file as part of incremental changes, see *Use Cases for Updating Keystore File as Part of Incremental Changes*.

Terminating SSL/TLS at the Load Balancer (FrontEnd SSL) using SCM

Note: This section is valid only for Siebel CRM environments provisioned with SCM version CM_23.8.1 or higher.

When Container Engine for Kubernetes provisions a load balancer for a Kubernetes service of type LoadBalancer, you can specify that you want to terminate SSL at the load balancer. This configuration is known as frontend SSL. To implement frontend SSL, we have to define a listener at a port such as 443, and associate an SSL certificate with the listener.

Load balancers commonly use single domain certificates. However, load balancers with listeners that include request routing configuration might require a subject alternative name (SAN) certificate (also called multi-domain certificate) or a wildcard certificate. The Load Balancing service supports each of these certificate types.

Oracle Cloud Infrastructure (OCI) accepts x.509 type certificates in PEM format only. The following is an example PEM encoded certificate:

```
-----BEGIN CERTIFICATE-----
<Base64_encoded_certificate>
-----END CERTIFICATE-----
```

To terminate SSL certificate at the load balancer with custom SSL certificate, you must supply a certificate during environment provisioning using the following payload parameters:

- load_balancer_ssl_cert_path
- load_balancer_private_key_path
- load_balancer_private_key_password

For more information, see *Payload Parameters for Siebel CRM Deployment*. If the above optional parameters are not provided during environment provisioning, SCM will generate a self-signed certificate and associate the same with the load balancer listener through Traefik Ingress service.

Updating SSL/TLS Certificates for an Existing Load Balancer Post Deployment

Solution 1 - Updating certificates to existing Load Balancer from OCI Console

1. Go to the OCI console and navigate to Load Balancer service.
2. Go to the Load Balancer of the current environment.
3. Click on **Certificates** on left menu and select **Load Balancer Managed certificate** in the **Certificate resource** dropdown.
4. Click on **Add certificate** and upload SSL certificate and private key in respective fields.
5. Go to the listeners from left menu and edit the listener with name 'TCP-443'.
6. Select **Load Balancer Managed certificate** in the **Certificate resource** dropdown.
7. Select the new load balancer certificate in the 'certificate name' dropdown.

Solution 2 - Updating certificates and creating new Load Balancer using GitOps setup.

- If private key is encrypted, first decrypt it using the command:
  ```
  openssl rsa -in <load_balancer_private_key_path> -out <decrypted_load_balancer_private_key_path>
  ```
- Create a Kubernetes TLS secret for load balancer SSL certificate using the command:
  ```
  kubectl create secret tls lb-tls-certificate --key <decrypted_load_balancer_private_key_path> --cert
   <load_balancer_ssl_cert_path> -n <namespace>
  ```

  > **Note:** If `lb-tls-certificate` is already present, you need to delete it first using command:
  > `'kubectl delete secret lb-tls-certificate -n <namespace>'`

- Update the SSL certificate in Ingress definition:
  a. SSH into the SCM instance.
  b. Enter the container:
     ```
     sudo podman exec -it cloudmanager bash
     ```
  c. Go to the Traefik resources directory:
     ```
     cd /home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/traefik/traefik-resources/
     ```
  d. Edit the `tlsStore.yaml` file.
  e. Update `secretName` under `defaultCertificate` to `lb-tls-certificate` if not present.
  f. Push your changes to remote git repository:
     ```
     git add .
     git commit -m "<message>"
     git push
     ```

Traefik detects the new secret and reloads it automatically.

> **Note:** To replace an existing load balancer TLS certificate for an existing Siebel CRM environment that uses ingress-nginx in SCM 26.2 or earlier, see *Rotating the Load Balancer TLS Certificate for Ingress-Nginx*.

Auto-enablement of Siebel Migration Application

This topic is part of *Deploying Siebel CRM on OCI*.

The Siebel Migration application is a Web-based tool for migrating Siebel Repositories and seed data and performing related tasks, which is provided with the Siebel Application Interface (SAI) installation.

An environment deployed through "lift-and-shift" mechanisms using the lift tool and SCM has the Siebel Migration application auto-enabled in the deployed Siebel CRM environment. Once the deployment is done, Siebel Migration Application endpoint will be included in the URL list with a form ending with /siebel/migration. Use the `migration_package_mt_export_path` parameter described in *Payload Parameters for Siebel CRM Deployment*.

> **Note:** Follow SCM Incremental changes model for migrating web artifacts like image files, javascript files, and so on. For more information about the activities that you can perform in the Siebel Management Console (SMC) post-deployment, refer Siebel Bookshelf. For more information about troubleshooting, see *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

Preparing Siebel File System and Migration Directories

When deploying Siebel CRM, you must prepare the Siebel File System directories before deployment.

> **Note:** You need to perform this procedure only when you use BYO file systems. This applies when the deployment payload specifies existing Siebel File System or migration export paths under `infrastructure.mounttarget_exports`. If SCM creates and manages the file systems for the deployment, you do not need to manually mount the file systems or create these directories.

To prepare the file system directories:

1. Mount the Siebel File System on a host that can access the mount target. Mount both the Siebel File System export path and the migration export path.

   ```
   sudo mount -t nfs -o nfsvers=3,nosuid,resvport -o nolock <mount_target_ip>:/<sfs_export_path>
    <path_on_your_host_to_mount_sfs>

   sudo mount -t nfs -o nfsvers=3,nosuid,resvport -o nolock <mount_target_ip>:/<migration_export_path>
    <path_on_your_host_to_mount_migration>
   ```

2. Create the Siebel File System base directory, as follows:
   a. Go to the mounted Siebel File System location:
      ```
      cd <path_on_your_host_to_mount_sfs>
      ```
   b. Create the base directory:
      ```
      sudo mkdir -p <SIEBEL_NAMESPACE>/FS
      ```
   c. Assign ownership to the configured user ID and group `0`:
      ```
      sudo chown -R <run_as_user_value>:0 <SIEBEL_NAMESPACE>
      ```
   d. Grant group read, write, and execute permissions:
      ```
      sudo chmod g+rwx <SIEBEL_NAMESPACE>
      ```

3. Create the base directory for the migration file system, as follows:
   a. Go to the mounted migration location:
      ```
      cd <path_on_your_host_to_mount_migration>
      ```
   b. Create the migration directory:
      ```
      sudo mkdir migration
      ```
   c. Assign ownership to the configured user ID and group `0`:
      ```
      sudo chown -R <run_as_user_value>:0 migration
      ```
   d. Grant group read, write, and execute permissions:
      ```
      sudo chmod g+rwx migration
      ```

In the above examples:

- `<SIEBEL_NAMESPACE>` is the Siebel CRM namespace name in uppercase.
- `<run_as_user_value>` is the value specified for `siebel.security_context.run_as_user` in the Siebel CRM provisioning payload. If you do not specify `siebel.security_context`, the default value is `1000`.

Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM

You can use a customer-managed OCI Vault Key Management Service (KMS) key to encrypt selected SCM-created resources. You can use this option when you want to manage your own encryption keys for compliance, security, or operational control.

To use a customer-managed OCI Vault KMS key, you must:

- BYO KMS key to encrypt SCM-created boot volumes and, when SCM creates the SCM File Storage.
- Set `infrastructure.siebel_storage_kms_key_ocid` to encrypt Siebel file systems created by SCM.
- Set `infrastructure.kubernetes.oke.kms_key_id` to encrypt Kubernetes secrets when SCM creates a new OKE cluster. This option applies only when `infrastructure.kubernetes.kubernetes_type` is set to `OKE`. It does not modify existing or BYO OKE clusters.

Before you provision the environment, complete the required IAM prerequisites. For more information, see *IAM Prerequisites for Using KMS Keys*.

If you do not provide the KMS key ID, SCM uses Oracle-managed keys and preserves the default OCI encryption behavior.

IAM Prerequisites for Using KMS Keys

Before you provision SCM, you must prepare the following resources:

- An active OCI Vault KMS key in the target region.
- A defined tag namespace, key, and value when SCM-created file storage or OKE uses a customer-managed KMS key.
- A dynamic group that matches the SCM-created file storage or OKE resource by the defined tag:
  - Use the following dynamic group rule for SCM-created file storage:
    ```
    ALL { resource.type = 'filesystem', tag.<tag-namespace>.<tag-key>.value = '<tag-value>' }
    ```
  - Use the following dynamic group rule for SCM-created OKE:

```
ALL { resource.type = 'cluster', tag.<tag-namespace>.<tag-key>.value = '<tag-value>' }
```

- An IAM policy that allows the dynamic group and the OCI File Storage service to use the KMS key:

```
Allow dynamic-group <fss-dynamic-group-name> to use keys in compartment <kms-key-compartment-name>
Allow service FssOc<n>Prod to use keys in compartment <kms-key-compartment-name> where target.key.id =
 '<kms-key-ocid>'
```

- For SCM boot volume encryption, an IAM policy that allows OCI Block Volume to use the KMS key:

```
Allow service blockstorage to use keys in compartment <kms-key-compartment-name> where target.key.id =
 '<kms-key-ocid>'
```

- For OKE, an IAM policy that authorizes the OKE cluster resource principal to use the KMS key:

```
Allow dynamic-group <dynamic-group-name> to use keys in compartment <kms-key-compartment-name>
```

Note:

- SCM does not create the dynamic group or IAM policy. Before you submit the SCM provisioning request, the tenancy administrator must create dynamic group and policies. You must allow time for dynamic group membership to propagate in OCI.
- The OKE policy applies to the OKE cluster resource principal, not to the SCM user profile or SCM instance principal.

KMS Key Rotation

When you rotate an OCI Vault encryption key, OCI creates a new key version. The KMS key OCID does not change.

New encryption operations use the current key version. Data and secrets encrypted with an earlier key version remain accessible because OCI Vault retains the earlier key version for decryption.

Key rotation does not automatically re-encrypt existing data or secrets. To use the new key version for existing encrypted content, you must perform the applicable service or application re-encryption, rewrite, or rotation procedure. You must also plan and validate this operation before you use it in production.

Note: You must not disable, schedule deletion of, or delete an earlier key version while it is still required to decrypt existing data.

If you switch to a different encryption key, and not just to a new version of the same key, update the relevant configuration to use the new key OCID.

For more information, see the OCI Vault key rotation documentation.

Executing the Payload to Deploy Siebel CRM

This topic describes how to execute the payload to deploy Siebel CRM. This topic is part of *Deploying Siebel CRM on OCI*.

To execute the payload to deploy Siebel CRM

1. Create an application/json body with the payload information. For an example, see *Example Payload to Deploy Siebel CRM*.
2. Do a POST API like the following:

```
POST https://<CM_Instance_IP>:16690/scm/api/v1.0/environment
```

> **Note:** Specify a payload appropriate for your use case. For an example payload and for usage guidelines, see *Example Payload to Deploy Siebel CRM*.

3. Use Basic Auth and provide credentials like the following:

```
User: "admin"

Password: "<Password available in the file /home/opc/cm_app/{CM_RESOURCE_PREFIX}/config/api_creds.ini>"
```

Environment information is displayed. Copy the `selfLink` value for monitoring purposes. For example:

```
"selfLink": "https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/4ZZYX5"
```

Example Payload to Deploy Siebel CRM

To deploy Siebel CRM on OCI, you can prepare and execute the payload based on the following guidelines:

- To deploy Siebel CRM with the default configuration (greenfield deployment use case 1), omit the `config_id` parameter.
- To create a Siebel CRM configuration to customize (greenfield deployment use case 2), use the `POST` API command in *Creating the Configuration and Obtaining the Configuration ID*. Make sure you include all payload parameters used in the greenfield deployment for use case 1.
- To deploy Siebel CRM with a customized configuration (greenfield deployment use case 2), use the `POST` API command in *Executing the Payload to Deploy Siebel CRM*. In the payload only include the `config_id` parameter (set to the configuration ID you obtained when you created the configuration) and name parameter. Omit all other parameters.
- For usage guidance on additional parameters required for the lift and shift use case or for greenfield deployments, see *Payload Parameters for Siebel CRM Deployment*.

This section contains the following topics:

- *Example Payload when "Do not use Vault" Checkbox is Selected*
- *Example Payload when "Use existing resources" Checkbox is Not Selected*
- *Example Payload when "Use existing VCN" Checkbox is Selected*
- *Example Payload when "Use existing resources" Checkbox is Selected*
- *Example Database Sections for DBCS_VM Database Type*
- *Example for BYO Database*
- *Example Kubernetes Cluster Sections for BYO-Kubernetes*
- *Example Git Section for BYO-Git*
- *Example for BYO Namespace*
- *Example for Configuring a Standalone Siebel Gateway*
- *Example for Using Resource Tags*
- *Example for Using Customer-Managed KMS Keys for Infrastructure Resources Created by SCM*

Example Payload when "Do not use Vault" Checkbox is Selected

```
{
     "config_id": "<config_id of custom configuration>",
     "name": "DevExample",
     "siebel": {
          "registry_url": "iad.ocir.io",
          "siebel_architecture": "CRM",
          "registry_user": "deploygroup/user.name@example.com",
          "registry_password": "<xxxxxx>",
          "bucket_url": "https://objectstorage.us-example-1.oraclecloud.com/p/s0EgeDE9-
 NMc2lTazIY3LuXO1IbGx5ASAilKxJexLHNjirdl4AKJh8RBxou1J4S1/n/deploygroup/b/bucket_example/o/",
          "keystore" : {
               "siebel_server_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore.jks",
               "siebel_server_keystore_password": "xxxxxx",
               "siebel_client_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore_client.jks",
               "siebel_client_keystore_password": "xxxxxx",
               "siebel_truststore_path": "/home/opc/customcerts/ca/siebelcerts/truststore.jks",
               "siebel_truststore_password": "xxxxx",
               "siebel_server_certificate_alias": "siebel",
               "siebel_client_certificate_alias": "siebel",
               "siebel_controller_certificate_alias": "siebctrlcert"
          }
     },
     "infrastructure": {
          "git": {
               "git_type": "gitlab",
               "gitlab": {
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<yyyyy>",
                    "git_user": "<username>",
                    "git_selfsigned_cacert": "/home/opc/certs/rootCA.crt"
               }
          },
          "siebel_cluster_subnet_ocid": "<cluster_subnet_ocid>",
          "siebel_lb_subnet_ocid": "<lb_subnet_ocid>",
          "siebel_private_subnet_ocid": "<private_subnet_ocid>",
          "siebel_db_subnet_ocid": "<db_subnet_ocid>",
          "vcn_ocid_of_db_subnet": "<VCN_ocid_of_worker_node>",
          "load_balancer_type": "public",
          "kubernetes": {
               "kubernetes_type": "OKE",
               "oke": {
                    "oke_node_count": 3,
                    "oke_node_shape": "VM.Standard.E4.Flex",
                    "oke_node_shape_config": {
                         "memory_in_gbs": 60,
                         "ocpus": 4
                    }
               }
          }
     },
     "database": {
          "db_type": "ATP",
          "atp": {
               "admin_password": "<Plain-text of your admin password>",
               "storage_in_tbs": 1,
               "cpu_cores": 3,
               "wallet_password": "<Plain-text of your wallet password's secret>"
          },
          "auth_info": {
               "siebel_admin_username": "<provide your own values>",
               "siebel_admin_password": "<Your Siebel admin password's secret in plain-text>",
               "default_user_password": "<Your default user password's secret in plain-text>",
               "table_owner_password": "<Your table owner password's secret in plain-text>",
```

```
                "table_owner_user": "<provide your own values>",
                "anonymous_user_password": "<Your anonymous user password's secret in plain-text>"
           }
      },
      "size": {
           "ses_resource_limits": {
                "cpu": "2",
                "memory": "4Gi"
           },
      "ses_resource_requests": {
                "cpu": "1.0",
                "memory": "4Gi"
           },
      "cgw_resource_limits": {
                "cpu": "2",
                "memory": "4Gi"
           },
      "cgw_resource_requests": {
                "cpu": "1000m",
                "memory": "4Gi"
           },
      "sai_resource_limits": {
                "cpu": "1",
                "memory": "4Gi"
           },
      "sai_resource_requests": {
                "cpu": "1",
                "memory": "4Gi"
           }
      }
}
```

Example Payload when "Use existing resources" Checkbox is Not Selected

```
{
     "config_id": "<config_id of custom configuration>",
     "name": "DevExample",
     "siebel": {
          "registry_url": "iad.ocir.io",
          "siebel_architecture": "CRM",
          "registry_user": "deploygroup/user.name@example.com",
          "registry_password": "<xxxxxx>",
          "bucket_url": "https://objectstorage.us-example-1.oraclecloud.com/p/s0EgeDE9-
NMc2lTazIY3LuXO1IbGx5ASAilKxJexLHNjirdl4AKJh8RBxou1J4S1/n/deploygroup/b/bucket_example/o/",
          "keystore" : {
               "siebel_server_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore.jks",
               "siebel_server_keystore_password": "xxxxxx",
               "siebel_client_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore_client.jks",
               "siebel_client_keystore_password": "xxxxxx",
               "siebel_truststore_path": "/home/opc/customcerts/ca/siebelcerts/truststore.jks",
               "siebel_truststore_password": "xxxxx",
               "siebel_server_certificate_alias": "siebel",
               "siebel_client_certificate_alias": "siebel",
               "siebel_controller_certificate_alias": "siebctrlcert"
          }
     },
     "infrastructure": {
          "git": {
               "git_type": "gitlab",
               "gitlab": {
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<yyyyy>",
                    "git_user": "<user.name>",
                    "git_selfsigned_cacert": "/home/opc/certs/rootCA.crt"
               }
```

```
            },
            "siebel_lb_subnet_cidr" : "10.0.1.0/24",
            "siebel_private_subnet_cidr" : "10.0.2.0/24",
            "siebel_db_subnet_cidr" : "10.0.3.0/24",
            "siebel_cluster_subnet_cidr" : "10.0.4.0/24",
            "load_balancer_type": "public",
            "kubernetes": {
                 "kubernetes_type": "OKE",
                 "oke": {
                      "oke_node_count": 3,
                      "oke_node_shape": "VM.Standard.E3.Flex",
                      "oke_node_shape_config": {
                           "memory_in_gbs": "60",
                           "ocpus": "4"
                      }
                 }
            }
       },
       "database": {
            "db_type": "ATP",
            "atp": {
                 "admin_password": "<OCID of your admin password>",
                 "storage_in_tbs": 1,
                 "cpu_cores": 3,
                 "wallet_password": "<OCID of your wallet password's secret>"
            },
       "auth_info": {
            "siebel_admin_username": "<provide your own values>",
            "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
            "default_user_password": "<OCID of your default user password's secret>",
            "table_owner_password": "<OCID of your table owner password's secret>",
            "table_owner_user": "<provide your own values>",
            "anonymous_user_password": "<OCID of your anonymous user password's secret>"
            }
       },
       "size": {
            "ses_resource_limits": {
                 "cpu": "2",
                 "memory": "4Gi"
            },
            "ses_resource_requests": {
                 "cpu": "1.0",
                 "memory": "4Gi"
            },
            "cgw_resource_limits": {
                 "cpu": "2",
                 "memory": "4Gi"
            },
            "cgw_resource_requests": {
                 "cpu": "1000m",
                 "memory": "4Gi"
            },
            "sai_resource_limits": {
                 "cpu": "1",
                 "memory": "4Gi"
            },
            "sai_resource_requests": {
                 "cpu": "1",
                 "memory": "4Gi"
            }
       }
  }
```

Example Payload when "Use existing VCN" Checkbox is Selected

```
{
     "config_id": "<config_id of custom configuration>",
     "name": "DevExample",
     "siebel": {
          "registry_url": "iad.ocir.io",
          "siebel_architecture": "CRM",
          "registry_user": "deploygroup/user.name@example.com",
          "registry_password": "<xxxxxx>",
          "bucket_url": "https://objectstorage.us-example-1.oraclecloud.com/p/s0EgeDE9-
 NMc2lTazIY3LuXO1IbGx5ASAilKxJexLHNjirdl4AKJh8RBxou1J4S1/n/deploygroup/b/bucket_example/o/",
          "keystore" :
               {
               "siebel_server_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore.jks",
               "siebel_server_keystore_password": "xxxxxx",
               "siebel_client_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore_client.jks",
               "siebel_client_keystore_password": "xxxxxx",
               "siebel_truststore_path": "/home/opc/customcerts/ca/siebelcerts/truststore.jks",
               "siebel_truststore_password": "xxxxx",
               "siebel_server_certificate_alias": "siebel",
               "siebel_client_certificate_alias": "siebel",
               "siebel_controller_certificate_alias": "siebctrlcert"
               }
          },
     "infrastructure": {
          "git": {
               "git_type": "gitlab",
               "gitlab": {
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<yyyyy>",
                    "git_user": "<username>",
                    "git_selfsigned_cacert": "/home/opc/certs/rootCA.crt"
               }
          },
          "siebel_cluster_subnet_ocid": "<cluster_subnet_ocid>",
          "siebel_lb_subnet_ocid": "<lb_subnet_ocid>",
          "siebel_private_subnet_ocid": "<private_subnet_ocid>",
          "siebel_db_subnet_ocid": "<db_subnet_ocid>",
          "vcn_ocid_of_db_subnet": "<VCN_ocid_of_worker_node>",
          "load_balancer_type": "public",
          "kubernetes": {
               "kubernetes_type": "OKE",
               "oke": {
                    "oke_node_count": 3,
                    "oke_node_shape": "VM.Standard.E3.Flex",
                    "oke_node_shape_config": {
                         "memory_in_gbs": "60",
                         ocpus": "4"
                    }
               }
          }
     },
     "database": {
          "db_type": "ATP",
          "atp": {
               "admin_password": "<OCID of your admin password>",
               "storage_in_tbs": 1,
               "cpu_cores": 3,
               "wallet_password": "<OCID of your wallet password's secret>"
               },
     "auth_info": {
          "siebel_admin_username": "<provide your own values>",
          "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
          "default_user_password": "<OCID of your default user password's secret>",
```

```
            "table_owner_password": "<OCID of your table owner password's secret>",
            "table_owner_user": "<provide your own values>",
            "anonymous_user_password": "<OCID of your anonymous user password's secret>"
            }
      },
      "ses_resource_limits": {
            "cpu": "2",
            "memory": "4Gi"
            },
      "ses_resource_requests": {
            "cpu": "1.0",
            "memory": "4Gi"
            },
      "cgw_resource_limits": {
            "cpu": "2",
            "memory": "4Gi"
            },
      "cgw_resource_requests": {
            "cpu": "1000m",
            "memory": "4Gi"
            },
      "sai_resource_limits": {
            "cpu": "1",
            "memory": "4Gi"
            },
      "sai_resource_requests": {
            "cpu": "1",
            "memory": "4Gi"
            }
      }
}
```

Example Payload when "Use existing resources" Checkbox is Selected

The following is an example payload sent to SCM to deploy Siebel CRM using user provided inputs regarding existing infrastructure for Siebel CRM deployment. Specific section, for example for OKE, for mount target etc. are further given as separate examples in the subsequent sections.

```
{
      "name": "test1",
      "siebel": {
            "siebel_architecture": "CRM",
            "registry_url": "iad.ocir.io",
            "registry_user": "<registry_user>",
            "registry_password": "<registry_password>",
            "database_type": "Vanilla",
            "industry": "Telecommunications",
            "keystore" :
                  {
                        "siebel_server_keystore_path": "/home/opc/customcerts/ca/siebelcerts/keystore.jks",
                        "siebel_server_keystore_password": "xxxxxx",
                        "siebel_client_keystore_path": "/home/opc/customcerts/ca/siebelcerts/
keystore_client.jks",
                        "siebel_client_keystore_password": "xxxxxx",
                        "siebel_truststore_path": "/home/opc/customcerts/ca/siebelcerts/truststore.jks",
                        "siebel_truststore_password": "xxxxx",
                        "siebel_server_certificate_alias": "siebel",
                        "siebel_client_certificate_alias": "siebel",
                        "siebel_controller_certificate_alias": "siebctrlcert"
                  }
      },
      "infrastructure": {
            "git": {
                  "git_type": "gitlab",
                  "gitlab": {
```

```
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<gitlab token>",
                    "git_user": "root",
                    "git_selfsigned_cacert": "/home/opc/certs/rootCA.crt"
               }
          },
          "load_balancer_type": "public",
          "siebel_lb_subnet_cidr" : "10.0.1.0/24",
          "siebel_private_subnet_cidr" : "10.0.2.0/24",
          "siebel_db_subnet_cidr" : "10.0.3.0/24",
          "siebel_cluster_subnet_cidr" : "10.0.4.0/24",
          "kubernetes": {
               "kubernetes_type": "BYO_OKE",
               "byo_oke": {
                    "oke_cluster_id": "<cluster-ocid>",
                    "oke_endpoint": "PRIVATE",
                    "oke_kubeconfig_path": "<path-to-kubeconfig-file>"
                    "byo_ns": true
               }
          },
          "mounttarget_exports": {
               "siebfs_mt_export_paths": [
                    {"mount_target_private_ip" : "10.0.255.171","export_path": "/siebfs0"}
               ]
          }
     },
     "database": {
          "db_type": "BYOD",
          "byod": {
               "wallet_path": "/home/opc/certs/wallet",
               "tns_connection_name": "<provide tns connection name value>"
          },
          "auth_info": {
               "siebel_admin_username": "<provide your own values>",
               "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
               "default_user_password": "<OCID of your default user password's secret>",
               "table_owner_password": "<OCID of your table owner password's secret>",
               "table_owner_user": "<provide your own values>",
               "anonymous_user_password": "<OCID of your anonymous user password's secret>"
               }
          },
     "size": {
          "ses_resource_limits": {
               "cpu": "2",
               "memory": "4Gi"
          },
          "ses_resource_requests": {
               "cpu": "1.0",
               "memory": "4Gi"
          },
          "cgw_resource_limits": {
               "cpu": "2",
               "memory": "4Gi"
          },
          "cgw_resource_requests": {
               "cpu": "1000m",
               "memory": "4Gi"
          },
          "sai_resource_limits": {
               "cpu": "1",
               "memory": "4Gi"
          },
          "sai_resource_requests": {
               "cpu": "1",
               "memory": "4Gi"
          }
```

```
        }
    }
```

Note: When using an existing namespace (that is when `byo_ns` is set to `true`), you must ensure that the existing namespace name matches the `name` field in the deployment payload.

Example Database Sections for DBCS_VM Database Type

The following is an example database section of the payload for the DBCS_VM database type, using a VM standard shape type:

```
    "database": {
              "db_type": "DBCS_VM",
              "dbcs_vm": {
                   "db_version": "21.0.0.0",
                   "database_edition": "ENTERPRISE_EDITION_HIGH_PERFORMANCE",
                   "availability_domain": "1",
                   "db_home_admin_password": "<OCID of your db home admin password's secret>",
                   "shape": "VM.Standard1.1",
                   "data_storage_size_in_gbs": "512",
                   "db_admin_username": "<provide your own values>",
                   "db_admin_password": "OCID of your db admin password's secret"
              }
              "auth_info": {
                   "siebel_admin_username": "<provide your own values>",
                   "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
                   "default_user_password": "<OCID of your default user password's secret>",
                   "table_owner_password": "<OCID of your table owner password's secret>",
                   "table_owner_user": "<provide your own values>",
                   "anonymous_user_password": "<OCID of your anonymous user password's secret>"
              }
         },
```

The following is an example database section of the payload for the DBCS_VM database type, using a VM flex shape type:

```
    "database": {
              "db_type": "DBCS_VM",
              "dbcs_vm": {
                   "db_version": "21.0.0.0",
                   "database_edition": "ENTERPRISE_EDITION_HIGH_PERFORMANCE",
                   "availability_domain": "1",
                   "db_home_admin_password": "<OCID of your db home admin password's secret>",
                   "shape": "VM.Standard.E4.Flex",
                   "cpu_count": "2",
                   "data_storage_size_in_gbs": "512",
                   "db_admin_username": "<provide your own values>",
                   "db_admin_password": "OCID of your db admin password's secret"
              }
              "auth_info": {
                   "siebel_admin_username": "<provide your own values>",
                   "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
                   "default_user_password": "<OCID of your default user password's secret>",
                   "table_owner_password": "<OCID of your table owner password's secret>",
                   "table_owner_user": "<provide your own values>",
                   "anonymous_user_password": "<OCID of your anonymous user password's secret>"
              }
         },
```

The following is an example database section of the payload for the DBCS_VM database type, using BYO-FS with payload parameter `dbcs_vm > mount_target_ip` and `export_path` included:

```
    "database": {
```

```
            "db_type": "DBCS_VM",
            "dbcs_vm": {
                "db_version": "21.0.0.0",
                "database_edition": "ENTERPRISE_EDITION_HIGH_PERFORMANCE",
                "availability_domain": "1",
                "db_home_admin_password": "<OCID of your db home admin password's secret>",
                "shape": "VM.Standard.E4.Flex",
                "cpu_count": "2",
                "data_storage_size_in_gbs": "512",
                "db_admin_username": "<provide your own values>",
                "db_admin_password": "<OCID of your db admin password's secret>",
                "mount_target_private_ip": "<IP address of your mount target>",
                "export_path": "<Export path in the mount target for using in DATA DIR>"
            },
            "auth_info": {
                "siebel_admin_username": "<provide your own values>",
                "siebel_admin_password": "<OCID of your Siebel admin password's secret>",
                "default_user_password": "<OCID of your default user password's secret>",
                "table_owner_password": "<OCID of your table owner password's secret>",
                "table_owner_user": "<provide your own values>",
                "anonymous_user_password": "<OCID of your anonymous user password's secret>"
            }
    },
```

Example for BYO Database

If you want to BYO database, you must configure the `auth_info` section in Siebel CRM deployment payload.

The following is an example database section of the Siebel CRM payload for BYOD:

```
    "database": {
        "db_type": "BYOD",
        "byod": {
            "wallet_path": "/home/opc/siebel/config/wallet",
            "tns_connection_name": "phoenixx"
        },
        "auth_info": {
            "table_owner_password": " xxx",
            "table_owner_user": " xxx",
            "anonymous_user_password": "yyy",
            "siebel_admin_password": "zzz",
            "siebel_admin_username": "sadmin"
        }
    }
```

Example Kubernetes Cluster Sections for BYO-Kubernetes

Payloads for all Kubernetes cluster options.

Example payload when user chooses to go with SCM creating OKE during environment provisioning:

```
    {
        "infrastructure": {
            "kubernetes": {
                "kubernetes_type": "OKE",
                "oke": {
                    "oke_node_count": 3,
                    "oke_node_shape": "VM.Standard.E3.Flex",
                    "oke_node_shape_config": {
                        "memory_in_gbs": "60",
                        "ocpus": "4"
                    }
                }
            }
        }
```

```
}
```

Example payload when user chooses to use their cluster for environment provisioning and Kubernetes type is BYO_OKE:

```
{
     "infrastructure": {
          "kubernetes": {
          "kubernetes_type": "BYO_OKE",
          "byo_oke": {
               "oke_cluster_id": "ocid1.****",
               "oke_endpoint": "PRIVATE",
               "oke_kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml"
          },
          "ingress_controller": {
               "ingress_service_type": "LoadBalancer",
               "ingress_controller_service_annotations": {
                    "oci.oraclecloud.com/load-balancer-type": "lb",
                    "service.beta.kubernetes.io/oci-load-balancer-internal": "false",
                    "service.beta.kubernetes.io/oci-load-balancer-shape": "flexible",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-min": "10",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-max": "100",
                    "service.beta.kubernetes.io/oci-load-balancer-ssl-ports": "443",
                    "service.beta.kubernetes.io/oci-load-balancer-tls-secret": "lb-tls-certificate",
                    "service.beta.kubernetes.io/oci-load-balancer-subnet1":
 "ocid1.subnet.oc1.iad.aaaaaaaayt53nlge54fhrhvrnvyvvgqvtenngwz4tqljvpn2chn7ws4chm6q"
               }
          }
     }
}
}
```

Example payload when user chooses to use their cluster for environment provisioning and kubernetes type is BYO_OCNE:

```
{
     "infrastructure": {
          "kubernetes": {
               "kubernetes_type": "BYO_OCNE",
               "byo_ocne": {
                    "kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml"
               }
          },
          "ingress_controller": {
               "ingress_service_type": "LoadBalancer",
               "ingress_controller_service_annotations": {
                    "oci.oraclecloud.com/load-balancer-type": "lb",
                    "service.beta.kubernetes.io/oci-load-balancer-internal": "false",
                    "service.beta.kubernetes.io/oci-load-balancer-shape": "flexible",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-min": "10",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-max": "100",
                    "service.beta.kubernetes.io/oci-load-balancer-ssl-ports": "443",
                    "service.beta.kubernetes.io/oci-load-balancer-tls-secret": "lb-tls-certificate",
                    "service.beta.kubernetes.io/oci-load-balancer-subnet1":
 "ocid1.subnet.oc1.iad.aaaaaaaayt53nlge54fhrhvrnvyvvgqvtenngwz4tqljvpn2chn7ws4chm6q"
               }
          }
     }
}
```

Example payload when user chooses to use their cluster for environment provisioning and kubernetes type is OCNE, observability is enabled and local-storage is used for Prometheus and oracle-opensearch:

```
{
     "infrastructure": {
          "kubernetes": {
```

```
                "kubernetes_type": "BYO_OCNE",
                "byo_ocne": {
                     "kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml"
                }
           },
           "ingress_controller": {
                "ingress_service_type": "LoadBalancer",
                "ingress_controller_service_annotations": {
                     "oci.oraclecloud.com/load-balancer-type": "lb",
                     "service.beta.kubernetes.io/oci-load-balancer-internal": "false",
                     "service.beta.kubernetes.io/oci-load-balancer-shape": "flexible",
                     "service.beta.kubernetes.io/oci-load-balancer-shape-flex-min": "10",
                     "service.beta.kubernetes.io/oci-load-balancer-shape-flex-max": "100",
                     "service.beta.kubernetes.io/oci-load-balancer-ssl-ports": "443",
                     "service.beta.kubernetes.io/oci-load-balancer-tls-secret": "lb-tls-certificate",
                     "service.beta.kubernetes.io/oci-load-balancer-subnet1":
 "ocid1.subnet.oc1.iad.aaaaaaaayt53nlge54fhrhvrnvyvvgqvtenngwz4tqljvpn2chn7ws4chm6q"
                }
           }
      }
 },
 "observability": {
      "siebel_monitoring": true,
      "oci_config": {
           "oci_config_path": "/home/opc/config/config1",
           "oci_private_api_key_path": "/home/opc/config/oci_api_key.pem",
           "oci_config_profile_name": "DEFAULT"
      },
      "prometheus": {
           "storage_class_name": "local-storage",
           "local_storage_info": {
                "local_storage": "/mnt/test",
                "kubernetes_node_hostname": "olcne-worknode-1"
           }
      },
      "oracle_opensearch": {
           "storage_class_name": "local-storage",
           "local_storage_info": [
           {
                "local_storage": "/mnt/test1",
                "kubernetes_node_hostname": "olcne-worknode-2"
           },
           {
                "local_storage": "/mnt/test2",
                "kubernetes_node_hostname": "olcne-worknode-2"
           },
           {
                "local_storage": "/mnt/test3",
                "kubernetes_node_hostname": "olcne-worknode-2"
           }
           ]
      },
      "monitoring_mt_export_path": {
           "mount_target_private_ip": "10.0.1.168",
           "export_path": "/olcne-migration"
      }
 }
 }
```

Example Git Section for BYO-Git

If you want to BYO Git, you must configure the `byo_git` section in Siebel CRM deployment payload. For more information on Git repositories, see *Git Repositories for Siebel CRM Deployment*.

The following is an example of the git section of the Siebel CRM deployment payload when the `git_type` parameter is set to `byo_git`:

Example payload when the Git protocol type is set to `http`:

```
{
     "infrastructure": {
          "git": {
               "git_type": "byo_git",
               "byo_git": {
                    "git_protocol_type": "http",
                    "git_scm_repo_url":"https://xxxx.xxxxx.com/xxx/test1/repositories/test1-cm",
                    "git_scm_repo_branch": "main2",
                    "git_scm_flux_folder": "Siebel-crm4",
                    "git_helm_repo_url": "https://xxxx.xxxxx.com/xxx/test1/repositories/test1-cmhc",
                    "git_helm_repo_branch": "main1",
                    "git_accesstoken": "***********",
                    "git_user": "xxxx.xx@xx.com"
               }
          }
     }
}
```

Example payload when the Git protocol type is set to `ssh`:

```
{
     "infrastructure": {
          "git": {
               "git_type": "byo_git",
               "byo_git": {
                    "git_protocol_type": "ssh",
                    "git_ssh_host_key_sha256": {
                         "devops.scmservice.xxxxxx.oci.oraclecloud.com": "SHA256:m2uQAR8WI+o
+EffY/1GR6XXXXXXXX"
                    },
                    "git_scm_repo_url":"ssh://xxxx.xxxxx.com/xxx/test1/repositories/test1-cm",
                    "git_scm_repo_branch": "main2",
                    "git_scm_flux_folder": "Siebel-crm4",
                    "git_helm_repo_url": "ssh://xxxx.xxxxx.com/xxx/test1/repositories/test1-hc",
                    "git_helm_repo_branch": "main1",
                    "git_ssh_private_key": "/home/opc/siebel/oci_api_key.pem",
                    "git_user": " xxxx.xx@xx.com"
               }
          }
     }
}
```

Example payload when using existing configuration with `byo_git`:

```
"name": "demo1",
"config_id": "OOOAA",
"infrastructure": {
     "git": {
          "git_type": "byo_git",
          "byo_git": {
               "git_protocol_type": "ssh",
               "git_ssh_host_key_sha256": {
                    "devops.scmservice.xxxxxx.oci.oraclecloud.com": "SHA256:m2uQAR8WI+o+EffY/1GR6XXXXXXXX"
               },
               "git_scm_repo_url": "ssh://xxxx.xxxxx.com/xxx/test1/repositories/test1-cm",
               "git_scm_repo_branch": "main2",
               "git_scm_flux_folder": "siebel-crm4",
               "git_helm_repo_url": "ssh:// xxxx.xxxxx.com/xxx/test1/repositories/test1-hc ",
               "git_helm_repo_branch": "main1",
               "git_ssh_private_key": "/home/opc/siebel/oci_api_key.pem",
```

```
                "git_user": "xxxx.xx@xx.com"
            }
        },
    }
```

Note: You must use the same Git type for environment provisioning and configuration. For example, if you have set the `git_type` parameter to `byo_git` in the configuration payload, then you must set the `git_type` parameter to `byo_git` in the provisioning payload also. If the `git_type` specified in the configuration payload is different from the environment provisioning payload, an error is thrown during the provisioning payload validation.

Example for BYO Namespace

If you want to bring your own namespace, you must set the `byo_ns` parameter to `true` under the `infrastructure > kubernetes > byo_oke` or `byo_ocne` or `byo_others` section in Siebel CRM deployment payload. For more information on BYO namespace, see *Using a BYO Namespace*.

The following examples show snippets of the kubernetes section of the Siebel CRM deployment payload when the `byo_ns` parameter is set to `true`:

Example payload for BYO namespace with `byo_oke`:

```
"name": "<byo-namespace>"
    "infrastructure": {
    "kubernetes": {
        "kubernetes_type": "BYO_OKE",
            "byo_oke": {
            "oke_kubeconfig_path": "/home/opc/.kube/config",
            "byo_ns": true
        }
    }
}
```

Example payload for BYO namespace with `byo_other`:

```
"name": "<byo-namespace>"
    "infrastructure": {
    "kubernetes": {
        "kubernetes_type": "BYO_OTHER",
            "byo_other": {
            "kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml",
            "byo_ns": true
        }
    }
}
```

Note: When using an existing namespace (that is, when `byo_ns` is set to `true`), you must ensure that the existing namespace name matches the `name` field in the configuration or deployment payload.

Example for Configuring a Standalone Siebel Gateway

By default, when deploying Siebel CRM using SCM, a three-node Siebel Gateway cluster is created. However, if you want to create a single gateway node without any cluster features, you must set the `gateway_deployment_type` parameter to `standalone` under the `siebel` section in Siebel CRM deployment payload.

The following example shows a snippet of the `siebel` section of the Siebel CRM deployment payload when the `gateway_deployment_type` parameter is set to `standalone`:

Example payload for configuring standalone Siebel Gateway:

```
{
     "name": "gatewayX",
     "siebel": {
          "registry_url": "xxx.ocir.io",
          "registry_user": "siebeldev/xyz.xx@oracle.com",
          "registry_password": "xxxxx",
          "database_type": "Sample",
          "industry": "Financial Services",
          "gateway_deployment_type": "standalone"
     }
}
```

Example for Using Resource Tags

The following examples shows how to configure the `resource_tags` section under the `siebel` and `observability` sections:

Example payload for configuring `resource_tags` for OCI resources under:

- The `siebel` section:

```
{
   "siebel": {
      "resource_tags": {
         "defined_tags": {
            "Operations": {
               "CostCenter": "12345"
               }
            },
            "freeform_tags": {
            "owner": "scm",
            "app": "siebel"
         }
      }
   }
}
```

- The `observability` section:

```
{
   "observability": {
      "resource_tags": {
         "defined_tags": {
            "Operations": {
               "CostCenter": "12345"
               }
            },
            "freeform_tags": {
            "owner": "scm",
            "app": "siebel_observability"
         }
      }
   }
}
```

> **Note:** Do not configure `defined_tags` or `freeform_tags` directly under `siebel` or `observability`. You must always place them under `resource_tags`.

Example Payload for using `resource_tags` when enabling observability after deployment:

```
{
   "observability": {
      "siebel_monitoring": true,
      "monitoring_mt_export_path": {
         "mount_target_private_ip": "10.0.255.102",
```

```
            "export_path": "/anarnoldauto_scm"
         },
         "resource_tags": {
            "defined_tags": {
               "Operations": {
                  "CostCenter": "12345"
               }
            },
            "freeform_tags": {
               "owner": "scm",
               "app": "siebel_test"
            }
         }
      }
   }
```

Note:

- SCM applies tags only to supported OCI resources that it creates.
- SCM does not automatically retag existing OCI resources.
- SCM supports `observability.resource_tags` only for OCI deployments.
- SCM rejects non-empty `observability.resource_tags` values for OCNE, OpenShift, and other BYO non-OCI deployments during payload validation.

For more information on resource tags, see *Using Resource Tags*.

Example for Using Customer-Managed KMS Keys for Infrastructure Resources Created by SCM

The following example shows how to configure the `siebel_storage_kms_key_ocid` parameter and `siebel_storage_kms_key_dg_defined_tag` section under the `infrastructure` section:

```
{
   "infrastructure": {
      "siebel_storage_kms_key_ocid": "ocid1.key.oc1.<region>.<unique-id>",
      "siebel_storage_kms_key_dg_defined_tag": {
         "namespace": "<tag-namespace>",
         "key": "<tag-key>",
         "value": "<tag-value>"
      }
   }
}
```

Note:

- Include `siebel_storage_kms_key_ocid` and `siebel_storage_kms_key_dg_defined_tag` together only when you want to use customer-managed key for encrypting SCM-created Siebel file systems.
- Make sure the key, defined tag, dynamic group, and key-use policy already exist before you provision the environment.
- Do not provide these fields when you use BYO Siebel file systems through `mounttarget_exports`, or when you skip the Siebel Resource Manager stack.

The following example shows the `infrastructure` section when you use a customer-managed key for an SCM-created OKE cluster:

```
{
   "infrastructure": {
      "kubernetes": {
         "kubernetes_type": "OKE",
         "oke": {
            "oke_node_count": 3,
            "oke_node_shape": "VM.Standard.E3.Flex",
            "oke_node_shape_config": {
               "memory_in_gbs": "60",
               "ocpus": "4"
            },
            "kms_key_id": "ocid1.key.oc1.<region>.<unique-id>",
            "kms_key_dg_defined_tag": {
               "namespace": "<tag-namespace>",
               "key": "<tag-key>",
               "value": "<tag-value>"
            }
         }
      }
   }
}
```

Note:

- Include `kms_key_id` and `kms_key_dg_defined_tag` together only when you want to use customer-managed encryption.
- Make sure that the key, defined tag, dynamic group, and key-use policy already exist before you provision the environment.

Using a Customer-Managed KMS Key for an SCM-Created Autonomous Database

You can use customer-managed OCI KMS key to encrypt SCM created Autonomous Database for your Siebel CRM environment.

Before you change the database encryption key, make sure that you have the following information:

- ATP OCID
- ATP compartment
- Vault OCID
- Master encryption key OCID

The database must be in the **AVAILABLE** lifecycle state.

Prerequisites

Before you change the database encryption key:

1. Create or select an active OCI Vault and an enabled AES-256 master encryption key in the appropriate region. For more information, refer *Prerequisites to Use Customer-Managed Encryption Keys on Autonomous AI Database in OCI Vault*.
2. Ask a tenancy administrator or IAM administrator to create an ATP dynamic group and policy.

   The dynamic group and policy must allow the database resource principal to use the Vault and key.
3. Use the actual ATP OCID in the dynamic group rule for an existing database:

   ```
   resource.id = '<atp_OCID>'
   ```
4. Use the dynamic group name and the compartment that contains the selected Vault and key in the policy statements.

   ```
   Allow dynamic-group <atp_dynamic_group_name> to use vaults in compartment <kms_compartment>
   Allow dynamic-group <atp_dynamic_group_name> to use keys in compartment <kms_compartment>
   ```

   For more information on console steps and policy examples, refer *Create Dynamic Group and Policies for Customer Managed Keys with Vault in Same Tenancy as Database*.
5. If your tenancy uses a tag-based dynamic group convention, apply the required defined tag to the existing ATP database. You must preserve all existing tags. For more information on console and CLI instructions, refer *Adding a Tag to a Single Resource*.
6. Allow about one hour for policy access to take effect after you change a dynamic group rule or add a tag that changes dynamic group membership. For more information, refer *Updating Dynamic Groups*.

Using the Customer-Managed Key

After you complete the prerequisites, change the existing ATP database's encryption key from an Oracle-managed key to the selected customer-managed key.

For the Console procedure, refer *Use Customer-Managed Encryption Keys with Vault Located in Local Tenancy*.

For the OCI CLI alternative, refer to the Autonomous Database *configure-key* reference.

If the key-change work request fails, see *Troubleshooting Customer-Managed KMS Key Issues for an SCM-Created Autonomous Database*.

Additional Administrative Tasks in Siebel Cloud Manager

This topic provides several additional API-based administrative tasks that you might need to perform as part of using SCM to deploy and administer Siebel CRM on OCI. This topic includes the following information:

- *Resetting the Administrative Password*
- *Changing the Log Level*
- *Checking the Status of a Requested Environment*
- *Checking the Status of a Requested Configuration*
- *Resubmitting the Environment Creation Workflow*
- *Resubmitting the Environment Creation Workflow*
- *Updating Parameters During Rerun of Environment or Configuration APIs*

Resetting the Administrative Password

You can change the administrative password for SCM through the API, using a PUT request as follows:

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/credentials
Content-Type: application/json
{
"confirm":"true",
"admin_username": "admin",
"admin_password": "<your_password>"
}
Response : 200 - admin credentials are set
```

Changing the Log Level

You can set the log level for SCM application and the Ansible workflow through the API, by using a PUT request. The valid log levels are as follows. This topic is part of *Additional Administrative Tasks in Siebel Cloud Manager*.

```
CRITICAL
ERROR
WARNING
INFO
DEBUG
```

Use a PUT request like the following:

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/configure
Auth: Basic auth
Content-Type: application/json
{"log_level": "INFO"
}
Response : 200 - INFO
```

Checking the Status of a Requested Environment

To find the latest status of a requested environment, run `GET` API using the selfLink. The selfLink is displayed when you execute the payload, as shown in *Executing the Payload to Deploy Siebel CRM* . The output JSON will have a status section with the stages, such as those shown below. This topic is part of *Additional Administrative Tasks in Siebel Cloud Manager*.

```
GET https://<IP Address>:<Port>/scm/api/v1.0/environment/4QVRX5
```

Siebel CRM Application URLs

At the end of the Ansible workflow publish stage, URLs for the Siebel CRM applications are populated, similar to the following:

```
"urls": [
    "https://<IP Address>/siebel/app/callcenter/enu",
    "https://<IP Address>/siebel/app/eservice/enu",
    "https://<IP Address>/siebel/app/sservice/enu",
    "https://<IP Address>/siebel/smc"
]
```

Siebel CRM URLs can be viewed using the environment query (`GET`).

If the URLs are not populated, they can be formed by finding the IP address of the Loadbalancer (Kubernetes service or Load Balancer instance in OCI). For example:

```
https://<IP Address from LB>/siebel/app/callcenter/enu
```

To connect using `kubectl`, check the log files mentioned below.

Viewing Logs

Different kinds of logs are useful for capturing the flow and debugging. For any failures, you can review logs and correct issues. For example, you can correct policy issues in OCI and rerun the workflow.

- SCM application logs (retrieve using `GET` API). For example:

  Retrieve consolidated logs using a URL like this:

  ```
  GET https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>/logs
  ```

  Retrieve logs for specific stages using a URL like this:

  ```
  GET https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>/logs/<stage_name>
  ```

  You can obtain the `<stage_name>` from `GET` info stages section. For example, a URL like the following provides the log of the import database stage:

  ```
  GET https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>/logs/import-siebel-db
  ```

  You can also view logs by connecting to the SCM instance using `SSH` and the following command:

  ```
  sudo podman exec -it cloudmanager bash
  ```

- Ansible logs (retrieve using `GET` API). For example:

  ```
  GET https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>/logs/ansible
  ```

  You can also find Ansible logs inside the working directory of a given environment, as follows:

  - `SSH` to the SCM application instance using `opc user`.
  - Exec into the container using the command:

    ```
    - sudo podman exec -it cloudmanager bash
    ```

  - Change directory to `/home/opc/siebel/<env_id>/` (for example, `/home/opc/siebel/4QVRX5` is the working directory for an environment with environment ID 4QVRX5).
  - You can find Ansible logs inside the `artifacts` directory, such as `/home/opc/siebel/4QVRX5/artifacts`. Multiple folders might be found, such as if the workflows ran multiple times.
  - Review the `stdout` file and the `rc` file under the given run folder to see failure and debug information.

Checking the Status of a Requested Configuration

To find the latest status of a requested configuration, run `GET` API using the configuration selfLink. The output JSON will have a status section with the stages, such as those shown below. This topic is part of *Additional Administrative Tasks in Siebel Cloud Manager*.

```
GET https://<CM_Instance_IP>:<Port>/scm/api/v1.0/configuration/<config_id>
```

Related Topics

Customizing Configurations Prior to Greenfield Deployment

Resubmitting the Environment Creation Workflow

You can use a `PUT` API like the following to resubmit the environment request. This topic is part of *Additional Administrative Tasks in Siebel Cloud Manager*.

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>
```

The environment ID is a unique number and can be fetched from the selfLink of the output from the environment creation.

In order to run a specific stage due to a failure, the stage name can be passed in the URL using the `run-only-this-stage` query parameter.

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>?run-only-this-stage=import_siebel_db
```

This will execute only the stage where provided stage i.e. import_siebel_db.

In order to run a specific stage and all the subsequent stages, the stage name to begin should be passed in the `run-this-stage-and-all-following-stages` parameter.

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>?run-this-stage-and-all-following-stages=import_siebel_db
```

This will execute the provided stage that is import_siebel_db, and the stages after that.

The values for the parameters: `run-only-this-stage` and `run-this-stage-and-all-following-stages` can be fetched from the GET method API of environment/configuration.

Updating Parameters During Rerun of Environment or Configuration APIs

This section covers broadly these use case categories:

- How users can update some of the parameters during rerun of environment/configuration APIs to correct the invalid/incorrect values and resume the workflows.
- When an environment provisioning is triggered using an existing configuration, that is `config_id` is passed in payload and still one wants to use different infrastructure setup for environment keeping other configuration customizations.
- Update the environment status as completed if the environment has failed in the App URL validation stage.

You can update parameters by passing them in payload for these API methods:

- PUT /environment - rerun of existing environment
- PUT /configuration - rerun of existing configuration
- POST /environment method which uses existing configuration, that is `config_id` is passed in payload.

SCM allows to override only predefined set of parameters which are detailed in the following use case scenarios. For other parameters if overridden, validation error will be thrown. The reason for this is they are considered as immutable fields to keep the environment intact.

This section covers the following use cases:

- *Use Case 1 - Non BYO Case - When "Use existing resources" Checkbox is Not Selected*
- *Use Case 2 - BYO Case - When "Use existing resources" Checkbox is Selected*
- *Use Case 3 - BYOD Case - When "Use existing resources" Checkbox is Not Selected and Only Existing Database is Used*
- *Use Case 4 - When running a CI activity needs env_status to be changed*

Note:

- For all the following use cases, the parameters related to 'siebel' section cannot be overridden during API execution.
- Any other parameter, which is not mentioned in the respective use case, is not allowed to be overridden during API execution.
- For non-BYO use case, 'database' values cannot be overridden.

Use Case 1 - Non BYO Case - When "Use existing resources" Checkbox is Not Selected

For Non BYO Case, one can pass below parameters in payload during PUT or POST (when `config_id` is passed in payload) API execution.

- CIDR parameters (only applicable when you chose Advanced Network Configuration during SCM stack creation)
  - infrastructure > siebel_lb_subnet_cidr
  - infrastructure > siebel_private_subnet_cidr
  - infrastructure > siebel_db_subnet_cidr
  - infrastructure > siebel_cluster_subnet_cidr
- Size parameters
  - size > ses_resource_limits
  - size > sai_resource_limits
  - size > cgw_resource_limits
  - size > ses_resource_requests
  - size > sai_resource_requests
  - size > cgw_resource_requests

Here, it is not mandatory to pass all parameters. One can pass only required parameter during API execution.

Example Payload for PUT/POST method when "Use existing resources" Checkbox is Not Selected:

```
{
 "infrastructure": {
 "siebel_lb_subnet_cidr" : "10.0.1.0/24",
 "siebel_private_subnet_cidr" : "10.0.2.0/24",
 "siebel_db_subnet_cidr" : "10.0.3.0/24",
 "siebel_cluster_subnet_cidr" : "10.0.4.0/24"
 },
 "ses_resource_limits": {
 "cpu": "4",
 "memory": "24Gi"
 },
 "ses_resource_requests": {
 "cpu": "1",
 "memory": "4Gi"
 },
 "sai_resource_limits": {
 "cpu": "2",
 "memory": "4Gi"
 },
 "sai_resource_requests: {
 "cpu": "1",
 "memory": "4Gi"
 },
 "cgw_resource_limits": {
 "cpu": "2",
 "memory": "4Gi"
 },
 "cgw_resource_requests”: {
 "cpu": "1",
 "memory": "4Gi"
 }
 }
```

```
}
```

Use Case 2 - BYO Case - When "Use existing resources" Checkbox is Selected

For BYO Case, one can pass the following parameters in payload during PUT or POST (when `config_id` is passed in payload) API execution.

- OKE parameters
  - infrastructure > kubernetes > byo_oke > oke_cluster_id
  - infrastructure > kubernetes > byo_oke > oke_endpoint
  - infrastructure > kubernetes > byo_oke > oke_kubeconfig_path
  - infrastructure > mounttarget_exports
- Database parameters - If you pass database section in payload, it is mandatory to pass all of the below fields.
  - database > db_type
  - database > byod
  - database > auth_info
- Size parameters
  - size > ses_resource_limits
  - size > sai_resource_limits
  - size > cgw_resource_limits
  - size > ses_resource_requests
  - size > sai_resource_requests
  - size > cgw_resource_requests

Example Payload for PUT/POST method when "Use existing resources" Checkbox is Selected

```
{
"infrastructure": {

"kubernetes": {
"kubernetes_type": "BYO_OKE",
"byo_oke": {
"oke_cluster_id": "<cluster-ocid>",
"oke_endpoint": "PRIVATE",
"oke_kubeconfig_path": "<path-to-kubeconfig-file>"
}
}

"mounttarget_exports": {
"siebfs_mt_export_paths": [
{
"mount_target_private_ip" : "10.0.0.82","export_path": "/siebfs0"
}
]
}
},
"database": {
"db_type": "BYOD",
"byod": {
"wallet_path": "/home/opc/siebel/wallet",
"tns_connection_name": "test_tp"
```

```
},
"auth_info": {

"admin_user_name": "*****",
"admin_user_password": "********",
"anonymous_user_password": "********",
"default_user_password": "*******",
"table_owner_password": "*******",
"table_owner_user": "******"
}
},
"size": {
"ses_resource_limits": {
"cpu": "4",
"memory": "24Gi"
},
"ses_resource_requests”: {
"cpu": "1",
"memory": "4Gi"
},
"sai_resource_limits": {
"cpu": "2",
"memory": "4Gi"
},
"sai_resource_requests": { "
cpu": "1",
"memory": "4Gi"
},
"cgw_resource_limits": {
"cpu": "2",
"memory": "4Gi"
},
"cgw_resource_requests": {
"cpu": "1",
"memory": "4Gi"
}
}
}
```

Use Case 3 - BYOD Case - When "Use existing resources" Checkbox is Not Selected and Only Existing Database is Used

For BYO Case, one can pass the following parameters in payload during PUT or POST (when `config_id` is passed in payload) API execution.

- CIDR parameters (only applicable when you chose Advanced Network Configuration during SCM stack creation)
  - infrastructure > siebel_lb_subnet_cidr
  - infrastructure > siebel_private_subnet_cidr
  - infrastructure > siebel_db_subnet_cidr
  - infrastructure > siebel_cluster_subnet_cidr
- Database parameters - If you pass database section in payload, it is mandatory to pass all the following fields.
  - database > db_type
  - database > byod
  - database > auth_info

- Size parameters
    - size > ses_resource_limits
    - size > sai_resource_limits
    - size > cgw_resource_limits
    - size > ses_resource_requests
    - size > sai_resource_requests
    - size > cgw_resource_requests

Example Payload for PUT/POST method when "Use existing resources" Checkbox is not Selected and only existing Database is used:

```
{
"infrastructure": {
"siebel_lb_subnet_cidr" : "10.0.1.0/24",
"siebel_private_subnet_cidr" : "10.0.2.0/24",
"siebel_db_subnet_cidr" : "10.0.3.0/24",
"siebel_cluster_subnet_cidr" : "10.0.4.0/24"

}
},
"database": {
"db_type": "BYOD",
"byod": {
"wallet_path": "/home/opc/siebel/wallet",
"tns_connection_name": "test_tp"
},
"auth_info": {

"admin_user_name": "*****",
"admin_user_password": "********",
"anonymous_user_password": "********",
"default_user_password": "*******",
"table_owner_password": "*******",
"table_owner_user": "******"
}
},
"ses_resource_limits": {
"cpu": "4",
"memory": "24Gi"
},
"ses_resource_requests": {
"cpu": "1",
"memory": "4Gi"
},
"sai_resource_limits": {
"cpu": "2",
"memory": "4Gi"
},
"sai_resource_requests": {
"cpu": "1",
"memory": "4Gi"
},
"cgw_resource_limits": {
"cpu": "2",
"memory": "4Gi"
},
"cgw_resource_requests": {
"cpu": "1",
"memory": "4Gi"
}
```

```
 }
}
```

Use Case 4 - When running a CI activity needs env_status to be changed

To run the CI activities of the Siebel CRM deployment such as SFS cleanup etc, the environment has to be in a completed stage. The `env_status` parameter will define if the environment is completed or not. If the environment has failed in the app validation stage (because of some app urls not coming up), then this API can be used to update the environment as completed.

Example:

```
PUT https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/<env_id>

{
 "env_status": "completed"
}
```

Troubleshooting a Siebel Cloud Manager Instance or Requested Environment

This topic provides several additional tasks for reviewing and troubleshooting a SCM instance or an environment that you requested. This topic includes the following information:

- *Troubleshooting a Siebel Cloud Manager Instance*
- *Troubleshooting BYO Git Failure*
- *Examining Your Deployment*
- *Reviewing the PostInstallDBSetup Execution Status*
- *Troubleshooting Oracle Resource Manager Stack Apply Job Failure*
- *Troubleshooting Customer-Managed KMS Key Issues for an SCM-Created Autonomous Database*
- *Troubleshooting Proxy Server Settings Propagation Failure*
- *Troubleshooting Handshake Failed Server State in Siebel Management Console*
- *Troubleshooting Issues Related to Siebel Migration Application in an SCM Deployed Siebel CRM Environment*
- *Troubleshooting Issues Related to Siebel CRM Observability – Monitoring Solution*
- *Troubleshooting Issues Related to Siebel CRM Observability – Log Analytics Solution*
- *Troubleshooting Issues Related to Customer-Managed KMS Keys*
- *Troubleshooting Stalled Resources Error*

Troubleshooting a Siebel Cloud Manager Instance

The SCM instance might not run in any of the following cases:

- When the OCI configuration is not set up correctly, access to the Siebel Cloud Manager (SCM) instance is blocked, and the system returns a safe response, as shown below:

```
{
    "data": {
        "log_path": "logs/common_logger.log"
    },
```

```
    "message": "The request could not be completed. Review the referenced logs for details.",
     "status": "failed"
}
```

In the above example, the `"message"` field is intentionally general.

The `data.log_path` parameter value identifies the relative path to the SCM log file that contains detailed diagnostic information about the failure. This path is relative to the SCM home directory; it is neither an absolute operating system path nor a browser URL. Administrators can use this path together with the approximate time of the request to locate the relevant diagnostic information. It identifies the log file relevant to the specific failure. For example, it may reference the common SCM log:

```
logs/common_logger.log
```

If the failure occurs during a Siebel deployment stage and a stage-specific diagnostic log is available, `data.log_path` may instead contain the corresponding safe, relative path to that log.

For example, if the SCM home directory on the server is `/home/opc` and the response contains:

```
{
    "data": {
        "log_path": "logs/common_logger.log"
    }
}
```

The administrator can review the corresponding log file on the server at:

```
/home/opc/logs/common_logger.log
```

The complete server path is used only when accessing and reviewing the log on the host. SCM does not expose the absolute file system path in the API response; the response contains only the safe, relative path.

- When the SCM container fails to start. You can check the status of the SCM container and restart it as follows:
  a. SSH in to the SCM instance.
  b. Check the status of the SCM container:

     ```
     sudo podman ps
     ```

     If this command does not return any output, it implies that the container hasn't started. In this case, check the `cloud- init-output.log` file:

     ```
     vi /var/log/cloud-init-output.log
     ```

     If the log indicates that the container has failed or is not running, run the following command to confirm the status of the SCM container:

     ```
     sudo systemctl status cloudmanager
     ```

     If the command output displays the container status as FAILED, restart the SCM container and check its status again as follows:

     ```
     sudo systemctl restart cloudmanager
     sudo systemctl status cloudmanager
     ```

Troubleshooting BYO Git Failure

When using BYO Git, the `process_cg_artifacts` stage fails during Helm URL validation with the following error:

```
SSL certificate problem: unable to get local issuer certificate
```

By default, Git verifies SSL certificates when connecting to remote repositories over HTTPS during operations like `'git clone'` or `'git ls-remote'`. An error may occur if Git fails to locate or access the necessary local CA certificates required for SSL validation.

This issue often arises when the default CA bundle is not configured properly, or Git cannot find the expected certificate path on the local machine. It can also occur if your CA certificate is not included in the bundle. In these scenarios, Git will throw an error with the message stating that it cannot verify the SSL certificate of the remote repository.

You must configure Git to use the local certificate authority file to resolve this error. To use the local certificate authority file, set the `http.sslCAInfo` configuration in Git to the correct CA certificate path by executing the following command from the SCM container or pod:

```
git config --global http.sslCAInfo=/etc/pki/tls/certs/ca-bundle.crt
```

In the above command:

- `http.sslCAInfo` specifies the path to the file containing one or more trusted root certificates that Git uses to verify SSL connections.
- `/etc/pki/tls/certs/ca-bundle.crt` is a common location for CA certificates on Linux-based systems. This path might vary depending on the operating system and environment.

Examining Your Deployment

You can examine the Kubernetes deployment with `kubectl`, such as in the following commands run on the virtual machine for SCM. This topic is part of *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

```
sudo podman exec -it cloudmanager bash

source /home/opc/siebel/<env_id>/k8sprofile

kubectl -n <deployment name supplied in deployment POST request> get all
```

The last command shown above displays information about all the Kubernetes objects that were created for a given Siebel CRM environment when it was deployed on OCI (which might have been subsequently modified through making incremental changes). These objects or pods correspond to instances (and replicas) of Siebel Server, Siebel Gateway (CGW), Siebel Application Interface (SAI), and Siebel Management Console (SMC). Also present are an ingress controller to control ingress to proxy servers, and a Metacontroller and a Siebel Controller for executing incremental changes.

You can compare the information displayed by this `kubectl` command to the representation of the deployment and its primary elements in SMC.

Reviewing the PostInstallDBSetup Execution Status

A PostInstallDBSetup job is run as part of the deployment pipeline. It is run as a Kubernetes job. The failure of this job does not stop the deployment and the application execution. However, you are advised to check the logs to confirm that PostInstallDBSetup ran successfully. In case of failure, take appropriate corrective action, as described in *Siebel Database Upgrade Guide* on *Siebel Bookshelf*. This topic is part of *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

Before checking the PostInstallDBSetup execution, first set up the `kubectl` CLI. Connect to the SCM container and set the profile for your environment, as follows:

```
sudo podman exec -it cloudmanager bash

source /home/opc/siebel/<env_id>/k8sprofile
```

Next, run `kubectl` commands to find the PostInstallDBSetup job pod for which you will run the `logs` command. First run a command like this:

```
kubectl -n <env_name> get pods | grep postinstall
```

Example output:

```
postinstalldb-q2wnv                  0/1       Completed    0              92m
```

And then run a command like this:

```
kubectl -n demo3 logs po/postinstalldb-q2wnv
```

Example output:

```
+ /siebel/mde/siebsrvr/bin/PostInstallDBSetup -i /config/PostInstallDBSetup.ini -p
 S1e8eladm1n123 -z SiebelAdmin123 'PostInstallDBSetup' database final configuration is not required on this
 instance as it has already been executed in a prior install.

real 0m21.438s

user 0m2.504s

sys 0m0.952s

Exit Status : 8
```

Detailed logs for PostInstallDBSetup can be verified from the persistent folder stored in the file storage service (OCI service). To access this location from the SCM instance, use a command like the following in the same shell in which you connected to the SCM container:

```
cd /home/opc/siebel/<env_id>/siebfs0/<ENV_NAME_IN_CAPS>/SES/POSTINSTALLDB/siebsrvr/log
```

Troubleshooting Oracle Resource Manager Stack Apply Job Failure

Sometimes OCI resource creation fails with errors. These errors can be found in the apply job logs, which you can access using the OCI console. During such failures, you can trigger the Ansible workflow again by using a `PUT` rerun command that resubmits the environment creation workflow. The errors might resemble the following. This topic is part of *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

```
Error: 400-InvalidParameter

Provider version: 4.20.0, released on 2021-03-31. This provider is 36 updates behind to current.
```

```
Service: FileStorageFileSystem

Error Message: Ocid 'ocid1.compartment.oc1..aaaaaaaabwvdshyuwbyfpx72m4lq6yni673m2ewf7qrou7ha5dvaxrjeogfa'
 not found in Compartment Tree!

OPC request ID:
 a42cd5b1927359f403a56e8eabb378b8/47793109B015FB5F54CE70BC905ACF70/968A739323FC3A76967AD9E94862A1E2

Suggestion: Please update the parameter(s) in the Terraform config as per error message Ocid
 'ocid1.compartment.oc1..aaaaaaaabwvdshyuwbyfpx72m4lq6yni673m2ewf7qrou7ha5dvaxrjeogfa' not found in
 Compartment Tree!

on modules/storage/main.tf line 1, in resource "oci_file_storage_file_system" "siebelCM_Fss"

1: resource "oci_file_storage_file_system" "siebelCM_Fss" {
```

For example, use a command like this to resubmit the environment creation workflow:

```
https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/4QVRX5
```

Troubleshooting Customer-Managed KMS Key Issues for an SCM-Created Autonomous Database

If the work request to change the encryption key for an SCM-created Autonomous Database fails, use the following steps to troubleshoot customer-managed KMS key issues:

1. Review the work request error. Record the ATP OCID, work request OCID, timestamp, selected key, and selected Vault.
2. Verify that the ATP database is in the `AVAILABLE` lifecycle state.
3. Verify that the Vault is active.
4. Verify that the master encryption key is enabled in the expected region and compartment.
5. Verify that the dynamic group rule identifies the existing ATP database.

   If the rule uses a defined tag, verify that the tag uses the correct namespace, key, and value.
6. Verify that the active policy grants the dynamic group use vaults and use keys permissions in the compartment that contains the selected Vault and key.
7. Allow the documented propagation time to complete if a dynamic group rule or matching tag was changed.
8. Retry the key-change operation on the same ATP database. You don't need to create another database or rerun the completed Siebel CRM deployment only to retry the KMS operation.

Note: You must keep the selected customer-managed key enabled after the database is configured to use it. If the key is disabled or deleted, the database can become inaccessible. Manage the key according to your organization's key management policy.

Troubleshooting Proxy Server Settings Propagation Failure

When provisioning SCM through OCI Marketplace, the proxy server settings sometimes fail to propagate to the Docker daemon. As a result, the connection to the container registry, object store bucket, and so on fails. Because the connection with the container registry wasn't established, the SCM container doesn't start. You can confirm the issue in the `/var/log/messages` log file.

The following is a sample from the log file that confirms the connectivity issue with the container registry:

```
cloud-init: Starting cloud manager container with CM_25.1.0 tag
cloud-init: Trying to pull repository phx.ocir.io/xxxxxxxxx/cm/server ...
dockerd: time="2025-02-28T10:24:39.915338674Z" level=info msg="Attempting next endpoint for pull after
 error: Get
\"https://phx.ocir.io/v2/xxxxxxxxx/cm/server/manifests/CM_25.1.0\": Get
\"https://phx.ocir.io/20180419/docker/token?scope=repository%xxxxxxxxx%2Fcm%2Fserver
%3Apull&service=phx.ocir.io\":
net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)"
cloud-init: Get
```

You must manually configure the proxy settings on the VM and start the SCM container to resolve this issue, as follows:

1. SSH into the SCM instance.
2. Create the `docker.service.d` directory as follows:
   ```
   mkdir -p /etc/systemd/system/docker.service.d
   ```
3. Create the `/etc/systemd/system/docker.service.d/http-proxy.conf` file with proxy configuration as follows:
   ```
   [Service]
   Environment="HTTP_PROXY=http://www-proxy-xxxx.xx.xxxxx.com:80"
   Environment="HTTPS_PROXY=http://www-proxy-xxxx.xx.xxxx.com:80"
   Environment="NO_PROXY=localhost,127.0.0.1,.internal,xxx.com,us.xxxxx.com,idc.xxxx.com,in.xxxx.com"
   ```
4. Reload the daemon and restart the docker service as follows:
   ```
   sudo systemctl daemon-reload
   sudo systemctl restart docker
   ```
5. Verify that the proxy configuration was successful by executing the following command:
   ```
   sudo systemctl show --property=Environment docker
   ```

   This command shows all environment variables configured for the Docker service. Search and verify that the proxy settings are correct.
6. Start the SCM container:
   ```
   cd /home/opc/cmapp/<CM_RESOUCE_PREFIX>
   bash start_cmserver.sh <SCM version>
   ```

Troubleshooting Handshake Failed Server State in Siebel Management Console

If you see a `Handshake failed` server state in the Management screen in Siebel Management Console (SMC), then you cannot perform management runtime activities from SMC. This topic is part of *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.

After all lift and shift activities are completed for a Siebel CRM deployment on OCI, and you navigate to the Servers view in the Management screen in SMC, you might see a State value of `Handshake failed` for one or more servers. This happens when the Siebel Gateway (CGW) instantiates the Server Manager sessions before all servers are in the running state.

To correct this state and restore the SMC management functionality, you must restart all of the siebelcgw pods. To do this, first connect to the SCM container and set the profile for your OCI environment, as follows:

```
sudo podman exec -it cloudmanager bash

source /home/opc/siebel/<env_id>/k8sprofile
```

Next, restart all of the CGW pods using the following command:

```
kubectl rollout restart sts siebelcgw -n <env_name>
```

Now you can log in to SMC again and verify the server states from the Management screen.

Troubleshooting Issues Related to Siebel Migration Application in an SCM Deployed Siebel CRM Environment

A few points to watch out for:

- Invalid "Object Manager" value under REST Inbound Defaults in Application Interface Profile. In such case, you can edit AI profile and update the REST Inbound Defaults Object Manager to valid Object Manager.
- Migration application requires components such as EAIObjMgr, WfProcMgr, WfProcBatchMgr to be online. If any of them are not enabled, you might face issue in the workings of migration app. This is also informed in "warning" section of the environment self link response. Make sure to include these components before using the migration application. You can also add these components incrementally using the steps given in this document.
- Missing Database privileges required for migration application.

Troubleshooting Issues Related to Siebel CRM Observability – Monitoring Solution

- To view and query various metrics, use Prometheus UI (https://<IP>/prometheus)
  Reference: *https://prometheus.io/docs/prometheus/latest/querying/basics/*
- When you define custom alert rules:
  - Verify if alerts are registered using Prometheus UI
  - If alert is not appearing in Prometheus, then check the Alertmanager pod logs to debug and identify the issues using command:
    ```
    kubectl logs siebel-alertmanager-<id> -n <namespace>
    ```
- When you customize Prometheus configuration and if it is not working, you may debug using this command:
  ```
  prometheus logs - kubectl logs prometheus-deployment-<id> -n <namespace>
  ```
- When you use Custom Siebel Metrics feature, if the metric did not propagate to Prometheus, you may view the logs of siebel-metric-exporter pod to identify the issue using this command:
  ```
  kubectl exec -it siebel-metric-exporter-<id> bash -n <namespace> Log Location - /src/
  siebel_metric_exporter.log
  ```

Troubleshooting Issues Related to Siebel CRM Observability – Log Analytics Solution

For troubleshooting cases where logs are not getting collected or streamed appropriately, a closer look at the log collector and log aggregator behavior will be helpful.

Log Collector:

- Check the status of the application (Siebel Server, for example).
- Check that the log collector, which is a sidecar implementation on Fluentd, is available.
- Check the logs of the log collector pods to identify application log files getting streamed.

Log Aggregator:

- Ingestion can be verified in the log aggregator streams.
- The logs of the log aggregator are available in the pod logs and vary based on the enabled output modules.
- The log traces generated by OCI plugin are distinctly different from those generated for OpenSearch.
- When both OCI Logging Analytics and OpenSearch are enabled, their logs will be mixed in log trace.

Troubleshooting Issues Related to Customer-Managed KMS Keys

Issue	Possible Cause	What You Can Do
SCM stack fails while creating SCM file storage with a KMS key.	The defined tag, dynamic group, File Storage service policy, or KMS key-use policy is missing or incorrect.	Verify the KMS key OCID, tag namespace, tag key, tag value, dynamic group rule, File Storage service principal, and policy propagation.
SCM boot volume encryption fails.	The Block Volume service cannot use the KMS key.	Verify the `blockstorage` service policy for the SCM file storage KMS key.
Siebel environment stack rejects the payload.	`siebel_storage_kms_key_ocid` and `siebel_storage_kms_key_dg_defined_tag` were not provided together, or they were provided for BYO FSS.	Provide both fields together only for SCM-created Siebel File Systems or omit both fields to use Oracle-managed encryption.
BYO file systems are not encrypted with the supplied key.	SCM does not create or re-encrypt BYO file systems.	Configure encryption on the BYO file systems outside SCM.
The OKE work request reports that the cluster resource principal cannot use the KMS key.	The OKE cluster is not matched by the dynamic group, or the dynamic group does not have key-use permission.	Verify the KMS key OCID, defined tag namespace, defined tag key, defined tag value, dynamic group rule, and IAM policy. Allow time for IAM policy propagation.
SCM logs warnings and proceeds but may fail later. For example, KMS prerequisite preflight could not verify defined tag, dynamic group, or policy prerequisites.	The SCM OCI profile cannot inspect tenancy-level prerequisites.	Ask the OCI administrator to verify the KMS key, defined tag, dynamic group, and IAM policy configuration.

Troubleshooting Stalled Resources Error

A stalled resources error occurs when a resource does not become ready within the configured deployment interval.

A resource can stall for the following reasons:

- An image pull fails.
- The cluster has insufficient resources.
- The resource configuration is invalid.

- A required dependency is unavailable.
- A volume mount fails.
- Network connectivity fails.
- The application fails to start.

When a Helm release fails, Helm remediation can roll back the release. Depending on the remediation and cleanup configuration, the failed Pod might no longer be available. This can make it difficult to collect logs and identify the root cause. You must re-create the resource to collect the required logs and identify the cause.

Re-creating the Resource for Debugging

If Helm rollback removes the failed Pod, create a new chart revision to start another deployment attempt:

1. Increment the chart version in the affected chart's `Chart.yaml` file:
   ```
   /home/opc/siebel/<ENV_ID>/<git_helm_repo>/<chart_name>/Chart.yaml
   ```
2. Commit and push the chart update:
   ```
   git add <chart_directory>
   git commit -m "Trigger stalled resource deployment for debugging"
   git push
   ```
3. Reconcile the Git source and the affected HelmRelease:
   ```
   flux reconcile source git <source_name> -n <namespace>
   flux reconcile helmrelease <helmrelease_name> -n <namespace> --with-source
   ```
4. Verify that Flux creates or upgrades the Helm release and that Kubernetes creates the required pods:
   ```
   flux get helmrelease <helmrelease_name> -n <namespace>
   kubectl get pods -n <namespace>
   ```

Monitoring the Deployment and Collecting Logs

You must monitor the deployment while it is in progress. Before Helm remediation rolls back the release, collect events, resource descriptions, and container logs, as follows:

```
kubectl get events -n <namespace> -w
kubectl describe <resource_type> <resource_name> -n <namespace>
kubectl describe pod <pod_name> -n <namespace>
kubectl logs <pod_name> -n <namespace> --all-containers=true
```

Review the events, resource descriptions, and container logs to identify the cause of the stalled resource.

Updating Siebel Cloud Manager with a New Container Image

You can update SCM with a new container image whenever one becomes available, such as when an updated version of SCM has become available.

The procedure listed in this section is applicable either when:

- You are upgrading from a version lesser than 25.4 to a new container image version lesser than 25.4 or
- You are upgrading from SCM 25.4 to a new container image later than SCM 25.4

Note: SCM 25.4 or later works on Oracle Linux 8, so you must first upgrade Oracle Linux on the VM running SCM to Oracle Linux 8. For more information, see *Upgrading Oracle Linux on the SCM VM*.

Note: If you are updating SCM from a version earlier than 26.6 to SCM 26.6 or later, on a VM that runs Podman, you must update the SCM startup script before you update the SCM container image as follows:

1. Open `/home/opc/cmapp/<CM_NAMESPACE>/start_cmserver.sh` in edit mode.
2. Add the following line immediately after the image tag replacement command and before the `systemctl daemon-reload` command:
   ```
   sudo find $SYSTEMD_DIR/cloudmanager.container -type f -exec sed -i "s|^User=opc$|User=1000|" {} \;
   ```
3. Update the ownership of the Artifactory token file, if the file exists, so that the SCM 26.6 or later container process can read it:
   ```
   sudo chown opc:opc /home/opc/cmapp/<CM_NAMESPACE>/config/artifactory_token
   ```

To update SCM with a new container image:

Note: If your SCM VM instance is running Oracle Linux 7 with Docker as the container management tool, then execute Docker equivalents of the Podman commands in the steps below.

1. SSH into the SCM virtual machine instance.
2. Run the following command to find the current SCM container version which is active and note the image tag, that is, the current SCM version.
   ```
   sudo podman ps
   ```
3. Get the latest SCM application version from Oracle Marketplace (for example, CM_23.1.0).
4. Run the shell script for starting the SCM server with the latest application version as the input, as in these examples:
   ```
   cd /home/opc
   bash start_cmserver.sh CM_23.1.0
   ```
5. Verify the startup of the SCM application using the following command:
   ```
   sudo podman ps
   ```
6. Check the version of the running container in the image tag. It should match the input provided during the start shell.
7. After updating SCM with the new version, run the following command to get the new SCM features:
   ```
   sudo podman exec -it cloudmanager bash
   cd /home/opc
   bash siebel-cloud-manager/scripts/cmapp/migration.sh
   ```
   Run all applicable options in the `migration.sh` script in the order presented, based on your source version.
   Note: You might see GitLab merge conflict errors during migration. In such cases, fix the conflicts manually and try again.
8. File systems are also mounted in the SCM instance. Once the container is restarted, the existing mounts will be disconnected. In order to mount again, execute the following command:

```
sudo mount -t nfs {FILESYSTEM_HOST}:/{env-namespace}-siebfs{filesystem-index} /home/opc/siebel/{env_id}/
{env-namespace}-siebfs{filesystem-index} -o nolock
```

All the information needed for the above command is available in the environment yaml file or the GET response of the corresponding environment.

9. After executing these commands, exit from the container.
10. Run the following commands to restart the container:
    ```
    cd /home/opc/cm_app/{CM_RESOURCE_PREFIX}/bash start_cmserver.sh <SCM_VERSION>
    ```
11. Confirm that the latest SCM version is up and running
    ```
    sudo podman ps
    ```

Updating Ingress Controller

Starting with SCM 26.3, new Siebel CRM deployments use Traefik as the default ingress controller. New environments get a supported, modern ingress setup automatically—no customer action required. But if you are using SCM version earlier than 26.3, update to SCM 26.3 and run the migration workflow in your environment. After migration, keep NGINX temporarily to validate and complete a controlled switchover. When validation is complete, remove the NGINX entry from your GitOps-managed configuration.

If you're on an SCM release earlier than 26.3, follow these steps to update the ingress controller:

1. Update SCM to 26.3. For more information, see *Updating Siebel Cloud Manager with a New Container Image*.
2. Run the SCM migration workflow to migrate the ingress controller from NGINX to Traefik:
    a. Log in to the SCM container.
    b. Go to the `cmapp` directory:
    ```
    cd /home/opc/siebel-cloud-manager/scripts/cmapp/
    ```
    c. Run the migration script:
    ```
    bash migration.sh
    ```
    d. Select option 8 (Traefik migration) in the migration menu.
    e. Enter the `env_id` of the environment you want to migrate.

After the migration, SCM keeps the NGINX ingress Helm charts and releases, so you have time to validate Traefik and complete a controlled switchover. You'll see a message similar to this:

"NGINX Ingress Helm charts and releases are retained post-migration to Traefik to allow time for switchover and validation. After transition is complete, you must manually remove the Nginx entry from the `/home/opc/siebel/<env_id>/<scm_repo>/<flux_folder>/infrastructure/kustomization.yaml` file and commit the changes to the repository."

After you confirm that Traefik is handling ingress as expected:

1. Remove the NGINX entry from:
    ```
    /home/opc/siebel/<env_id>/<scm_repo>/<flux_folder>/infrastructure/kustomization.yaml
    ```
2. Commit and push the change (GitOps-managed deployments).

From security standpoint, from SCM 26.3 onwards, TLS termination takes place at Traefik. That means, certificates and TLS configuration are managed at the Traefik ingress layer, and any HTTPS redirect logic must be implemented using Traefik constructs rather than relying on legacy NGINX annotations. For instructions about certificate management configurations, refer to relevant sections in this document.

Migrating (Lift-And-Shift) Existing Siebel CRM Deployments

The Siebel Lift utility is a command-line utility, developed in Python that is used for migrating an existing Siebel CRM deploymentto the cloud or any CNCF certified Kubernetes cluster. The Siebel Lift utility performs the following tasks:

- Creates deployment artifacts from your existing on-premises Siebel CRM deployment and stores them in a staginglocation.
- Reads the stored artifacts and uploads them to OCI Object Storage to populate the migration pipeline for your SiebelCRM deployment on OCI.

TheSiebel Lift utility is available on MOS as Siebel CRM 26.8.0.0 Lift Utility (Patch).

Greenfield deployments of SiebelCRM don't use the Siebel Lift utility.

For the procedure to migrate an existing Siebel CRM deployment, see the *Migrating Siebel CRM Deployments Using Siebel Lift Utility* chapter.

Removing a Siebel CRM Deployment on OCI

Where you need to remove (destroy) an existing deployment and its dependent components, including its associated registry and Git repository, you can do so by using the DELETE API method and specifying the unique environment ID of this deployment. Details for this API follow.

Method: DELETE

URL: `<CM_Instance_IP>/scm/api/v1.0/environment/<env_id>`

Response

```
Client Validation Failed(400):
{
        "description": "Invalid Environment"
    }
Success(200):
{
        "delete_job_id": ocid | null,
        "is_cm_project_removed": bool | null,
        "is_delete_request_made": bool | null,
        "is_dir_archived": bool | null,
        "is_helm_project_removed": bool | null,
        "is_ingress_removed": bool | null,
        "is_registry_removed": bool | null,
        "is_bucket_deleted": bool | null
    }
```

Note: If the `byo_ns` field is set to `true` in the Siebel CRM deployment payload, the DELETE API will not remove or clean up the namespace.

Keys in Response Definition

The following keys are part of the response definition:

- **delete_job_id**: The OCI resource manager delete job OCID, which can used to track the status of the deletion job.
- **is_cm_project_removed**: Indicates whether the SCM project in the Git instance was removed.
- **is_delete_request_made**: Indicates whether the DELETE request was made to OCI.
- **is_dir_archived**: Indicates whether the environment was moved to the `archive` directory.
- **is_helm_project_removed**: Indicates whether the Helm Charts project in the Git instance was removed.
- **is_ingress_removed**: Indicates whether the Load Balancer was removed.
- **is_registry_removed**: Indicates whether the Registry was removed.
- **is_bucket_deleted**: Indicates whether the object storage bucket was removed.

> **Note:** A Siebel CRM environment on OCI is created in over a dozen stages. If there were issues in any of these stages, then the environment provision can fail. (See also *Troubleshooting a Siebel Cloud Manager Instance or Requested Environment*.) For example, resource creation might fail if service limits were undefined in the user account. In the response from using the DELETE method to clean up such a failed environment, keys that have null values instead of Boolean values represent stages that did apply in this case: where the stage was not applicable due to an issue in a prior stage of creation. In the above example, the environment failed in the resource creation, and so no Load Balancer ingress would have been configured. So, in this case, the response contains the null value for the key `is_ingress_removed`.

Making Incremental Changes to Your Siebel CRM Deployment on OCI

This topic describes how to make incremental changes to your existing Siebel CRM deployment on OCI. This topic includes the following information:

- *Making Incremental Changes*
- *How Incremental Changes Are Processed*
- *Templates for Different Runtime Entities*
- *Use Cases for Making Incremental Changes*

> **Note:** If you are using greenfield deployment use case 2, described in *Customizing Configurations Prior to Greenfield Deployment*, your configuration customizations prior to deployment can include any of the types of changes described in this topic. Examples include adding or deleting components on a server, adding new profiles, or adding or deleting parameters for an enterprise, server, or component. After deployment, any configuration changes must be made in the deployed environment, as described in this topic. If you require the same changes in the original customized configuration that you created in greenfield configuration use case 2, then you must make the same changes in both locations.

- SSH into the SCM instance

- Exec into the SCM container using the following command
  `sudo podman exec -it cloudmanager bash`
- For deployed environments, configuration and runtime data are located here:
  `/home/opc/siebel/<env_id>/<Helm charts repository name>/siebel-config/paramconfig`

  > **Note:** If you're using GitLab as your Git repository, then the name of the Helm charts repository is of the following format: `<namespace>-helmcharts`.

- For a customized configuration that you can deploy in one or more environments, the configuration files are located here:
  `/home/opc/siebel/configuration/<config_id>/config_<namespace>_<config_id>-helmcharts/siebel-config/paramconfig`

Related Topics

Customizing Configurations Prior to Greenfield Deployment

Making Incremental Changes

Use the procedure below to make incremental changes to your Siebel CRM deployment. This topic is part of *Making Incremental Changes to Your Siebel CRM Deployment on OCI*.

> **Note:** Before making any updates, review all of the topics in this section.

To make incremental changes:

1. SSH into the SCM virtual machine instance.
2. Execute the following commands:
   `sudo podman ps`
   `sudo podman exec -it cloudmanager bash`
3. Execute the following command:
   `source /home/opc/siebel/<env_id>/k8sprofile`
4. Execute the following command:
   `cd /home/opc/siebel/<env_id>/<Helm charts repository name>/siebel-config/paramconfig`

   > **Note:** All configuration and runtime data for a deployed environment is located in `/home/opc/siebel/<env_id>/<Helm charts repository name>/siebel-config/paramconfig`.

5. Edit the required YAML file with the incremental changes you require (for example, run `vi enterprise.yaml`).
   For details, see *How Incremental Changes Are Processed* and all the remaining topics in this section.

   > **Note:** Back up all YAML files before you modify them, to help you back out your changes if you experience errors.

6. Edit the file `/home/opc/siebel/<env_id>/<Helm charts repository name>/siebel-config/Chart.yaml`. Increment the value of `version` (for example, increment 0.1.0 to 0.1.1).

7. Execute commands like the following to specify the files that are part of the update you are making and to push these changes to the deployment:

   > **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

   ```
   cd /home/opc/siebel/<env_id>/<Helm charts repository name>/siebel-config

   git status

   git add <modifiedfile1> <modifiedfile2>

   git commit -m "<message or comment>"

   git push
   ```

   Flux automatically upgrades the `siebel-config` Helm chart. The automatic flux reconcile might take a few minutes.
8. If needed, run `flux reconcile` manually using the following commands:

   ```
   flux reconcile source git siebel-repo -n <namespace>

   flux reconcile kustomization apps -n <namespace>
   ```
9. Verify the new version of siebel-config using the following command:

   ```
   flux get all -n <namespace>
   ```

   Incremental changes, including modified files, are pushed to configmaps, and siebel-controller will automatically pick up changes and execute the required actions.
10. To verify the incremental changes after performing the previous steps, wait at least 5 minutes. Then you can verify your changes from the Siebel Management Console (SMC) or by using server manager.

    If the changes are not reflected even after about 10 minutes, to review and analyze logs, first get the name of the siebel-controller pod using the following command:

    ```
    kubectl get pods -n <namespace>
    ```

    In this command, `<namespace>` is the relevant namespace for your deployment. Then, to see the logs, enter a command like the following:

    ```
    kubectl logs -n <namespace> siebel-controller-<pod_id> -f
    ```

    In this command, `<pod_id>` is the pod ID for which you want to view logs. To view only the latest logs, you can optionally use `--tail=0` at the end of this command.

How Incremental Changes Are Processed

Note the following considerations relevant to how incremental changes are processed. This topic is part of *Making Incremental Changes to Your Siebel CRM Deployment on OCI*.

- For each upgrade or flux reconcile that you perform, configure job runs, which validate the application configuration. These runs typically take about 2 minutes.

- The Siebel Controller picks up the runtime incremental changes as soon as the above configure job is completed.
- A continuous synchronization operation identifies the changes in configmaps and does the required actions. It might take up to 10 minutes for synchronization to identify the changes.
- In some cases, server restart is required, which is handled by the controller. Whenever a server restart happens, wait at least 5 minutes before you run the next incremental changes, so that the server state will be good.
- YAML files function as configuration files, which are both case-sensitive and indentation-sensitive. YAML uses spaces ( ) to define document structure. YAML does not allow tabs (\t).
- As illustrated in Step 10 of the procedure in *Making Incremental Changes*, replace `<namespace>` with your namespace. Also replace `<env_id>` with your environment ID.

  > **Note:** Do not replace `<enterprise_name>` and `<server_name>` in the URLs, because these are substituted programmatically.

- To reduce errors in the code flow, use the YAML validator before adding a block to these files representing incremental configuration changes.
- The configuration files `server_edge.yaml` and `sai_quantum.yaml` are Siebel Server (sieb_server) and Siebel Application Interface (SAI) profile configuration (ai_profile) files, respectively. The names *edge* and *quantum* are assigned automatically.
  - Any file with the prefix *server_* refers to a Siebel Server (sieb_server). For example, in the filename `server_edge.yaml`, *edge* is a sieb_server name. Any changes you make in this file apply to all replicas of the server *edge*.

    > **Note:** You can create other YAML configuration files to configure other Siebel Server instances that are not replicas of *edge*, as described in *Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers*.

  - Any file with the prefix *sai_* refers to a SAI profile configuration (ai_profile). For example, in the filename `sai_quantum.yaml`, *quantum* is an ai_profile name. Any changes you make in this file apply to all replicas of the ai_profile *quantum*.

    > **Note:** You can create other YAML configuration files to configure any other SAI instances that are not replicas of *quantum*, as described in *Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers*.

  > **Note:** In this section, the files `siebel_edge.yaml` and `sai_quantum.yaml` are used as example configuration files for Siebel Server and SAI. In your Siebel CRM deployment environment, other files might apply instead of or in addition to these files.

- Other YAML files in which you can make configuration changes include `enterprise.yaml`, `comp_definitions.yaml`, `named_subsystem.yaml`, `siebel-ingress-app.yaml`, and `siebel.yaml`.

Templates for Different Runtime Entities

The following table identifies several YAML files used for different types of configuration for Siebel CRM runtime entities, and provides sample data formats. This topic is part of *Making Incremental Changes to Your Siebel CRM Deployment on OCI*.

Note: As noted, YAML follows proper indentation. Make sure to keep the same format as shown below when you copy and edit the content.

Templates for Different Runtime Entities

Name/Description	Sample Data Format

Enterprise parameters
 	
File:
 	
enterprise.yaml
 	
Add under parameters header.
	In enterprise.yaml
	
	`- basic_params:`
	`  - PA_ALIAS: NumRetries`
	`  PA_VALUE: 10001`
	`- hidden_params: []`
	`- advanced_params:`
	`  - PA_ALIAS: FileSystem`
	`  PA_VALUE: /sfs`

Server parameters
 	
File:
 	
server_edge.yaml
 	
Add under parameters header in applicable section.
	In server_edge.yaml:
	
	`- basic_params:`
	`  - PA_ALIAS: CFGEnableOLEAutomation`
	`  PA_VALUE: 'False'`
	`  - PA_ALIAS: MaxThreads`
	`  PA_VALUE: '12'`
	`  - PA_ALIAS: NotifyHandler`
	`  PA_VALUE: AdminEmailAlert`
	`  - hidden_params: []`
	`  - advanced_params: []`

Component parameters
 	
File:
 	
server_edge.yaml
 	
Add under parameters header in applicable section.
	In server_edge.yaml:
	
	`- basic_params:`
	`  - PA_ALIAS: MaxTasks`
	`  PA_VALUE: '200'`
	`- hidden_params: []`
	`- advanced_params: []`

Component definitions
 	
File:
 	
comp_definitions.yaml
 	
Add under component_definitions header.
	In comp_definitions.yaml:
	
	`CustomADMBatchProc`
	`  definition:`
	`  CC_ALIAS: CustomADMBatchProc`
	`  CC_DESC_TEXT: Exports data items in batch`
	`  CC_DISP_ENABLE_ST: Active`
	`  CC_ENABLE_STATE: Enabled`
	`  CC_INCARN_NO: '0'`
	`  CC_NAME: Application Deployment Manager Batch Processor`
	`  CC_RUNMODE: Batch`
	`  CG_ALIAS: ADM`
	`  CG_NAME: Application Deployment Manager`
	`  CT_ALIAS: UDA Service`
	`  CT_NAME: Custom Business Service Manager`
	`  parameters:`
	`  - basic_params:`
	`  - PA_ALIAS: Method`
	`  PA_VALUE: BatchExport`
	`  - advanced_params:`
	`  - PA_ALIAS: CFGRepositoryFile`

Name/Description	Sample Data Format
	PA_VALUE: siebel_sia.srf
Component group definitions File: comp_definitions.yaml Add under component_groups header.	In comp_definitions.yaml: CustomLoyaltyEngine: definition: CG_ALIAS: CustomLoyaltyEngine CG_DESC_TEXT: Siebel Loyalty Engine Components CG_DISP_ENABLE_ST: Enabled CG_ENABLE_STATE: Enabled CG_ENT_ENABLED: Y CG_NAME: Siebel Loyalty Engine CG_NUM_COMPONENTS: '3'
Named subsystem definitions File: named_subsystem.yaml Add under named_subsystem header.	In named_subsystem.yaml: CustomADSISecAdpt: definition: NSS_ALIAS: CustomADSISecAdpt NSS_DESC: Custom ADSI Security Adapter used for authentication by customer facing applicationss NSS_NAME: ADSI Security Adapter SS_ALIAS: InfraSecAdpt_LDAP parameters: - basic_params: - PA_ALIAS: CredentialsAttributeType PA_VALUE: physicalDeliveryOfficeName - PA_ALIAS: SecAdptDllName PA_VALUE: sscfadsi - PA_ALIAS: ServerName PA_VALUE: CHANGE_ME
Enable component group on a server File: server_edge.yaml Add under component_groups header in applicable section.	In server_edge.yaml: SiebelWebTools: components: SWToolsObjMgr_enu: definition: CC_ALIAS: SWToolsObjMgr_enu CC_RUNMODE: Interactive CG_ALIAS: SiebelWebTools CT_ALIAS: AppObjMgr parameters: - basic_params: []
Enable components on a server File: server_edge.yaml Add under component_groups header in applicable section.	In server_edge.yaml: SWToolsObjMgr_enu: definition: CC_ALIAS: SWToolsObjMgr_enu CC_RUNMODE: Interactive CG_ALIAS: SiebelWebTools CT_ALIAS: AppObjMgr parameters: - basic_params: []

Use Cases for Making Incremental Changes

This topic describes some of the tasks you perform to support use cases for making incremental changes to your Siebel CRM deployment on OCI. This topic is part of *Making Incremental Changes to Your Siebel CRM Deployment on OCI*.

> **Note:** Back up all YAML files before you modify them, to help you back out your changes if you experience errors. For information applicable to the tasks that are part of these use cases, see *Making Incremental Changes* and *How Incremental Changes Are Processed*.

This topic contains the following information:

- *Use Cases for Setting Parameters*
- *Use Cases for Creating or Removing Custom Entities*
- *Use Cases for Enabling Component Groups or Components*
- *Use Cases for Changing Log Level While Running PostInstallDB Setup*
- *Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers*
- *Use Cases for Adding Web Artifacts and Other Siebel Artifact Files*
- *Use Cases for Updating Certificates for SISNAPI with TLS*
- *Use Cases for Updating Keystore File as Part of Incremental Changes*
- *Use Cases for Rerunning the PostInstallDB Job*

Use Cases for Setting Parameters

This topic provides detailed information about changes to make to support use cases for setting parameters. This topic is part of *Use Cases for Making Incremental Changes*.

Use Cases for Setting Parameters

Use Case/Notes	Sample Data Format

1a. Enterprise parameter, adding
File:
enterprise.yaml
Flag settings:
`PA_EFF_SRVR_RSTRT = N`
`PA_EFF_CMP_RSTRT = N`
Restart:
• No restart.
Verification:
• Updated parameter value can be seen in SMC configuration screen.
	`- basic_params:`
	`  - PA_ALIAS: NumRetries`
	`  PA_VALUE: '10001'`

Use Case/Notes	Sample Data Format
1b. Enterprise parameter, adding File: enterprise.yaml Flag settings: **PA_EFF_SRVR_RSTRT = Y** **PA_EFF_CMP_RSTRT = N** Restart: • Full restart. Log in to pod and run **siebps**. Check timestamp of restart (UTC time). Verification: • Updated parameter value can be seen in SMC configuration screen.	**- basic_params:** **- PA_ALIAS: EnableWorkspace** **PA_VALUE: 'True'**
2. Enterprise parameter, modifying File: enterprise.yaml Flag settings: **PA_EFF_SRVR_RSTRT = N** **PA_EFF_CMP_RSTRT = N** Restart: • No restart. Verification: • Updated parameter value can be seen in SMC configuration screen.	**- basic_params:** **- PA_ALIAS: NumRetries** **PA_VALUE: '10002'**
3. Enterprise parameter, removing configured value File: enterprise.yaml Restart: • No restart. Verification: • Default parameter value can be seen in SMC configuration screen.	Delete PA_ALIAS and PA_VALUE pair for a single parameter.
4a. Server parameter, adding File:	**- basic_params:** **- PA_ALIAS: NumRetries**

Use Case/Notes	Sample Data Format
server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = N** **PA_EFF_CMP_RSTRT = N** Restart: • No restart . Verification: • Updated parameter value can be seen in SMC configuration screen.	**PA_VALUE: '10005'**
4b. Server parameter, adding File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = Y** **PA_EFF_CMP_RSTRT = Y** Restart: • Server restart. All replicas of edge server are restarted (inside pod, stop_server_all and start_server_all are executed). Log in to pod and run siebps. Check timestamp of restart (UTC time). Verification: • Updated parameter value can be seen in SMC configuration screen.	**- basic_params:** **- PA_ALIAS: ConfigLdapAuthTimeout** **PA_VALUE: '20'** **- PA_ALIAS: EnableVirtualHosts** **PA_VALUE: 'True'**
5. Server parameter, modifying File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = Y** **PA_EFF_CMP_RSTRT = N** Restart: • Server restart. All replicas of edge server are restarted (inside pod, stop_server_all and start_server_all are executed). Verification:	**- basic_params:** **- PA_ALIAS: EnableVirtualHosts** **PA_VALUE: 'False'**

Use Case/Notes	Sample Data Format
• Updated parameter value can be seen in SMC configuration screen.	
6. Server parameter, removing configured value File: server_edge.yaml Restart: • No restart. Verification: • Default parameter value can be seen in SMC configuration screen.	Delete PA_ALIAS and PA_VALUE pair for a single parameter.
7a. Component parameter, adding File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = Y** **PA_EFF_CMP_RSTRT = N** Restart: • Server restart. All replicas of edge server are restarted (inside pod, stop_server_all and start_server_all are executed). Log in to pod and run **siebps**. Check timestamp of restart (UTC time). Verification: • Updated parameter value can be seen in SMC configuration screen.	**- basic_params:** **- PA_ALIAS: MaxTasks** **PA_VALUE: '50'**
7b. Component parameter, adding File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = N** **PA_EFF_CMP_RSTRT = Y** Restart: • Component restart. Verification:	**- basic_params:** **- PA_ALIAS: ConfigLdapAuthTimeout** **PA_VALUE: '15'**

Use Case/Notes	Sample Data Format
• Updated parameter value can be seen in SMC configuration screen.	
8. Component parameter, modifying File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = N** **PA_EFF_CMP_RSTRT = N** Restart: • No restart. Verification: • Updated parameter value can be seen in SMC configuration screen.	**- basic_params:** **- PA_ALIAS: ConfigLdapAuthTimeout** **PA_VALUE: '20'**
9. Component parameter, removing configured value File: server_edge.yaml Flag settings: **PA_EFF_SRVR_RSTRT = N** **PA_EFF_CMP_RSTRT = N** Restart: • No restart. Verification: • Default parameter value can be seen in SMC configuration screen.	Delete PA_ALIAS and PA_VALUE pair for a single parameter.
10. Named subsystem parameter, adding (no restart) File: named_subsystem.yaml Verification: • Updated parameter value can be seen in SMC configuration screen.	Specific to named subsystem.
11. Named subsystem parameter, modifying (no restart)	Specific to named subsystem.

Use Case/Notes	Sample Data Format
File: named_subsystem.yaml Verification: • Updated parameter value can be seen in SMC configuration screen.	
12. Named subsystem parameter, removing (no restart) File: named_subsystem.yaml Verification: • SMC configuration screen is updated.	Specific to named subsystem.
13. Component definition parameter, adding (no restart) File: comp_definitions.yaml Verification: • Updated parameter value can be seen in SMC configuration screen.	Specific to named subsystem.
14. Component definition parameter, modifying (no restart) File: comp_definitions.yaml Verification: • Updated parameter value can be seen in SMC configuration screen.	Specific to named subsystem.
15. Component definition parameter, removing (no restart) File: comp_definitions.yaml Verification: • SMC configuration screen is updated.	Specific to named subsystem.

Use Cases for Creating or Removing Custom Entities

This topic provides detailed information about changes to make to support use cases for creating or removing custom entities. No restarts apply in these use cases. This topic is part of *Use Cases for Making Incremental Changes*. You can create or remove the following custom entities:

- Component definitions
- Named subsystem definitions
- Component group definitions

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Use Cases for Creating or Removing Custom Component Definitions, Named Subsystem Definitions, and Component Group Definitions

Use Case/Notes	Sample Data Format

1. Component definition, creating
 	
File:
 	
comp_definitions.yaml
 	
Notes:
 	
• Replace all occurrences of the CustomADMBatchProc definition shown here, and its settings and parameters, with those for your custom component definition.
 	
Verification:
 	
• New component definition can be seen in SMC configuration screen.
	`CustomADMBatchProc:`
	`  definition:`
	`  CC_ALIAS: CustomADMBatchProc`
	`  CC_DESC_TEXT: Exports data items in batch`
	`  CC_DISP_ENABLE_ST: Active`
	`  CC_ENABLE_STATE: Enabled`
	`  CC_INCARN_NO: '0'`
	`  CC_NAME: Application Deployment Manager Batch Processor`
	`  CC_RUNMODE: Batch`
	`  CG_ALIAS: ADM`
	`  CG_NAME: Application Deployment Manager`
	`  CT_ALIAS: UDA Service`
	`  CT_NAME: Custom Business Service Manager`
	`  parameters:`
	`  - basic_params:`
	`  - PA_ALIAS: Method`
	`  PA_VALUE: BatchExport`
	`  - advanced_params: []`

2. Component definition, removing configuration File: comp_definitions.yaml Notes: • Delete the relevant block under the component_definitions header. Verification: • Component definition removal can be seen in SMC configuration screen.	Remove the entire block representing the custom component definition (for example, the configuration block in the previous row).
3. Named subsystem, creating	`CustomADSISecAdpt:` `  definition:`

Use Case/Notes	Sample Data Format

File:
 	
named_subsystem.yaml
 	
Notes:
 	
• Add the relevant block under the named_subsystem header.
• Replace all occurrences of the CustomADSISecAdpt definition shown here, and its settings and parameters, with those for your custom named subsystem definition.
 	
Verification:
 	
• New named subsystem can be seen in SMC configuration screen.
	<pre>NSS_ALIAS: CustomADSISecAdpt
	NSS_DESC: Custom ADSI Security Adapter used for authentication by customer facing applicationss
	NSS_NAME: ADSI Security Adapter
	SS_ALIAS: InfraSecAdpt_LDAP
	parameters:
	- basic_params:
	- PA_ALIAS: CredentialsAttributeType
	PA_VALUE: physicalDeliveryOfficeName
	- PA_ALIAS: SecAdptDllName
	PA_VALUE: sscfadsi
	- PA_ALIAS: ServerName
	PA_VALUE: CHANGE_ME</pre>

4. Named subsystem, removing configuration File: named_subsystem.yaml Notes: • Delete the relevant block under the named_subsystem header. Verification: • Named subsystem definition removal can be seen in SMC configuration screen.	Remove the entire block representing the custom named subsystem definition (for example, the configuration block in the previous row).

5. Component group, creating
 	
File:
 	
comp_definitions.yaml
 	
Notes:
 	
• Add the relevant block under the component_groups header.
• Replace all occurrences of the CustomLoyaltyEngine definition shown here, and its settings and parameters, with those for your custom component group definition.
 	
Verification:
 	
• New component group can be seen in SMC configuration screen.
	<pre>CustomLoyaltyEngine:
	definition:
	CG_ALIAS: CustomLoyaltyEngine
	CG_DESC_TEXT: Siebel Loyalty Engine Components
	CG_DISP_ENABLE_ST: Enabled
	CG_ENABLE_STATE: Enabled
	CG_ENT_ENABLED: Y
	CG_NAME: Siebel Loyalty Engine
	CG_NUM_COMPONENTS: '3'</pre>

6. Component group, removing configuration	Remove the entire block representing the custom component group definition (for example, the configuration block in the previous row).

Use Case/Notes	Sample Data Format
File: comp_definitions.yaml Notes: • Delete the relevant block under the component_group header. Verification: • Component group definition removal can be seen in SMC configuration screen.	

Use Cases for Enabling Component Groups or Components

This topic provides detailed information about changes to make to support use cases for enabling component groups or components on a server. This topic is part of *Use Cases for Making Incremental Changes*. No restarts apply in these use cases.

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Use Case/Notes	Sample Data Format

1. Component group, enabling on a server
 	
Files:
 	
• server_edge.yaml
• sai_quantum.yaml
• siebel-ingress-app.yaml
Notes:
 	
In server_edge.yaml:
 	
• Add the relevant block under the component_groups header.
• Replace all occurrences of the SiebelWebTools component group shown here, and its components,with those for your custom component definition.
• All components in that component group must be mentioned. Components that are not identified are disabled.
In sai_quantum.yaml:
	In server_edge.yaml :
	`SiebelWebTools:`
	`components:`
	`SWToolsObjMgr_enu:`
	`definition:`
	`CC_ALIAS: SWToolsObjMgr_enu`
	`CC_RUNMODE: Interactive`
	`CG_ALIAS: SiebelWebTools`
	`CT_ALIAS: AppObjMgr`
	`parameters:`
	`- basic_params: []`
	
	In server_edge.yaml : Add the new Component group to be enabled in "profiles → ServerConfigParams → EnableCompGroupsSIA". This change is required to keep the server_edge.yaml Profile configuration to be intact with the list of Component Groups enabled in the server.
	
	`profiles:`
	`Profile:`
	`LastUpdated: 2021/12/21 11:08:59`
	`ProfileName: siebel`
	`ServerConfigParams:`
	`EnableCompGroupsSIA: EAI,SiebelWebTools`
	
	In sai_quantum.yaml:
	
	`sai_quantum:`
	`profiles:`
	`- ConfigParam:`
	`Applications:`

Use Case/Notes	Sample Data Format

• Add entry for each Interactive Component/Object Manager in the component group in ConfigParam/ Applications
• Substitute the values required for your object manager component.
• Perform git add, git commit, and git push operations after updating sai_ quantum.yaml.
In siebel-ingress-app.yaml:
• Update the required ingress content to provide access to components and services.
• Substitute the values required for your deployment, to confirm the updates made in the other configuration files.
• Perform git add, git commit, and git push operations after updating siebel-ingress-app.yaml.
	`- AnonUserPool: 0`
	`AppDisplayName: ''`
	`AppDisplayOrder: 0`
	`AppIcon: ''`
	`AuthenticationProperties:`
	`AnonPassword: *****`
	`AnonUserName: GUESTCST`
	`GuestSessionTimeout: 300`
	`MaxTabs: 1`
	`SessionTimeout: 900`
	`SessionTimeoutWLCommand: UpdatePrefMsg`
	`SessionTimeoutWLMethod: HeartBeat`
	`SessionTimeoutWarning: 60`
	`SessionTokenMaxAge: 2880`
	`SessionTokenTimeout: 900`
	`SingleSignOn: false`
	`TrustToken: ''`
	`UserSpec: ''`
	`AvailableInSiebelMobile: false`
	`EAISOAPMaxRetry: 0`
	`EAISOAPNoSessInPref: false`
	`EnableExtServiceOnly: false`
	`Language: enu`
	`Name: webtools`
	`ObjectManager: SWToolsObjMgr_enu`
	`StartCommand: ''`
	`UseAnonPool: false`
	In siebel-ingress-app.yaml (located in the Siebel Cloud Manager Container `/home/opc/siebel/<env_id>/<namespace>-cloudmanager/flux-crm/traefik/traefik-resources`):
	`- match: PathPrefix(`/siebel/app/siebelwebtools/enu`)`
	`kind: Rule`
	`middlewares:`
	`- name: siebel-add-slash`
	`- name: siebel-max-body-size`
	`services:`
	`- name: quantum`
	`port: 4430`
	`scheme: https`
	`serversTransport: siebel-transport`
	`sticky:`
	`cookie:`
	`name: siebel-app-ingress`
	`httpOnly: true`
	`secure: true`
	`maxAge: 172800`

2. Component group, removing configuration
Files:
• server_edge.yaml
• sai_quantum.yaml
• siebel-ingress-app.yaml
Notes:
In server_edge.yaml:
	In server_edge.yaml:
	Remove the entire block representing the component group configuration (for example, the below configuration block for removing SiebelWebTools).
	`SiebelWebTools:`
	`components:`
	`SWToolsObjMgr_enu:`
	`definition:`
	`CC_ALIAS: SWToolsObjMgr_enu`
	`CC_RUNMODE: Interactive`
	`CG_ALIAS: SiebelWebTools`
	`CT_ALIAS: AppObjMgr`
	`parameters:`
	`- basic_params: []`

Use Case/Notes	Sample Data Format
• Delete the relevant block under the component_groups header. • Perform git add, git commit, and git push operations after updating server_edge.yaml. • Verification: Component group removal can be seen in SMC configuration screen In sai_quantum.yaml: • Delete the entry for each Interactive Component/Object Manager in the component group in ConfigParam/ Applications • Perform git add, git commit, and git push operations after updating sai_ quantum.yaml. • Verification: Component group removal can be seen in SMC configuration screen. In siebel-ingress-app.yaml: • Delete the object manager related ingress content rule. • Perform git add, git commit, and git push operations after updating siebel-ingress-app.yaml.	In server_edge.yaml : Remove the Component group to be disabled from " Profiles→ServerConfigParams → EnableCompGroupsSIA ". This change is required to keep the server_ edge.yaml Profile configuration to be intact with the list of Component Groups enabled in the server. **rofiles:** **Profile:** **LastUpdated: 2021/12/21 11:08:59** **ProfileName: siebel** **ServerConfigParams:** **EnableCompGroupsSIA: SiebelWebTools (To be removed)** In sai_quantum.yaml: Remove the entire block representing the Interactive Object_manager configuration (for example, the below configuration block for removing SWToolsObjMgr_enu). **sai_quantum:** **profiles:** **- ConfigParam:** **Applications:** **- AnonUserPool: 0** **AppDisplayName: ''** **AppDisplayOrder: 0** **AppIcon: ''** **AuthenticationProperties:** **AnonPassword: ***** **AnonUserName: GUESTCST** **GuestSessionTimeout: 300** **MaxTabs: 1** **SessionTimeout: 900** **SessionTimeoutWLCommand: UpdatePrefMsg** **SessionTimeoutWLMethod: HeartBeat** **SessionTimeoutWarning: 60** **SessionTokenMaxAge: 2880** **SessionTokenTimeout: 900** **SingleSignOn: false** **TrustToken: ''** **UserSpec: ''** **AvailableInSiebelMobile: false** **EAISOAPMaxRetry: 0** **EAISOAPNoSessInPref: false** **EnableExtServiceOnly: false** **Language: enu** **Name: webtools** **ObjectManager: SWToolsObjMgr_enu** **StartCommand: ''** **UseAnonPool: false** In siebel-ingress-app.yaml : Remove the below configuration for removing the siebelwebtools endpoint. (located in the Siebel Cloud Manager Container **/home/opc/siebel/<env_id>/<namespace>-cloudmanager/flux-crm/ traefik/traefik-resources**): **- match: PathPrefix(`/siebel/app/siebelwebtools/enu`)** **kind: Rule** **middlewares:** **- name: siebel-add-slash** **- name: siebel-max-body-size** **services:** **- name: quantum** **port: 4430** **scheme: https**

Use Case/Notes	Sample Data Format

	`serversTransport: siebel-transport`
	`sticky:`
	`cookie:`
	`name: siebel-app-ingress`
	`httpOnly: true`
	`secure: true`
	`maxAge: 172800`

3. Component, enabling on a server.
 	
Files:
• server_edge.yaml
• sai_quantum.yaml
• siebel-ingress-app.yaml
Notes:
 	
In server_edge.yaml:
• Provide the required values under the components header.
• Replace the component shown here with those for the components you are enabling.
• Perform git add, git commit, and git push operations after updating server_edge.yaml
In sai_quantum.yaml:
• Add entry for Interactive Component/Object Manager in ConfigParam/Applications
• Substitute the values required for your object manager component.
• Perform git add, git commit, and git push operations after updating sai_quantum.yaml.
In siebel-ingress-app.yaml:
• Update the required ingress content to provide access to components and services.
• Substitute the values required for your deployment, to confirm the updates made in the other configuration files.
• Perform git add, git commit, and git push operations after updating siebel-ingress-app.yaml.
	In server_edge.yaml:
	
	`SWToolsObjMgr_enu:`
	`definition:`
	`CC_ALIAS: SWToolsObjMgr_enu`
	`CC_RUNMODE: Interactive`
	`CG_ALIAS: SiebelWebTools`
	`CT_ALIAS: AppObjMgr`
	`parameters:`
	`- basic_params: []`
	
	In sai_quantum.yaml:
	
	`- AnonUserPool: 0`
	`AppDisplayName: ''`
	`AppDisplayOrder: 0`
	`AppIcon: ''`
	`AuthenticationProperties:`
	`AnonPassword: *****`
	`AnonUserName: GUESTCST`
	`GuestSessionTimeout: 300`
	`MaxTabs: 1`
	`SessionTimeout: 900`
	`SessionTimeoutWLCommand: UpdatePrefMsg`
	`SessionTimeoutWLMethod: HeartBeat`
	`SessionTimeoutWarning: 60`
	`SessionTokenMaxAge: 2880`
	`SessionTokenTimeout: 900`
	`SingleSignOn: false`
	`TrustToken: ''`
	`UserSpec: ''`
	`AvailableInSiebelMobile: false`
	`EAISOAPMaxRetry: 0`
	`EAISOAPNoSessInPref: false`
	`EnableExtServiceOnly: false`
	`Language: enu`
	`Name: webtools`
	`ObjectManager: SWToolsObjMgr_enu`
	`StartCommand: ''`
	`UseAnonPool: false`
	
	In siebel-ingress-app.yaml (located in `/home/opc/siebel/<env_id>/<namespace>-cloudmanager/flux-crm/traefik/traefik-resources`):
	
	`- match: PathPrefix(`/siebel/app/siebelwebtools/enu`)`
	`kind: Rule`
	`middlewares:`
	`- name: siebel-add-slash`
	`- name: siebel-max-body-size`
	`services:`
	`- name: quantum`
	`port: 4430`
	`scheme: https`
	`serversTransport: siebel-transport`

Use Case/Notes	Sample Data Format

	`sticky:`
	`cookie:`
	`name: siebel-app-ingress`
	`httpOnly: true`
	`secure: true`
	`maxAge: 172800`

4. Component, removing configuration. Files: • server_edge.yaml • sai_quantum.yaml • siebel-ingress-app.yaml Notes: In server_edge.yaml: • Delete the relevant block under the components header. • Perform git add, git commit, and git push operations after updating server_edge.yaml. • Verification: Component group removal can be seen in SMC configuration screen In sai_quantum.yaml: • Delete the entry for Interactive Component/Object Manager in the component group in ConfigParam/ Applications • Perform git add, git commit, and git push operations after updating sai_ quantum.yaml. • Verification: Component group removal can be seen in SMC configuration screen. In siebel-ingress-app.yaml: • Delete the object manager related ingress content rule. • Perform git add, git commit, and git push operations after updating siebel-ingress-app.yaml.	In server_edge.yaml: Remove the entire block representing the component configuration. (Refer Use case 2, Removing a component group) In sai_quantum.yaml: Remove the entire block representing the Interactive Object_manager configuration.(Refer Use case 2, Removing a component group) In siebel-ingress-app.yaml : Remove the ingress content rule for removing the Object manager endpoint. (Refer Use case 2, Removing a component group)

Use Cases for Changing Log Level While Running PostInstallDB Setup

This topic provides detailed information about changes to make to support use cases for changing log levels associated with the process of running PostInstallDBSetup. This topic is part of *Use Cases for Making Incremental Changes*.

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Use Case/Notes	Sample Data

Set/change log level for PostInstallDBSetup
 	
File:
 	
siebel.yaml
 	
Notes:
 	
• Add the parameters for logs under values in the file siebel.yaml located in the Siebel Cloud Manager container in:
 	
`/home/opc/siebel/<ENV_ID>/<Cloud manager repository name>/flux-crm/apps/base/siebel`
 	
• Perform git pull, add, git commit, and git push operations
	`values:`
	`  logs:`
	`  siebelLogEvents: 5`
	`  dbUtilLogEvents: "SQLParseAndExecute=5,SQLDBUtilityLog=5"`

Use Cases for Adding Profiles, Deployments, or Adding Resources to Individual Siebel Servers

This topic provides detailed information about changes to make to support use cases for adding profiles, deployments, or for adding resources to individual Siebel Servers. No restarts required in these use cases. This topic is part of *Use Cases for Making Incremental Changes*.

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Use Cases for Adding Profiles or Deployments

Use Case/Notes	Sample Data Format

1. Increase replicas (for ses, sai) or Change resources for individual Siebel Servers (not for sai servers)
 	
Note: sesResources defined at the profile level for individual Siebel server takes higher precedence over the generic sesResources overridden in payload.
 	
File:
	`siebelServer:`
	`- profile: siebel`
	`replicas: 3`
	`sesResources:`
	`limits:`
	`cpu: 4`
	`memory: 24Gi`
	`requests:`
	`cpu: 1`
	`memory: 8Gi`

Use Case/Notes	Sample Data Format
siebel.yaml Notes: • Edit siebel.yaml under: **/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel** • Increase replicas under siebelServer for the respective pod, using these commands: git pull git add . git commit -m "<message>" git push For cgw replica number value change, the siebel-gateway.yaml and siebel-config.yaml should be changed.	**siebsrvr_prefix: edge** **- profile: siebel** **replicas: 3** **sesResources:** **limits:** **cpu: 4** **memory: 24Gi** **requests:** **cpu: 1** **memory: 8Gi** **siebsrvr_prefix: tibus** **saiServer:** **- profile: sai_lily** **replicas: 2** **sai_prefix: quantum** **- profile: slc15zny95121** **replicas: 2** **sai_prefix: alchemist**
2. Add a new Siebel Server profile File: New YAML file Notes: • Create file name with prefix **server_** (for example, server_lily.yaml). • Under the server name are: o component_groups has all comp group and comp details o parameters has server level parameters o deployment is used during server deployment o Profile is used to create server profile • Modify all occurrences of the following, according to your configuration: o <namespace> o <profile_name> o <username> o <password> o <guest username> o <guest password>	**server_lily:** **component_groups:** **CallCenter:** **components:** **SCCObjMgr_enu:** **definition:** **CC_ALIAS: SCCObjMgr_enu** **CC_RUNMODE: Interactive** **CG_ALIAS: CallCenter** **CT_ALIAS: AppObjMgr** **parameters:** **- basic_params:** **- PA_ALIAS: MaxTasks** **PA_VALUE: '200'** **SServiceObjMgr_enu:** **definition:** **CC_ALIAS: SServiceObjMgr_enu** **CC_RUNMODE: Interactive** **CG_ALIAS: CallCenter** **CT_ALIAS: AppObjMgr** **parameters: []** **eServiceObjMgr_enu:** **definition:** **CC_ALIAS: eServiceObjMgr_enu** **CC_RUNMODE: Interactive** **CG_ALIAS: CallCenter** **CT_ALIAS: AppObjMgr** **parameters:** **- basic_params:** **- PA_ALIAS: MaxTasks** **PA_VALUE: '222'** **deployment:** **DeploymentInfo:** **ProfileName: <profile_name>**

Use Case/Notes	Sample Data Format

	<pre>ServerDeployParams:
	DeployedLanguage: enu
	PrimaryLanguage: ENU
	parameters:
	- basic_params:
	- PA_ALIAS: CFGEnableOLEAutomation
	PA_VALUE: 'False'
	- PA_ALIAS: MaxThreads
	PA_VALUE: '11'
	- hidden_params: []
	- advanced_params: []
	profiles:
	Profile:
	LastUpdated:
	ProfileName: <profile_name>
	ServerConfigParams:
	AnonLoginPassword: <guest password>
	AnonLoginUserName: <guest username>
	CACertFileName: null
	CertFileNameServer: null
	ClusteringEnvironmentSetup: NotClustered
	Db2InstHome: ''
	EnableCompGroupsSIA: CallCenter
	Encrypt: null
	LocalSynchMgrPort: '40400'
	ModifyServerAuth: null
	ModifyServerEncrypt: null
	NameserverHostName: siebelcgw-0
	NamesrvrPort: '8888'
	Password: <password>
	SCBPort: '2321'
	SiebelClusterGateway: null
	SiebelEnterprise: siebel
	SqlServerPort: null
	UseOracleConnector: 'true'
	Username: <username></pre>

3. Add a new Siebel Application Interface profile
 	
File:
 	
New YAML file
 	
Notes:
 	
• Create file name with prefix **sai_** (for example, sai_trust.yaml).
• Under profiles, modify all occurrences of the following according to your configuration:
 	
o <namespace>
o <profile_name>
o <guest username>
o <guest password>
	<pre>sai_trust:
	profiles:
	- ConfigParam:
	Applications:
	- AnonUserPool: 0
	AppDisplayName: ''
	AppDisplayOrder: 0
	AppIcon: ''
	AuthenticationProperties:
	AnonPassword: <guest password>
	AnonUserName: <guest username>
	GuestSessionTimeout: 300
	MaxTabs: 1
	SessionTimeout: 900
	SessionTimeoutWLCommand: HeartBeat
	SessionTimeoutWLMethod: UpdatePrefMsg
	SessionTimeoutWarning: 60
	SessionTokenMaxAge: 2880
	SessionTokenTimeout: 900
	SingleSignOn: false
	TrustToken: ''
	UserSpec: ''
	AvailableInSiebelMobile: false
	EAISOAPMaxRetry: 0
	EAISOAPNoSessInPref: false</pre>

Use Case/Notes	Sample Data Format

```
        EnableExtServiceOnly: false
        Language: enu
        Name: callcenter
        ObjectManager: sccobjmgr_enu
        StartCommand: ''
        UseAnonPool: false
      ConnMgmt:
        CACertFileName: ''
        CertFileName: ''
        KeyFileName: ''
        KeyFilePassword: ''
        PeerAuth: false
        PeerCertValidation: false
      DAV:
        LogProperties:
          LogLevel: ERROR
      EAI:
        LogProperties:
          LogLevel: ERROR
GatewayIdentity:
        AuthToken: null
        GatewayHost: siebelcgw-0
         GatewayPort: '8888'
 RESTInBound:
        Baseuri: https://smc-0.smc.<namespace>.svc.cluster.local:4430/siebel/
v1.0/
        LogProperties:
          LogLevel: ERROR
        MaxConnections: 20
        ObjectManager: sccobjmgr_enu
        RESTAuthenticationProperties:
          AnonPassword: <guest password>
          AnonUserName: <guest username>
          AuthenticationType: Basic
          OAuthEndPoint: ''
          SessKeepAlive: 120
          TrustToken: ''
          UserSpec: ''
          ValidateCertificate: true
        RESTResourceParamList: []
      RESTInBoundResource: []
      RESTOutBound:
        LogProperties:
          LogLevel: ERROR
      SOAPOutBound:
        LogProperties:
          LogLevel: ERROR
      UI:
        LogProperties:
          LogLevel: ERROR
      defaults:
        AuthenticationProperties:
          AnonPassword: <guest password>
          AnonUserName: <guest username>
          GuestSessionTimeout: 300
          MaxTabs: 1
          SessionTimeout: 900
          SessionTimeoutWLCommand: HeartBeat
          SessionTimeoutWLMethod: UpdatePrefMsg
          SessionTimeoutWarning: 60
          SessionTokenMaxAge: 2880
          SessionTokenTimeout: 900
          SingleSignOn: false
```

Use Case/Notes	Sample Data Format

	<pre>TrustToken: ''
	UserSpec: ''
	DoCompression: false
	EnableFQDN: false
	FQDN: ''
	swe:
	AllowStats: true
	Language: ENU
	MaxQueryStringLength: -1
	SeedFile: ''
	SessionMonitor: false
	Profile:
	AccessPermission: ReadWrite
	LastUpdated:
	ProfileName: <profile name></pre>

4. Deploying a Siebel Server
 	
File:
 	
siebel.yaml
 	
Notes:
 	
• Ensure that the status of the configure job that is launched when the Siebel Server profile is added is "Completed".
 	
`source /home/opc/siebel/<env_id>/k8sprofile`
 	
`kubectl -n <namespace> get po -l batch.kubernetes.io/job-name=configure`
 	
If the status of configure job is "Running", the deployment will fail and the SES profile will not appear in SMC even if the pod status is "Running".
• Edit siebel.yaml under:
 	
`/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel`
• Add a new ses section under siebelServer.
• If you do not see the deployment in SMC, check pod logs to find progress or error information.
• The <server prefix> must be unique and be the same as the profile filename suffix: for server_lily.yaml, <server prefix> would be lily.
	<pre>siebelServer:
	- profile: <siebserver profile name>
	replicas: 1
	siebsrvr_prefix: <server prefix>
	sesResources:
	limits:
	cpu: 4
	memory: 24Gi
	requests:
	cpu: 1
	memory: 6Gi</pre>

Use Case/Notes	Sample Data Format
5. Enabling access to Siebel CRM application UIs. For example, Call Center UI. File: **siebel-ingress-app.yaml** Notes: • Edit the **siebel-ingress-app.yaml** file under **/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/traefik/traefik-resources** • Add the Ingress configuration to allow users to access the Siebel CRM application UIs. For example, Call Center UI.	**- match: PathPrefix(`/siebel/app/callcenter/enu`)** **kind: Rule** **middlewares:** **- name: siebel-max-body-size** **services:** **- name: quantum** **port: 4430** **scheme: https** **serversTransport: siebel-transport** **sticky:** **cookie:** **httpOnly: true** **maxAge: 172800** **name: siebel-app-ingress** **secure: true**
6. Deploying a Siebel Application Interface File: siebel.yaml Notes: • Before you begin the deployment, ensure that the status of the configure job that is launched when the Siebel Application Interface profile is added is "Completed": **source /home/opc/siebel/<env_id>/k8sprofile** **kubectl -n <namespace> get po -l batch.kubernetes.io/job-name=configure** If the status of configure job is "Running", the deployment will fail and the SAI profile will not appear in SMC even if the pod status is "Running". • Edit siebel.yaml under: **/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel** • Add a new sai section under saiServer. • If you do not see the deployment in SMC, check pod logs to find progress or error information. • The <sai prefix> must be unique and be the same as the profile filename	**saiServer:** **- profile:<sai profile name>** **replicas: 1** **sai_prefix: <sai prefix>**

Use Case/Notes	Sample Data Format
suffix: for sai_trust.yaml, <server prefix> would be trust.	

Use Cases for Adding Web Artifacts and Other Siebel Artifact Files

This topic provides detailed information about the steps to support use cases for adding or updating web artifacts or other Siebel artifact files after deployment or as part of incremental changes. No restarts apply in these use cases. This topic is part of *Use Cases for Making Incremental Changes*.

Now all the Siebel artifact files will be part to Helm charts Git repository and it is propagated to Siebel applications from here.

One can view and edit Siebel artifacts file from the location `<Helm charts repository name>/siebel-artifacts/build/mde`.

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

To add a new siebel artifacts files after deployment:

1. Exec into the SCM container:
   ```
   sudo podman exec -it cloudmanager bash
   ```
2. Execute the following command:

   ```
   cd /home/opc/siebel/<ENV_ID>/<Helm charts repository name>/siebel-artifacts/build/mde
   ```
3. Add new Siebel artifact files in the required path locations. If the required path does not exist, then you can create the folder path and include new files.
4. Update the `siebel-artifacts` Helm charts version, as follows:

   ```
   vi /home/opc/siebel/<ENV_ID>/<Helm charts repository name>/siebel-artifacts/Chart.yaml
   ```

   Increment the version and save.
5. Push all changes to the remote Git repository, as follows:

   ```
   git pull
   git status
   git add <file1> <file2>
   git commit -m "<message>"
   git push
   ```

   You can also perform the above steps from the Git user interface.

   Once the changes are pushed, flux controllers identify the change in the `siebel-artifacts` Helm charts version and start an update. As part of the update, a pre-update job, image-builder, is triggered. The image-builder job builds a custom container image with a new tag having all the updated artifact files.

   - If there is no difference, then the image-builder job exits with an appropriate message.

- If there are changes from the previous build, then the image-builder job builds a custom container image with a new tag and pushes it to the container registry.

Later, the new container image tag is identified and updated in the required Git references by the flux controllers. Then the siebserver, sai, and gateway pods are restarted and deployed with the new container image having updated Siebel artifact files.

6. Once the pods are restarted, to realize the changes, access one of the containers using the following command and verify if new files are present:

```
kubectl exec -it <pod_name> bash -n <namespace>
```

7. Verify the files in the `files`, `images`, and `scripts` directories from your browser:

```
https://<EXTERNAL_IP>/siebel/files/custom/<file>
https://<EXTERNAL_IP>/siebel/images/custom/<file>
https://<EXTERNAL_IP>/siebel/scripts/siebel/custom/<file>
```

> **Note:** You will find an inline comment # {"$imagepolicy": <namespace>:cm-siebel-image-policy:tag"} next to some image tags in the Git repository. This is a marker used for automation; do not modify this comment. To enable custom web artifacts in applications, you must update manifest entries manually.

Use Cases for Updating Certificates for SISNAPI with TLS

This topic provides detailed information about the steps to support use cases for adding or updating custom Transport Layer Security (TLS) certificate post deployment.

During initial environment provisioning, the TLS files required for Sysnapi with TLS configuration will be extracted from keystore (`keystore.jks` and `keystore_client.jks`) and truststore (`trustore.jks`) files and pushed to Git in the following location:

```
<envdir>/<Helm charts repository name>/siebel-config/tls_certs
```

If one needs to update custom TLS certificate post deployment, the following steps need to be followed:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Go to the Git repository location: `<Helm charts repository name>/siebel-config/tls_certs`
2. Update TLS files, commit, and push the changes.

   Here the filename should be same and only "pem" format is supported.

   The certificates should follow certain rules:

   - ca.key.pem – A private key used for issuing new certificates.
   - ca.cert.pem – A CA certificate that must be imported into `keystore.jks`, `keystore_client.jks`, and `truststore.jks`.
   - server_key.pem – A private key used for issuing Siebel server certificate.
   - server.pem – An SSL certificate with valid DNS entries that must be present in `keystore.jks`.
   - client_key.pem – A private key used for issuing Siebel client certificate.
   - client.pem – A Siebel client certificate that must be present in `keystore_client.jks`.

3. Increment the chart version in file `<Helm charts repository name>/siebel-config/Chart.yaml` and commit changes. Wait for 10 minuntes, so that flux will automatically reconcile and uptake above changes. Alternatively, you can manually reconcile using below commands:

```
flux reconcile source git siebel-repo -n <namespace>
flux reconcile kustomization apps -n <namespace>
```

   The reconcile process might take upto 10 minutes. The new custom TLS files will be pulled and Kubernetes secret - "keystore" will be updated with new values.
4. Execute these commands to update SES, SAI, and CGW containers with new certificates.

   Edit `<Helm charts repository name>/siebel/Chart.yaml`, increment chart version, and commit the same.

   To enable TLS communication for a particular component, one needs to add the below parameter under that component in server yaml file. For more information about parameter addition, see *Making Incremental Changes*.

```
CommType: TLS
```

Use Cases for Updating Keystore File as Part of Incremental Changes

Note: Effective May 1, 2026, many Trusted Root Certification Authorities will no longer issue SSL certificates that include both Server Authentication and Client Authentication Extended Key Usage (EKU). Existing certificates remain valid until expiration or revocation and continue to support both `serverAuth` and `clientAuth` EKUs. However, renewals or new certificates issued after the deadline do not include the Client Authentication EKU and therefore cannot be used for mTLS client authentication.

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Update the `keystore.jks`, `keystore_client.jks`, and `truststore.jks` files in Git:
   - Using browser UI:
     i. Access your Git instance from browser and go to the Helm charts repository.
     ii. Navigate to the `siebel-config/keystore` folder.
     iii. Upload and commit the new custom `keystore.jks`, `keystore_client.jks`, and `truststore.jks` files.
     iv. Edit `siebel-config/Chart.yaml` and increment chart version and commit the same.
   - Using terminal:
     i. SSH to SCM instance.
     ii. Enter the container:
     ```
     sudo podman exec -it cloudmanager -bash
     ```
     iii. Go to the `keystore` directory:
     ```
     cd <env_dir>/<Helm charts repository name>/siebel-config/keystore
     ```
     iv. Copy the custom `keystore.jks`, `keystore_client.jks`, and `truststore.jks` files to the `keystore` directory.
     v. Open `Chart.yaml` and increment the chart version:
     ```
     vi <env_dir>/<Helm charts repository name>/siebel-config/Chart.yaml
     ```
     vi. Commit and push the changes to the remote repository:

```
git pull
git add <file1> <file2>
git commit -m <message>
git push
```

2. Wait for 10 minutes so that flux will automatically reconcile and uptake above changes. Or you can manually reconcile using below commands:

```
flux reconcile source git siebel-repo
flux reconcile kustomization apps
```

The reconcile process might take up to 10 minutes. The new custom `keystore.jks`, `keystore_client.jks`, and `truststore.jks` files are pulled, and Kubernetes secret "keystore" is updated with new certificate values.

> **Note:** While the flux reconciliation is in progress, if you are monitoring your cluster by querying the status of the pod and helm release:
>
> - The `kubectl -n <namespace> get pods` command will return the status of the pod (`configure-xxxxx`) as `init:3/4`.
> - The helm -n <namespace> `ls -a` command will return the status of the Helm release (`siebel-config`) as `pending-upgrade`.
>
> The status of the pod `init:3/4` and Helm release `pending-upgrade` is as expected, you must proceed to step 3.

3. Execute the following commands to update Siebel Server/SAI/CGW containers with new certificates.
   a. Edit `<Helm charts repository name>/siebel/Chart.yaml`, increment chart version, and commit the same.
   b. Edit `<Helm charts repository name>/siebel-gateway/Chart.yaml`, increment chart version, and commit the same.

Use Cases for Rerunning the PostInstallDB Job

This topic describes the steps to rerun the `postInstallDB` job.

To rerun the `postInstallDB` job:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Connect to SCM using one of the following methods, depending on your SCM deployment:
   - SSH in to the SCM VM instance and exec in to the SCM container, or
   - Connect to the Kubernetes cluster where SCM is running and exec in to the SCM pod.
2. Go to the Helm charts directory:

```
cd /home/opc/siebel/<env_id>/<namespace>-helmcharts
```

3. Go to the `siebel` directory:

```
cd siebel
```

4. Increment the chart version in the `Chart.yaml` file. For example, increment the version from 0.1.0 to 0.1.1.
5. Commit the changes in the Git repository:

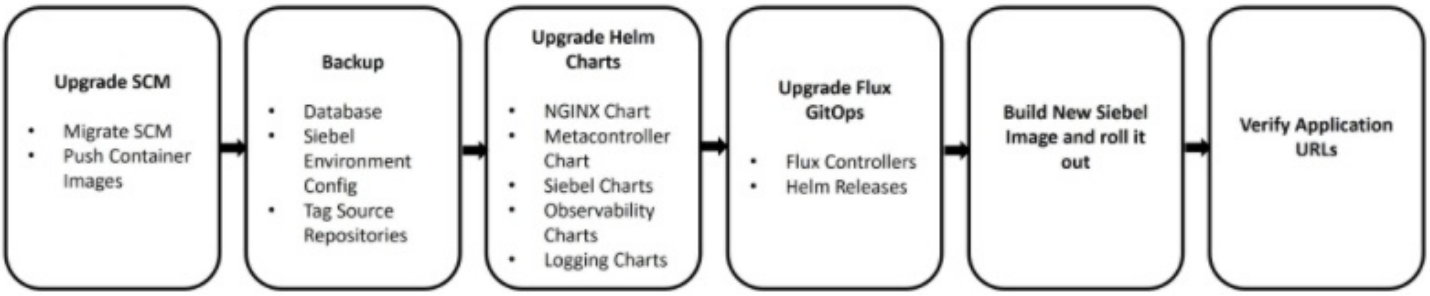
```
git status
git add Chart.yaml
git commit -m "siebel helm chart committed"
```

```
git push
```

Flux automatically upgrades the `siebel` Helm chart. The automatic Flux reconciliation process might take up to ten minutes. The `postinstallDB` job is triggered after the Flux reconciliation process completes.

High-Level Steps for Installing Siebel Monthly Update in a Siebel CRM Environment Deployed using SCM

You can use the procedure in this topic to install latest monthly updates in a Siebel CRM environment deployed using SCM. These steps do not include repository upgrade steps, which are optional and identical to those relevant for on-premises Siebel CRM deployments.



For detailed steps to install a Siebel monthly update, see *Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM*.

Enabling TLS 1.3 Support in Environments Prior to 23.11

Siebel CRM 23.11 provides the capacity to enable TLS 1.3 communication from client to server and server tier to server tier. Refer "Supported TLS Versions and RSA SHA" section in "Security Guide" of Siebel CRM bookshelf for more details.

In SCM deployed pre-23.11 environments, we need to take specific action to enable TLS 1.3 communication. At this stage, the recommended action will be to rename the 'conf' folder in applicationcontainer_internal and applicationcontainer_external to, say, conf_old. Make sure to bring down the Siebel CRM environment before renaming the 'conf' folder and restart at the end.

Stop the Siebel CRM Environment

```
sudo podman exec -it cloudmanager bash (Exec into the container)
source /home/opc/siebel/<ENV_ID>/k8sprofile

kubectl -n <namespace> get statefulset --> (Before bringing down the environment, note down the number of
 replicas of each statefulset)
kubectl -n <namespace> scale --replicas=0 statefulset/siebelcgw
kubectl -n <namespace> scale --replicas=0 statefulset/smc
kubectl -n <namespace> scale --replicas=0 statefulset/edge , where edge is the siebel server (bring down all
 other siebel servers if present)
kubectl -n <namespace> scale --replicas=0 statefulset/quantum, where quantum is the ai server (bring down
 all other ai servers if present)
exit
```

Enable TLS 1.3 Support in Pre-23.11 Environments

Rename the 'conf' folder to say 'conf_old' in below persistent paths:

- Exec into the SCM container
  ```
  sudo podman exec -it cloudmanager bash
  ```
- /home/opc/siebel/<ENV_ID>/<namespace>-siebfs*/<NAMESPACE>/CGW/siebelcgw-*/applicationcontainer_internal/conf where,
  - <namespace>-siebfs* denotes the siebel file system siebfs0,siebfs1, siebfs2 and so on.
  - siebelcgw-* denotes the cgw replicas siebelcgw-0, siebelcgw-1, siebelcgw-2 and so on.
- /home/opc/siebel/<ENV_ID>/<namespace>-siebfs*/<NAMESPACE>/SAI/smc-0/applicationcontainer_external/conf
- /home/opc/siebel/<ENV_ID>/<namespace>-siebfs*/<NAMESPACE>/edge/edge-0/applicationcontainer_internal/conf where,
  - edge,tibus and trust are the Siebel servers.
- /home/opc/siebel/<ENV_ID>/<namespace>-siebfs*/<NAMESPACE>/quantum/quantum-0/applicationcontainer_external/conf where,
  - quantum,alchemist and creative are the AI servers.

Bring up the Siebel CRM Environment

```
sudo podman exec -it cloudmanager bash (Exec into the container)
source /home/opc/siebel/<ENV_ID>/k8sprofile

kubectl -n <namespace> scale --replicas=3 statefulset/siebelcgw
kubectl -n <namespace> scale --replicas=1 statefulset/smc
kubectl -n <namespace> scale --replicas=1 statefulset/edge , where edge is the siebel server (bring down all
 other siebel servers if present)
kubectl -n <namespace> scale --replicas=1 statefulset/quantum, where quantum is the ai server (bring down
 all other ai servers if present)
kubectl -n <namespace> get pods (Verify the pods running status)
exit
```

Once the environment is up and running, any customizations made to server.xml have to be redone.

TLS 1.3 Support Verification

OCI Load balancer supports TLS versions till 1.2, so when the smc/application is accessed from the client, we would still see the request is served from TLS 1.2 connection protocol. For more information, refer *SSL Tunneling*.

But the Siebel AI requests from ingress are served by both TLS 1.2 & TLS 1.3 by default from 23.11. This can be verified from AI Tomcat's `server.xml` configuration.

```
sudo podman exec -it cloudmanager bash (Exec into the container)
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-siebfs0/<NAMESPACE>/quantum/quantum-0/
applicationcontainer_external/conf/
cat server.xml

<Connector port="4430" protocol="org.apache.coyote.http11.Http11NioProtocol"
compressableMimeType="text/css,text/javascript,application/x-javascript,application/javascript"
 useSendfile="off" compression="on" compressionMinSize="128" connectionTimeout="20000"
 noCompressionUserAgents="gozilla, traviata"
 maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
 SSLVerifyClient="require" SSLEngine="on" SSLVerifyDepth="2"
```

```
keystoreFile="/siebel/mde/applicationcontainer_external/siebelcerts/keystore.jks" keystorePass="siebel"
keystoreType="JKS"
truststoreFile="/siebel/mde/applicationcontainer_external/siebelcerts/truststore.jks"
truststorePass="siebel" truststoreType="JKS"

sslEnabledProtocols="TLSv1.2+TLSv1.3"

clientAuth="false"
relaxedQueryChars="&#x20;&#x22;&#x3C;&#x3E;&#x5B;&#x5C;&#x5D;&#x5E;&#x60;&#x7B;&#x7C;&#x7D;"
relaxedPathChars="&#x20;&#x22;&#x3C;&#x3E;&#x5B;&#x5C;&#x5D;&#x5E;&#x60;&#x7B;&#x7C;&#x7D;"
ciphers="TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_AES_128_GCM_SHA256, TLS_AES_256_GCM_SHA384,
TLS_CHACHA20_POLY1305_SHA256, TLS_AES_128_CCM_SHA256, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256, TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_CCM, TLS_ECDHE_ECDSA_WITH_AES_128_CCM"
```

Note: Notice the following line of code in the example above:

```
sslEnabledProtocols="TLSv1.2+TLSv1.3"
```

Rotating Secrets

This topic describes the procedure to rotate secrets (the registry access token, the Git token and the OCI configuration in the OCI configuration file). Rotation is the process of replacing existing sensitive information such as access token and encryption keys with new information. You must regularly rotate access tokens and encryption keys to prevent unauthorized access.

This topic contains the following sections:

- *Updating Registry Access Token*
- *Updating OCI Configuration in the OCI Configuration File*
- *Updating Git Access Token*
- *Updating Git CA Certificate*
- *Rotating the Load Balancer TLS Certificate for Ingress-Nginx*

Note: In the commands below, replace `<env_id>` with your environment ID and `<namespace>` with the environment name that was passed in the payload for Siebel CRM environment provisioning.

Updating Registry Access Token

Registry access token is used to access the container registry. You must rotate the container registry access token, configured through the `registry_password` parameter in the `siebel-config.yaml` file, regularly to prevent unauthorized access to the images in the container registry.

To update the registry access token, perform the following tasks:

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Update the `registry_password` parameter in the Git repository:
   a. Go to the environment directory:
      ```
      cd /home/opc/siebel/<env_id>
      ```
   b. Open the `siebel-config.yaml` file:
      ```
      vi <Cloud manager repository name>/flux-crm/apps/base/siebel/siebel-config.yaml
      ```
   c. Update the `registry_password` parameter value with the new access token.
2. Recreate the custom secret definition for the registry credentials:
   a. Delete the existing custom secret definition:
      ```
      source /home/opc/siebel/<env_id>/k8sprofile
      kubectl delete secret -n <namespace> customsecret
      ```
   b. Go to the `secrets` directory:
      ```
      cd /home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/infrastructure/secrets
      ```
   c. Create a new custom secret definition with the updated registry access token:
      ```
      kubectl --dry-run=client -n <namespace> create secret docker-registry customsecret \
      --docker-server=<registry_url> \
      --docker-username=<registry_username>\
      --docker-password=<registry_password> \
      --docker-email=siebel@oracle.com \
      -o yaml > customsecret.yaml
      ```
3. Commit the changes to the remote Git repository:
   ```
   git add .
   git commit -m "updated registry password and custom secrets"
   git config pull.rebase false
   git pull
   git push
   ```

   Note: Flux will reconcile and uptake the new changes in the Git repository and recreate a new custom secret.
4. Update `registry_password` in environment YAML file:
   a. Open the environment YAML file:
      ```
      vi /home/opc/siebel/environments/<env_id>_environment.yaml
      ```
   b. Update the `registry_password` parameter value with the new access token.

Updating OCI Configuration in the OCI Configuration File

The OCI configuration file (`~/.oci/config`) includes user credentials information such as `user, fingerprint, key_file` and so on that's used to authenticate users when they access OCI resources. You must update the OCI configuration information in the configuration file regularly to ensure only authorized users can access the OCI resources.

To update the OCI configuration in the configuration file, perform the following tasks:

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Update the OCI configuration in the SCM container:
   a. Update the private key file path, fingerprint and so on, in the OCI configuration file (`~/.oci/config`).
   b. Update the new private key in the `~/.oci/user.pem` file.

2. If you've enabled observability (that's, if you've set the value of the parameters `siebel_monitoring` and `enable_oci_monitoring` to `true`), update the OCI configuration in Helm chart repository:

   Update the Prometheus Helm chart:

   a. Go to the `prometheus` Helm chart directory:
      ```
      cd /home/opc/siebel/<Helm charts repository name>/prometheus
      ```
   b. Update the fingerprint and so on, in the `oci_config/config` file.
   c. Update the new private key in the `oci_api_key.pem` file.
   d. Increment the chart version in the `Chart.yaml` file.
   e. Commit the changes in the Git repository:
      ```
      git add .
      git commit -m "Updated OCI Configuration"
      git push
      ```

3. If you've enabled logging (that's if you've set the value of the parameters `siebel_logging` and `enable_oci_log_analytics` to `true`), update the OCI configuration in the Siebel logging Helm chart:
   a. Go to the `siebel-logging` directory:
      ```
      cd /home/opc/siebel/<Helm charts repository name>/siebel-logging
      ```
   b. Update the fingerprint and so on, in the `oci_config/config` file.
   c. Update the new private key in the `oci_api_key.pem` file.
   d. Increment chart version in the `Chart.yaml` file.
   e. Commit the changes in the Git repository:
      ```
      git add .
      git commit -m "Updated OCI Configuration"
      git push
      ```

4. Perform the flux reconcile manually:
   ```
   source /home/opc/siebel/<env_id>/k8sprofile
   flux reconcile source git siebel-repo -n <namespace>
   flux reconcile kustomization apps -n <namespace>
   ```

5. Restart the pods:
   ```
   kubectl rollout restart deploy prometheus-adapter-deployment -n <namespace>
   kubectl rollout restart deploy log-aggregator -n <namespace>
   ```

6. Verify the Prometheus adapter deployment pod logs and log aggregator pod logs to ensure that you aren't receiving the "401 Not Authenticated" error.

Updating Git Access Token

Git access token allows only authorized users to access the Git repository. You must update the Git access token regularly to ensure only authorized users can access the Git repository.

To update the Git access token, perform the following tasks:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Delete the secret flux-system:
   ```
   source /home/opc/siebel/<env_id>/k8sprofile
   ```

```
kubectl -n <namespace> delete secret flux-system
```

2. Set the `GIT_PASSWORD` environment variable:
   ```
   export GIT_PASSWORD=<git_accesstoken>
   ```
3. Execute the flux bootstrap with the new token:
   ```
   flux.sh bootstrap git --components-extra=image-reflector-controller,image-automation-controller --
   url=https://<GIT_HOSTNAME>/<GIT_USER>/<Cloud manager repository> --namespace=<namespace> --branch=master
    --path='flux-crm/clusters/staging' --log-level debug --watch-all-namespaces=false --image-pull-secret
    ocirsecret --registry <CONTAINER_REGISTRY_URL>/fluxcd --username=<GIT_USER> --silent --token-auth --ca-
   file=<GIT_SELFSIGNED_CACERT_PATH>
   ```
4. Update the Git access token in the SCM Git repository:
   a. Go to the environment directory:
      ```
      cd /home/opc/siebel/<env_id>
      ```
   b. Open the `siebel-config.yaml` file:
      ```
      vi <Cloud manager repository name>/flux-crm/apps/base/siebel/siebel-config.yaml
      ```
   c. Update the value of the `git_accesstoken` parameter.
5. Commit the changes to the remote Git repository:
   a. Go to the SCM Git repository:
      ```
      cd /home/opc/siebel/<env_id>/<Cloud manager repository name>
      ```
   b. Commit the changes to the Git repository:
      ```
      git add .
      git commit -m "updated git access token and custom secrets"
      git config pull.rebase false
      git pull
      git push
      ```
   c. Perform flux reconcile manually:
      ```
      flux reconcile source git <namespace> -n <namespace>
      flux reconcile kustomization apps -n <namespace>
      ```

Updating Git CA Certificate

A Git CA certificate is used to verify the authenticity of SSL/TLS connections when interacting with remote Git repositories over HTTPS. You must update the Git CA certificate regularly to ensure secure communication by validating the identity of the server.

To update the Git CA certificate, perform the following tasks:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Go to the `certs` directory in the SCM instance:
   ```
   sudo podman exec -it cloudmanager bash
   cd /home/opc/certs
   ```
2. Copy the certificate (new `rootCA.crt` file) to the `certs` directory.
3. Copy the certificate to the `/etc/pki/ca-trust/source/anchors` directory:
   ```
   sudo cp rootCA.crt /etc/pki/ca-trust/source/anchors/
   ```
4. Add the certificate to the trusted certificates:

```
sudo /bin/update-ca-trust
```

5. Delete the secret `flux-system`:

```
source /home/opc/siebel/<env_id>/k8sprofile
kubectl -n <namespace> delete secret flux-system
```

6. Set the `GIT_PASSWORD` environment variables:

```
export GIT_PASSWORD=<git_accesstoken>
```

7. Execute the flux bootstrap with the certificate:

```
flux.sh bootstrap git --components-extra=image-reflector-controller,image-automation-controller --
url=https://<GIT_HOSTNAME>/<GIT_USER>/<Cloud manager repository> --namespace=<NAMESPACE> --branch=master
 --path='flux-crm/clusters/staging' --log-level debug --watch-all-namespaces=false --image-pull-secret
 ocirsecret --registry <CONTAINER_REGISTRY_URL>/fluxcd --username=<GIT_USER> --silent --token-auth --ca-
file=<GIT_SELFSIGNED_CACERT_PATH>
```

8. Verify the health of the components in the flux bootstrap response.
9. Update the certificate in the Git repositories:
   a. Update in the SCM repository as follows:

```
cd /home/opc/siebel/<env_id>/<Cloud manager repository name>
git config --local http.sslCAInfo <CA Certificate Path>
```

   b. Update in the Helm charts repository as follows:

```
cd /home/opc/siebel/<env_id>/<Helm chart repository name>
git config --local http.sslCAInfo <CA Certificate Path>
```

Rotating the Load Balancer TLS Certificate for Ingress-Nginx

You can replace the public load balancer TLS certificate for an existing Siebel CRM environment that uses ingress-nginx in SCM 26.2 or earlier. This procedure updates the certificate without recreating the Kubernetes Service or OCI Load Balancer. The existing external IP address remains unchanged.

Note: This procedure applies only to SCM 26.2 or earlier environments that use ingress-nginx.

This topic includes the following sections:

- *Prerequisites*
- *Rotating the Load Balancer TLS Certificate*
- *Rolling Back the Certificate Change*

Prerequisites

Before you begin, make sure that you have:

- An administrative shell with the correct `kubeconfig` and access to the generated SCM Git repository.
- The environment namespace, environment ID, public DNS name, and generated repository path.
- The replacement TLS certificate and matching private key.
- Access to the existing OCI Load Balancer details.
- An approved timeout for the certificate change and a rollback retention period.

Note: Use this procedure only when ingress-nginx manages the environment. If both NGINX and Traefik resources exist, confirm the appropriate procedure for the environment's migration state.

Rotating the Load Balancer TLS Certificate

To rotate the OCI Load Balancer TLS certificate for ingress-nginx:

1. Set the following environment variables:

```
export ROTATE_NS='<siebel-namespace>'
export ROTATE_ENV_ID='<scm-environment-id>'
export ROTATE_HOST='<public-dns-name>'
export CERT_DIR='<tls-certificate-directory>'
export ROTATE_CERT="$CERT_DIR/tls.crt"
export ROTATE_KEY="$CERT_DIR/tls.key"
export ROTATE_NEW_SECRET="lb-tls-certificate-$(date -u +%Y%m%d%H%M%S)"
export ROTATE_REPO='/home/opc/siebel/<env-id>/<scm-repo>'
export ROTATE_FLUX_FOLDER='flux-crm'
```

   In the above commands:

   - Replace the placeholders with the values for your deployment.
   - Set `ROTATE_NEW_SECRET` to a new, versioned Kubernetes Secret name. Do not reuse the current secret name.

2. Use `ROTATE_FLUX_FOLDER` from step 1 to derive the configuration paths:

```
export ROTATE_NGINX_REL="${ROTATE_FLUX_FOLDER}/infrastructure/nginx"
export ROTATE_SECRET_REL="${ROTATE_FLUX_FOLDER}/infrastructure/${ROTATE_NS}/oci-lb-ssl-certificate-
secret.yaml"
export ROTATE_NGINX_DIR="${ROTATE_REPO}/${ROTATE_NGINX_REL}"
export ROTATE_SECRET_FILE="${ROTATE_REPO}/${ROTATE_SECRET_REL}"
```

3. Verify the cluster and reconciliation state:

```
kubectl config current-context
kubectl -n "$ROTATE_NS" get helmrelease.helm.toolkit.fluxcd.io nginx
kubectl -n "$ROTATE_NS" get kustomization.kustomize.toolkit.fluxcd.io infrastructure
kubectl -n "$ROTATE_NS" get helmrelease.helm.toolkit.fluxcd.io traefik
```

   Verify that:

   - The NGINX Helm release and infrastructure Kustomization are ready.
   - The Traefik lookup returns `NotFound`.

4. Derive the Helm release and service names:

```
export ROTATE_RELEASE="$(kubectl -n "$ROTATE_NS" get helmrelease.helm.toolkit.fluxcd.io nginx -o
 jsonpath='{.spec.releaseName}')"
export ROTATE_SVC="${ROTATE_RELEASE}-controller"
kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o wide
```

   For an SCM 26.2 deployment with the default configuration, the Helm release name is `<namespace>-ingress-nginx`, and the Kubernetes Service name is `<namespace>-ingress-nginx-controller`.

5. Verify the existing TLS configuration:

   a. Read the current secret name from the service annotation:

```
export ROTATE_OLD_SECRET="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o go-
template='{{index .metadata.annotations "service.beta.kubernetes.io/oci-load-balancer-tls-
secret"}}')"
```

```
kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o go-template='type={{.spec.type}}
{{"\n"}}sslPorts={{index .metadata.annotations "service.beta.kubernetes.io/oci-load-balancer-
ssl-ports"}}{{"\n"}}tlsSecret={{index .metadata.annotations "service.beta.kubernetes.io/oci-load-
balancer-tls-secret"}}{{"\n"}}'
kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o jsonpath='{range .spec.ports[*]}{.name}{"="}
{.port}{"->"}{.targetPort}{"\n"}{end}'
kubectl -n "$ROTATE_NS" get secret "$ROTATE_OLD_SECRET" -o jsonpath='{.type}{"\n"}'
```

You must continue only when:

- The service type is `LoadBalancer`.
- The SSL ports include `443`.
- The TLS Secret annotation contains a secret name.
- The HTTPS Service port targets NGINX HTTP.
- The existing secret type is `kubernetes.io/tls`.
- The new secret lookup returns `NotFound`.

b. Confirm that the proposed name is unused:

```
kubectl -n "$ROTATE_NS" get secret "$ROTATE_NEW_SECRET"
```

If the proposed secret already exists, choose another name.

6. Record the following:

a. The existing Kubernetes Service and OCI Load Balancer details:

```
export ROTATE_UID_BEFORE="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.metadata.uid}')"
export ROTATE_CLUSTER_IP_BEFORE="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.spec.clusterIP}')"
export ROTATE_LB_ADDR_BEFORE="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.status.loadBalancer.ingress[0].ip}{.status.loadBalancer.ingress[0].hostname}')"

printf 'serviceUID=%s\nclusterIP=%s\nexternalAddress=%s\n' \
 "$ROTATE_UID_BEFORE" "$ROTATE_CLUSTER_IP_BEFORE" "$ROTATE_LB_ADDR_BEFORE"
```

b. The OCI Load Balancer OCID from OCI Console or OCI CLI. These values must remain unchanged after rotation.

c. The current certificate fingerprint, from a host that can reach the endpoint:

```
openssl s_client -connect "${ROTATE_HOST}:443" \
 -servername "$ROTATE_HOST" </dev/null 2>/dev/null |
 openssl x509 -noout -fingerprint -sha256 -subject -issuer -serial -dates
```

d. The expected HTTP status or authentication response for each enabled application, SMC, Prometheus, and OpenSearch Dashboards URL.

7. Validate the replacement certificate and private key:

```
openssl x509 -in "$ROTATE_CERT" -noout \
 -fingerprint -sha256 -subject -issuer -serial -dates -ext subjectAltName
openssl x509 -in "$ROTATE_CERT" -noout -checkend 0
openssl pkey -in "$ROTATE_KEY" -check -noout
```

8. Compare the public-key hashes. Both commands must return the same hash:

```
openssl x509 -in "$ROTATE_CERT" -pubkey -noout |
 openssl pkey -pubin -outform DER 2>/dev/null |
 openssl dgst -sha256
```

```
openssl pkey -in "$ROTATE_KEY" -pubout -outform DER 2>/dev/null |
 openssl dgst -sha256
```

Verify that:

- The Subject Alternative Name (SAN) covers every externally exposed host name.
- The certificate is currently valid and meets your organization's remaining-validity requirements.
- The certificate file contains the leaf certificate first, followed by the required intermediate certificates.
- Only the operator can read the private key and temporary files.

If the key is encrypted:

a. Create a temporary directory with mode 0700, set umask 077, and decrypt the key interactively with `openssl pkey`. Do not put the password in a command-line argument or shell history.
b. Set `ROTATE_KEY` to the path of the protected, decrypted private-key file.
c. Remove the file after you complete verification and meet the rollback retention requirements.

9. Synchronize the generated Git repository:

```
cd "$ROTATE_REPO"
git status --short
export ROTATE_BRANCH="$(git branch --show-current)"
git fetch origin "$ROTATE_BRANCH"
git pull --ff-only origin "$ROTATE_BRANCH"
export ROTATE_PARENT="$(git rev-parse HEAD)"
```

If the working copy contains changes, the branch name is empty, or the pull cannot fast-forward, resolve the issue before continuing. Verify that the revision applied by Flux matches the recorded parent commit:

```
kubectl -n "$ROTATE_NS" get \
 kustomization.kustomize.toolkit.fluxcd.io infrastructure \
 -o jsonpath='{.status.lastAppliedRevision}{"\n"}'
```

10. Prepare the replacement secret manifest. Create a protected temporary directory and generate the manifest without applying it:

```
umask 077
export ROTATE_TMP="$(mktemp -d /tmp/scm-262-cert-rotation.XXXXXX)"
chmod 700 "$ROTATE_TMP"

kubectl -n "$ROTATE_NS" create secret tls "$ROTATE_NEW_SECRET" \
--cert="$ROTATE_CERT" --key="$ROTATE_KEY" \
--dry-run=client -o yaml > "$ROTATE_TMP/new-secret.yaml"

export ROTATE_ENV_LABEL="$(kubectl -n "$ROTATE_NS" get secret "$ROTATE_OLD_SECRET" -o
 jsonpath='{.metadata.labels.oracle\.siebel\.namespace-envid}')"
test -n "$ROTATE_ENV_LABEL"
```

If `ROTATE_ENV_LABEL` is empty, verify that `ROTATE_NS` specifies the correct namespace and `ROTATE_OLD_SECRET` specifies the existing TLS Secret in that namespace. Make sure that the secret has the `oracle.siebel.namespace-envid` label. After resolving the issue, retrieve the label again. Continue only when `ROTATE_ENV_LABEL` contains the correct environment label value.

```
kubectl label --local -f "$ROTATE_TMP/new-secret.yaml" \
"oracle.siebel.namespace-envid=$ROTATE_ENV_LABEL" \
```

```
--overwrite -o yaml > "$ROTATE_TMP/new-secret-labelled.yaml"
```

Note: The manifest contains private-key data. Do not print it, display its diff, or attach it to a support ticket.

11. Update and commit the generated configuration:
    a. Use a YAML-aware editor to make the following changes:
       - In `${ROTATE_NGINX_DIR}/release.yaml`, update `spec.values.controller.service.annotations["service.beta.kubernetes.io/oci-load-balancer-tls-secret"]` to the new secret name. Preserve all other annotations.
       - In each existing `${ROTATE_NGINX_DIR}/siebel-ingress-*.yaml` file, replace references to the old secret under `spec.tls[].secretName` with the new secret. Include application, SMC, and enabled observability ingress definitions.
       - Replace `${ROTATE_SECRET_FILE}` with `${ROTATE_TMP}/new-secret-labelled.yaml`.

       SCM stores the generated secret manifest in its private repository. Flux does not apply manifests in this namespace directory during normal reconciliation, so you must create the secret in the cluster in the next step.
    b. Review the non-secret changes and affected filenames:

```
git diff -- "$ROTATE_NGINX_REL/release.yaml" \
 "$ROTATE_NGINX_REL"/siebel-ingress-*.yaml
git diff --stat
git diff --check
git status --short
```

    c. Verify that no active NGINX reference points to the old secret. Commit the changes locally:

```
git add "$ROTATE_NGINX_REL" "$ROTATE_SECRET_REL"
git diff --cached --check
git diff --cached --name-only
git commit -m "Rotate ingress TLS certificate to ${ROTATE_NEW_SECRET}"
export ROTATE_COMMIT="$(git rev-parse HEAD)"
```

       Note: Do not push the commit yet.

12. Create and verify the new secret:
    a. Create new secret:

```
kubectl apply -f "$ROTATE_TMP/new-secret-labelled.yaml"
kubectl -n "$ROTATE_NS" get secret "$ROTATE_NEW_SECRET" \
 -o jsonpath='{.metadata.uid}{" "}{.type}{"\n"}'
```

    b. Verify that the fingerprint matches the replacement certificate. Retrieve only the public certificate; do not retrieve `.data.tls.key`.

```
kubectl -n "$ROTATE_NS" get secret "$ROTATE_NEW_SECRET" \
 -o jsonpath='{.data.tls\.crt}' |
 openssl base64 -d -A |
 openssl x509 -noout -fingerprint -sha256 -subject -issuer -dates
```

    c. Verify that the fingerprint matches the input certificate. At this stage, no resource references the new secret, so existing traffic remains unchanged.
13. Publish the commit and reconcile the existing service.
    a. Verify that the remote branch still matches the recorded parent:

```
git fetch origin "$ROTATE_BRANCH"
test "$(git rev-parse "origin/$ROTATE_BRANCH")" = "$ROTATE_PARENT"
```

If the comparison fails, resolve the Git changes before continuing. Do not force-push or patch the Service directly.

b. Publish the prepared commit:

```
git push origin "HEAD:$ROTATE_BRANCH"
```

c. Allow Flux and Helm to update the existing service. Do not suspend Flux or replace the service:

```
flux reconcile kustomization infrastructure -n "$ROTATE_NS" \
 --with-source --timeout=30m
flux reconcile helmrelease nginx -n "$ROTATE_NS" --timeout=30m
kubectl -n "$ROTATE_NS" wait \
 helmrelease.helm.toolkit.fluxcd.io/nginx \
 --for=condition=Ready --timeout=30m
```

14. Verify that the endpoint serves the replacement certificate:

```
kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" \
 -o go-template='{{index .metadata.annotations "service.beta.kubernetes.io/oci-load-balancer-tls-
secret"}}{{"\n"}}'

kubectl -n "$ROTATE_NS" get events \
 --field-selector "involvedObject.kind=Service,involvedObject.name=$ROTATE_SVC" \
 --sort-by=.lastTimestamp
```

The OCI Cloud Controller Manager updates the existing load balancer listener asynchronously. Check the served fingerprint at controlled intervals until it matches the replacement certificate:

```
openssl s_client -connect "${ROTATE_HOST}:443" \
 -servername "$ROTATE_HOST" </dev/null 2>/dev/null |
 openssl x509 -noout -fingerprint -sha256 -subject -issuer -serial -dates
```

If the approved timeout expires, perform the rollback procedure. Do not delete the service or change only the listener in OCI Console. For more information, see *Rolling Back the Certificate Change*.

15. Verify that the service and load balancer remain unchanged:

```
export ROTATE_UID_AFTER="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.metadata.uid}')"
export ROTATE_CLUSTER_IP_AFTER="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.spec.clusterIP}')"
export ROTATE_LB_ADDR_AFTER="$(kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" -o
 jsonpath='{.status.loadBalancer.ingress[0].ip}{.status.loadBalancer.ingress[0].hostname}')"

test "$ROTATE_UID_AFTER" = "$ROTATE_UID_BEFORE"
test "$ROTATE_CLUSTER_IP_AFTER" = "$ROTATE_CLUSTER_IP_BEFORE"
test "$ROTATE_LB_ADDR_AFTER" = "$ROTATE_LB_ADDR_BEFORE"
```

Verify that the OCI Load Balancer OCID also matches the recorded value. Check the enabled URLs against their recorded HTTP status or authentication responses.

Rolling Back the Certificate Change

If rotation fails, you must restore the previous configuration. The old secret remains available because the procedure does not modify it.

To roll back the certificate change:

1. Revert the rotation commit. From a clean working copy on the same branch, run:

```
cd "$ROTATE_REPO"
git status --short
git revert --no-edit "$ROTATE_COMMIT"
git push origin "HEAD:$ROTATE_BRANCH"
```

2. Reconcile the previous configuration:

```
flux reconcile kustomization infrastructure -n "$ROTATE_NS" \
 --with-source --timeout=30m
flux reconcile helmrelease nginx -n "$ROTATE_NS" --timeout=30m
kubectl -n "$ROTATE_NS" wait \
 helmrelease.helm.toolkit.fluxcd.io/nginx \
 --for=condition=Ready --timeout=30m
```

3. Verify that the endpoint uses the previous certificate:

```
kubectl -n "$ROTATE_NS" get service "$ROTATE_SVC" \
 -o go-template='{{index .metadata.annotations "service.beta.kubernetes.io/oci-load-balancer-tls-
secret"}}{{"\n"}}'

openssl s_client -connect "${ROTATE_HOST}:443" \
 -servername "$ROTATE_HOST" </dev/null 2>/dev/null |
 openssl x509 -noout -fingerprint -sha256 -subject -issuer -serial -dates
```

   Verify that the annotation names the old secret and the served fingerprint matches the previous certificate.
4. Verify the environment after rollback.

   Confirm that the Service UID, ClusterIP, external address, and OCI Load Balancer OCID match the values recorded before rotation. If you also updated protected SCM environment metadata, restore its previous values.

   Record the operation as rolled back. Retain the new secret for investigation until cleanup is approved and a reference check confirms that no resource uses it.

> **Note:** Remove protected temporary files and any decrypted private-key copy after verification and rollback retention requirements are satisfied.

Assigning Pods to Nodes - Implementing Affinity and Anti-affinity on OKE using Siebel Cloud Manager

Using SCM, you can constrain a pod so that it is restricted to run on particular node(s) or to prefer to run on particular nodes. There are several ways to do this and the recommended approaches all use label selectors to facilitate the selection. Affinity definitions are available in Kubernetes API reference. These can be added as a customization in configuration.

This topic has the following sections:

- *Customizing the Configuration with Affinity*
- *Use Cases for Making Incremental Changes*

Customizing the Configuration with Affinity

Affinity changes will go as a customization that require changes in the SCM repository. Affinity can be defined for the following Siebel pods:

- Siebel Server pods (edge, tibus and so on)
- SAI Server pods (quantum, alchemist and so on)
- cgw pod
- smc pod

To add affinity:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. SSH into the SCM instance.
2. Enter commands like the following:
   ```
   sudo podman exec -it cloudmanager bash
   ```
3. Override the configuration in different Siebel CR pods:
   a. Edit the `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel/siebel.yaml` file to add "affinity" for each Siebel server/SAI server as:

   ```
   - sesServer:
   - profile: sieb_server_profile1
   replicas: 1
   siebsrvr_prefix: tibus
   affinity:
   podAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
   - labelSelector:
   matchExpressions:
   - key: app.siebel.tier
   operator: In
   values:
   - edge
   topologyKey: kubernetes.io/hostname
   weight: 100

   - saiServer:
   - profile: ai_automotive_greenfield
   replicas: 1
   sai_prefix: quantum
   affinity:
   nodeAffinity:
   preferredDuringSchedulingIgnoredDuringExecution:
   - weight: 1
   preference:
   matchExpressions:
   - key: topology.kubernetes.io/zone
   operator: In
   values:
   - UK-LONDON-1-AD-1
   ```

   b. Edit the `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel/siebel-gateway.yaml` file to add "affinity" for siebel-gateway (CGW) pods:

   ```
   cgw:
   ```

```
replicas: 3
affinity:
podAntiAffinity:
preferredDuringSchedulingIgnoredDuringExecution:
- weight: 100
podAffinityTerm:
labelSelector:
matchExpressions:
- key: app.siebel.tier
operator: In
values:
- cgw
topologyKey: "kubernetes.io/hostname"
```

c. Sameway, affinity can also be added to SMC pod by editing /home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel/siebel-gateway.yaml:

```
smc:
 affinity:{} // affinity definition for smc pod goes here
```

4. Commit your customization in the SCM Git repository. Make sure to add all modified files. The above changes will be included in the initial environment provisioning, where you specify the configuration ID.
5. Check the status of a requested configuration. For more information, see *Checking the Status of a Requested Configuration*.
6. Deploy the environment with the customized configuration. In this step you specify only the configuration ID and the deployment name. For more information, see *About this Chapter*.

Use Cases for Making Incremental Changes

Here are some use cases for adding affinity definitions to individual Siebel pods.

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Adding affinity for individual Siebel Server

1. Exec into the SCM container:

```
sudo podman exec -it cloudmanager bash
```

2. Edit: `siebel.yaml` under `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel`

```
 sesServer:
 - profile: sieb_server_profile1
 replicas: 1
 siebsrvr_prefix: edge
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: topology.kubernetes.io/zone
 operator: In
 values:
 - UK-LONDON-1-AD-2
 preferredDuringSchedulingIgnoredDuringExecution:
 - weight: 1
 preference:
 matchExpressions:
 - key: another-node-label-key
 operator: In
```

```
values:
- another-node-label-value
```

3. Run the following commands:

```
git pull
git add .
git commit -m "<message>"
git push
```

Adding affinity for individual Sai Server

1. Edit: `siebel.yaml` under `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel`

```
saiServer:
- profile: application_interface_profile1
replicas: 1
sai_prefix: quantum
affinity:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
- key: topology.kubernetes.io/zone
operator: In
values:
- UK-LONDON-1-AD-1
```

2. Run the following commands:

```
git pull
git add .
git commit -m "<message>"
git push
```

Adding affinity for CGW pods

1. Edit: `siebel-gateway.yaml` under `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel`

```
cgw:
replicas: 3
affinity: {} // affinity definition for cgw pod goes here.
```

2. Run the following commands:

```
git pull
git add .
git commit -m "<message>"
git push
```

Adding affinity for SMC pod

1. Edit: `siebel-gateway.yaml` under `/home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/apps/base/siebel`

```
smc:
affinity: {} // affinity definition for smc pod goes here.
```

2. Run the following commands:

```
git pull
git add .
git commit -m "<message>"
git push
```

Adding and Overriding Environment Variables

This topic describes the steps to add new environment variables and override existing environment variables in individual Siebel pods. You can add environment variables and override the existing environment variables either by customizing the configuration or by making incremental changes.

This topic has the following sections:

- *Adding and Overriding Environment Variables by Customizing the Configuration*
- *Adding and Overriding Environment Variables by Making Incremental Changes*

Adding and Overriding Environment Variables by Customizing the Configuration

You can use customization to add new environment variables and also override existing environment variables in the following Siebel pods:

- Siebel Server pods such as edge, tibus and so on
- SAI pods such as quantum, alchemist and so on
- CGW pods
- SMC pods

To add or override the environment variables:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. SSH into the SCM instance:
   `sudo podman exec -it cloudmanager bash`
2. Go to the `/home/opc/siebel/<ENV_ID>/<Cloud manager repository name>/flux-crm/apps/base/siebel` directory.
3. Add the `envlist` parameter to the Siebel pod sections as required in the files specified below:

Siebel Pod	Sample Data

Siebel server in the `siebel.yaml` file.	`siebelServer:`
	`- profile: siebel`
	`  replicas: 1`
	`  sesResources:`
	`  limits:`
	`  cpu: 4`
	`  memory: 24Gi`
	`  requests:`
	`  cpu: 1`
	`  memory: 8Gi`
	`  siebsrvr_prefix: edge`
	`  envlist:`
	`  - name: SBL_HEAP_OPTS`
	`  value: "-Xms700m -Xmx7G"`

Siebel Pod	Sample Data
	- name: SIEBEL_LOG_EVENTS value: "2"
SAI server in the **siebel.yaml** file.	saiServer: - profile: sai_profile replicas: 1 saiResources: limits: cpu: 4 memory: 24Gi requests: cpu: 1 memory: 8Gi saisrvr_prefix: quantum envlist: - name: SBL_HEAP_OPTS value: "-Xms700m -Xmx7G" - name: SIEBEL_LOG_EVENTS value: "2"
CGW pod in the **siebel-gateway.yaml** file.	cgw: replicas: 3 cgwResources: limits: cpu: 4 memory: 24Gi requests: cpu: 1 memory: 8Gi envlist: - name: SIEBEL_LOG_EVENTS value: "3"
SMC pod in the **siebel-gateway.yaml** file.	smc: replicas: 1 envlist: - name: SIEBEL_LOG_EVENTS value: "1"

4. Commit the customization in the SCM Git repository as follows:

```
git pull
git add .
git commit -m "<message>"
git push
```

> **Note:** Ensure that you include all the modified files, as these changes will be included in the initial environment provisioning that is based on this configuration ID.

5. Check the status of a requested configuration. For more information, see *Checking the Status of a Requested Configuration*.
6. Deploy the environment with the customized configuration specifying the configuration ID and the deployment name. For more information, see *About this Chapter*.

Adding and Overriding Environment Variables by Making Incremental Changes

You can add environment variables and override environment variables in individual Siebel pods after Siebel deployment by making incremental changes.

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

To add or override environment variables for Siebel server:

1. SSH in to the SCM container.
2. Edit the `siebel.yaml` file under the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory to add the `envlist` parameter, example as follows:

```
siebelServer:
     - profile: siebel
     replicas: 1
     ...
     siebsrvr_prefix: edge
     envlist:
          - name: SBL_HEAP_OPTS
               value: "-Xms700m -Xmx7G"
          - name: testContainerMode
               value: "sesTest"
```

3. Commit the changes in the SCM Git repository as follows:

```
git pull
git add .
git commit -m "<message>"
git push
```

To add or override environment variables for SAI server:

1. SSH in to the SCM container.
2. Edit the `siebel.yaml` file under the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory to add the `envlist` parameter, example as follows:

```
saiServer:
     - profile: siebel
          replicas: 1
          ...
          saisrvr_prefix: quantum
          envlist:
               - name: SBL_HEAP_OPTS
                    value: "-Xms700m -Xmx7G"
               - name: testContainerMode
                    value: "saiTest"
```

3. Commit the changes in the SCM Git repository as follows:

```
git pull
git add .
git commit -m "<message>"
git push
```

To add or override environment variables for CGW pod:

1. SSH in to the SCM container.

2. Edit the `siebel-gateway.yaml` file under the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory to add the `envlist` parameter, example as follows:

```
cgw:
     replicas: 3
     ...
     envlist:
          - name: SIEBEL_LOG_EVENTS
               value: "3"
```

3. Commit the changes in the SCM Git repository as follows:

```
git pull
git add .
git commit -m "<message>"
git push
```

To add or override environment variables for SMC pod:

1. SSH in to the SCM container.
2. Edit the `siebel-gateway.yaml` file under the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory to add the `envlist` parameter, example as follows:

```
smc:
     replicas: 1
     envlist:
          - name: SIEBEL_LOG_EVENTS
               value: "1"
```

3.
```
git pull
git add .
git commit -m "<message>"
git push
```

Cleaning up the Siebel File System

This topic describes how to clean up the Siebel File System by removing orphan records using an API. Orphan records are those that remain if a user deletes a parent record in the Siebel CRM application that has associated child records. The child records are not deleted from the Siebel File System with the parent record and so you must remove them by executing this API.

This API builds on the functionalities provided by sfscleanup service available with Siebel CRM installations and described in details in Siebel CRM System Administration Guide in Siebel Bookshelf. Executing the API will process records for every file in the file attachment directories (the att subdirectories) of the specified Siebel File System directories and performs one of several available operations to each record and file, depending on the file type and on the parameters that you set. For descriptions of the run-time parameters that you can set, refer to the payload parameters. The API call will trigger the file system cleanup job and after the clean up is done, reports will be generated.

Execute the clean up by calling POST API as follows:

POST endpoint:

```
https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/{ENV_ID}/sfscleanup
```

Sample payload:

```
{
 "discarded_files_path": "/some/path",
 "remove_old_revisions": true,
 "generate_report_only": true,
```

```
    "append_att_directory_path": true,
    "remove_non_siebel_files": true,
    "query_by_attachment_file": true,
    "no_of_file_id_per_query": "10",
    "use_or_clause_for_file_id": true,
    "run_for_specified_minutes": "10",
    "file_system_name": "namespace-siebfs0"
  }
```

Note: Specify payload parameters suitable for your circumstances by referring to the following Payload Parameters for File System Cleanup table:

Payload Parameter	Section	Definition
generate_report_only	Top Level	(required) If set to false, the service cleans up the specified file system else only generates a report without any cleanup actions.
remove_old_revisions	Top Level	Determines whether old versions of file attachments are to be removed. To remove old versions, set this value to true.
move_discarded_files	Top Level	Set this value to the move the discarded files. Files can be found at the location inside the Siebel Cloud Manager container - **`/home/opc/siebel/{ENV_ID}/{namespace}-siebfs{index}/{namespace}/SFSUTILS/SFSCLEANUP/discarded_files/{RUN_ID}`**
append_att_directory_path	Top Level	Set this value to true If you want the service to automatically append att to each directory.
remove_non_siebel_files	Top Level	Set this value to remove garbage files or non-Siebel files.
query_by_attachment_file	Top Level	Set this value to perform query by attachment files records.
no_of_file_id_per_query	Top Level	Set this value to the number of file attachment records to query.
use_or_clause_for_file_id	Top Level	Set this value when the service needs to use an OR clause to constrain the query row IDs, like this: (ROW_ID = 'Id1' OR ROW_ID = 'Id2' OR ...)
run_for_specified_minutes	Top Level	Set this value to the number of minutes to run the query.
file_system_name	Top Level	(required) Name of the file system in the format of {namespace}-siebfs{index}. Mounted file systems can be found in the environment directory inside the SCM container - **`/home/opc/siebel/{env_id}/`**.

Note that files in Siebel file system that are less than one hour old may not cleaned up.

For every cleanup job triggered via the POST API, a unique 6 character identifier "RUN_ID" will be generated through which the status can be checked. To get the details of the current job, execute a GET method API call on the following endpoint:

```
https://<CM_Instance_IP>:16690/scm/api/v1.0/environment/{ENV_ID}/sfscleanup/{RUN_ID}
```

After successful completion of the cleanup job, the reports of the run can be verified at the location inside the SCM container:

```
/home/opc/siebel/{ENV_ID}/{namespace}-siebfs{fs-index}/{NAMESPACE}/SFSUTILS/SFSCLEANUP
```

The report name can be fetched from the GET method response in the key - report_name. The report name follows a naming convention of - **`sfs_cleanup_{unix_timestamp}.txt`**.

An overall report containing historical statistics is also generated or updated at the location - **`/home/opc/siebel/{ENV_ID}/sfscleanup/overall_stats.txt`**.

4 Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer

About this Chapter

This chapter describes how to deploy Siebel CRM on a Kubernetes cluster using Siebel Installer for Siebel Cloud Manager (SCM) in:

- The cloud
- Your own data center:
  - On premises or
  - In Oracle Compute Cloud@Customer (OC3).
    For more information on OC3, refer *Oracle Cloud Compute @Customer*.

It contains the following topics:

- *Overview*
- *Moving Existing Siebel CRM on VM to a Kubernetes Orchestrated Deployment*
- *Moving Existing Siebel CRM on VM to an OC3 Kubernetes Cluster*
- *High-Level Steps for Deploying Siebel CRM on a Kubernetes Cluster*
- *Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster*
- *Downloading and Running Siebel Installer for SCM*
- *Installing SCM using Helm*
- *Migrating (Lift-And-Shift) Existing Siebel CRM Deployments*
- *Deploying Siebel CRM using SCM*
- *Updating SCM Configuration using Helm*
- *Reinstalling SCM using Helm*
- *Updating SCM using Helm*
- *Uninstalling SCM using Helm*
- *Troubleshooting Siebel CRM Deployment*
- *Deploying Siebel CRM on OpenShift using SCM*

Overview

You can deploy Siebel CRM on a Kubernetes cluster using Siebel Installer for SCM, either in the cloud or in your own data center (on premises or in OC3). Optionally, you can also:

- Enable the observability stack, that's metrics monitoring and log analytics, on a Kubernetes cluster on premises or in the cloud.

- Lift an existing Siebel CRM installation with all the customization and shift it to a Kubernetes orchestrated deployment.
- Perform new (greenfield) deployments on any of Cloud Native Computing Foundation (CNCF) certified Kubernetes, for example, Oracle Cloud Native Environment (OCNE) cluster:
  - On premises, or
  - In the cloud
- Enable Siebel CRM and Siebel Web Tools in languages other than English (ENU).

Note: When deploying Siebel CRM on a Kubernetes cluster on premises or in the cloud or in OC3 using Siebel Installer, SCM doesn't manage the infrastructure resources. It supports this deployment only through the “Bring Your Own Resources (BYOR)” option.

This chapter describes the steps required to do the following tasks:

- Download Siebel Installer for SCM.
- Set up SCM on a Kubernetes cluster.
- Deploy Siebel CRM on a Kubernetes cluster.

Moving Existing Siebel CRM on VM to a Kubernetes Orchestrated Deployment

You can move your existing Siebel CRM on VM to Kubernetes orchestrated deployments such as an Oracle Cloud Native Environment (OCNE) cluster using the Siebel Lift utility. This will retain all your customizations and offer the following benefits:

- Observability stack for metrics monitoring and log analytics.
- Other benefits of moving to a cloud-native paradigm, such as:
  - Dynamic auto-scaling.
  - Siebel CRM microservice applications deployment.
  - GitOps implementation.
  - Plug-and-play compatibility with other cloud native frameworks and applications, and more.

The Siebel Lift utility is available on MOS as **Siebel CRM 26.8.0.0 Lift Utility (Patch)**. For more information, see *Downloading and Running the Siebel Lift Utility*.

Moving Existing Siebel CRM on VM to an OC3 Kubernetes Cluster

OC3 is a fully managed, rack-scale distributed on-premises cloud platform that offers the OCI compute capabilities anywhere. OC3 is installed at a data center or location of your choice, offering a simple way to run your Siebel CRM applications while maintaining control over your data, security, and compliance within your premises.

Your Siebel CRM deployment on a Kubernetes cluster set up in OC3 enables you to:

- Utilize OCI's compute, storage, and networking capabilities to run Siebel CRM.
- Meet your organization's data residency requirements.
- Meet the requirements for real-time operations and low-latency connections to current data center assets.
- Use OCI services, APIs, and automation in the same way that the rest of Oracle's distributed cloud is.
- Use OCI Container Engine for Kubernetes (OKE) to achieve container automation, which simplifies enterprise-grade Kubernetes operations at scale.
- Free your IT resources to focus on meeting crucial business needs as the infrastructure is fully managed and supported by Oracle.
- Reduce costs through usage-based low consumption pricing and reasonably priced infrastructure subscriptions that result in significant savings.
- Use flexible storage for workloads requiring a lot of data, with 150 TB of storage capacity that is incrementally expandable to 3.65 PB. So, you just pay for the storage that is really used, not for the amount that is installed.

You can deploy Siebel CRM on a Kubernetes cluster set up in your OC3 instances using SCM with minimum efforts and take advantage of the benefits of OC3 with SCM as follows:

- Convenience of a cloud native observability stack.
- Reduce costs and save time by using SCM to run and operate your Siebel CRM deployment in OC3.
- Full support from Oracle for Siebel CRM deployment with SCM and OC3 infrastructure.

To successfully deploy Siebel CRM on a Kubernetes cluster set up in OC3 using SCM, you must ensure that:

- You set up and configure the following resources:
  - Open Container Initiative registry such as OCI Container Registry, Harbor and so on.
  - Kubernetes cluster and namespace such as OKE cluster, OCNE cluster, and so on.
  - Mount target and file system export path to store the SCM state and use as the Siebel CRM file system.
  - Database to use as the Siebel CRM database.
  - Git instance to use for Siebel CRM.

  For more information, see *Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster*.
- Have the prerequisites to deploy Siebel CRM in an OC3 instance available such as Python, OCI CLI and OCI configuration files. For more information, see *Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster*.
- Have the access details to update the different sections of the `values.yaml` file such as `siebelregistry`, `sshKey`, `userEncryptionKey`, `service`, and so on. For more information, see *Installing SCM using Helm*.
- Have the details to update the `ociConfig` and `instanceMetaData` sections in the `values.yaml` file with OC3 specific configuration. For more information, see *Installing SCM using Helm*.
- Refer to the steps to run Siebel Installer for installing SCM and deploying Siebel CRM. For more information, see *Downloading and Running Siebel Installer for SCM*.

Note: For more information, see *Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster*. Pay close attention to OC3 specific configurations required.

High-Level Steps for Deploying Siebel CRM on a Kubernetes Cluster

The following are the high-level steps to deploy Siebel CRM on a CNCF certified Kubernetes cluster (these steps are described in detail later in this document):

1. Install the prerequisite tools and make the prerequisite resources available with necessary connectivity and access provisions for deploying Siebel CRM on a Kubernetes cluster.
2. Download the Siebel Installer media zip files from My Oracle Support (MOS) and extract.
3. Run the installer and complete the setup and configuration.
4. Verify that the SCM image, images of SCM dependencies, and SCM Helm chart are available in your container registry.
5. Install SCM using Helm.
6. Verify the dependencies and prerequisites for Siebel CRM deployment.
7. Deploy Siebel CRM on a Kubernetes cluster using SCM APIs.
8. Verify Siebel CRM on the Kubernetes cluster.

Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster

This topic lists the prerequisites to deploy Siebel CRM on a Kubernetes cluster on premises or in the cloud, or in your data center on OC3, using Siebel Installer for SCM.

You'll need the following to successfully run Siebel Installer and install SCM:

- Linux VM version 8 or above for downloading installer and dependencies with a minimum of 35 GB free disk space and an additional 10 GB if you require support for Siebel Web Tools in languages other than English.
- Helm version 3.8 or later: A package that contains the resources needed to deploy an application on a Kubernetes cluster. The SCM Helm package contains the artifacts required to deploy Siebel on a Kubernetes cluster. It is also used to push the SCM package into the container registry and to deploy SCM and Siebel CRM on a Kubernetes cluster. For more information, refer the online documentation for "Installing Helm".
- Podman: An open-source tool for managing containers on Linux, Windows and so on. Here, Podman is used to manage the SCM container registry. For more information, refer the online documentation for "Podman Installation".
- Kubectl: A command line tool that helps users to manage their Kubernetes clusters. Here, Kubectl is used to manage the cluster on which SCM and Siebel CRM are deployed. For more information, refer the online documentation for kubectl.
- VNC session or Xterm to run Siebel Installer in the GUI mode.
- Siebel CRM: The minimum version of Siebel CRM for migration to OCNE is Siebel CRM 18.12 or later. The Siebel CRM on-premises environment must be running when you run the Siebel Lift utility.
- Container registry with the appropriate credentials: You must have an open container initiative compliant registry like Harbor with the following registry details:
    - Registry URL: The container registry URL.

    - Registry credentials: The user name and password to access the container registry.
    - (Optional) Registry prefix: When the prefix is specified, the repository path is constructed using the registry prefix. The registry user must have the privileges to create the repository, or it must be created before running the installer.

- Kubernetes cluster: A Kubernetes cluster to install SCM and Siebel CRM. You must configure access to the cluster in the Linux host, that is copy the `kubeconfig` file of the Kubernetes cluster to a directory on Linux and set the path of the environment variable `KUBECONFIG` as follows:

  ```
  export KUBECONFIG = "/scratch/.kube/<kubeConfigFile>"
  ```

  In this example, `<kubeConfigFile>` is the configuration file of the Kubernetes cluster on which you want to install SCM and deploy Siebel CRM.

- NFS share: You must create an NFS share to store the SCM and Siebel CRM environment state information. This directory should be accessible from the Kubernetes cluster worker nodes for mounting into the SCM pod and, later, into the Siebel CRM pods for the Siebel file system. For example:

  ```
  <nfsServerHost>:/<nfs-path>
  ```

  > **Note:** The NFS share should have the no_root_squash parameter set for exports.

- Kubernetes namespace: The logical division within the Kubernetes cluster in which you want to install SCM. You can create the namespace for SCM installation as follows:

  ```
  kubectl create namespace <namespace>
  ```

  In this example, `<namespace>` is the name of the Kubernetes namespace to install SCM in.

  > **Note:** You can also use an existing namespace, but ensure the namespace is empty. For more information, see *Using a BYO Namespace*.

- Custom Resource Definitions (CRDs): CRDs allow you to extend the Kubernetes API with custom objects, known as "custom resources". These are created at the cluster level, that is, they are available cluster-wide to any namespace within the cluster. Service accounts that perform Siebel CRM deployments will have cluster-level access.

> **Note:**
> - SCM instructions currently are in U.S. English (ENU).
> - While "lift-and-shift" supports all languages that Siebel CRM supports, Greenfield deployments of Siebel CRM using SCM currently support U.S. English (ENU) only.

For OC3, additionally, you must ensure that the following are available to successfully install SCM and deploy Siebel CRM:

- Python 3.14.4 or later: Python is required to run OCI Command Line Interface (CLI). Hence, you must set up an OCI CLI compatible version of Python on the Linux host machine on which you will run Siebel Installer and install SCM using Helm.
- OCI CLI: OCI CLI provides the same core functionality as the OCI Console. It is used to ensure that the OCI config file is set up correctly. To set up OCI CLI on the Linux host, run the following commands:

  ```
  pip3 install oci-cli --user
  ls -l .local/bin/
  export PATH=${PATH}:/home/<username>/.local/bin
  oci -v
  ```

> **Note:** When installing OCI CLI, if the compatible version of Python is not installed, OCI CLI will install Python.

- OCI configuration files: SCM installed on a Kubernetes cluster in OC3 uses OCI SDKs and CLI. Hence, you must ensure that the OCI API-compatible services running in the OC3 control plane components are accessible from the Linux host machine on which SCM is installed. To ensure accessibility, prepare the OCI SDK and OCI CLI configuration files in the OC3 environment as follows.

> **Note:** The default location of the OCI SDK and OCI CLI configuration files, `config` and `oci_cli_rc`, respectively, is the `~/.oci` directory.

> **Note:** All the paths mentioned here are only for example. You can select the paths of your choice when deploying in your environment.

1. Prepare the RSA key pair in PEM format as follows:
   a. Generate a 2048-bit private key in the PEM format, as follows:
   ```
   mkdir /home/opc/.oci; cd ~/.oci
   openssl genrsa -out /home/opc/.oci/oci_api_key.pem 2048
   chmod 600 /home/opc/.oci/oci_api_key.pem
   ```
   b. Generate a public key in the PEM format, as follows:
   ```
   openssl rsa -pubout -in /home/opc/.oci/oci_api_key.pem -out $HOME/.oci/oci_api_key_public.pem
   ```
   c. Open the public key PEM file and copy the key:
   ```
   cat /home/opc/.oci/oci_api_key_public.pem
   ```
   > **Note:** You must ensure that you copy the lines BEGIN PUBLIC KEY and END PUBLIC KEY also along with the key.
   d. Add the public key to your user account in the OC3 console, as follows:
      i. Sign in to the OC3 console.
      ii. Navigate to the **My profile** section.
      iii. In the left pane, click **API keys**.
      iv. Click **Add API key**.
      v. Select **Paste a public key**.
      vi. Paste the public key (copied in step c).
      vii. Click **Add**.
2. Download the Certificate Authority (CA) certificate bundle for the OC3 environment as follows:
   ```
   curl -k https://<oc3_region>/cachain > /home/opc/.oci/ca.crt
   ```
   In the example above, `<oc3_region>` is the customer region.
3. Update the OCI SDK configuration file with connectivity details such as the user credentials, tenancy OCID, and so on, as follows:
   ```
   vi /home/opc/.oci/config

   [DEFAULT]
   user=ocid1.user.xxxxxx..........oe2f249bo8ho4z2kp5di6gk20w.........kg1i705v.....
   fingerprint=c4:11:86:05:d3:4........:64:91:ea:2d......
   tenancy=ocid1.tenancy.xxxxxx..........ohtq81ez9p8etm2cry04u0m6..........
   region=<oc3_region>
   key_file=/home/opc/.oci/oci_api_key.pem
   ```
4. Create and update the OCI CLI configuration file with the CA certificate details as follows:

```
vi /home/opc/.oci/oci_cli_rc

[DEFAULT]
custom_cert_location=/home/opc/.oci/ca.crt
cert-bundle=/home/opc/.oci/ca.crt
```

5. Verify the OCI API connectivity in OC3, as follows:

```
oci os ns get
```

Response received:

```
{
     "data": "aveu8wbpqcen"
}
```

Downloading and Running Siebel Installer for SCM

This topic describes the steps for downloading Siebel Installer for SCM, running the installer to lay down the SCM image, images of SCM dependencies, and SCM Helm chart, and verifying the SCM images and SCM Helm chart in the container registry.

This topic includes the following sections:

- *Overview of Siebel Installer for SCM*
- *Downloading the Build*
- *Running Siebel Installer*
- *Running the Installer in the GUI Mode*
- *Running the Installer in the Silent Mode*
- *Siebel Installer Outputs*
- *Verifying the SCM Image and SCM Helm Chart in the Container Registry*

Overview of Siebel Installer for SCM

Siebel Installer simplifies the task of downloading the SCM artifacts required to set up SCM on a Kubernetes cluster on premises or in the cloud. Siebel Installer lays down the SCM image, images of SCM dependencies, and the SCM Helm chart, required to install SCM, in an assigned directory and then pushes them to the user specified container registry. You can download Siebel Installer from My Oracle Support (MOS) and proceed with the subsequent SCM installation procedure without internet access.

Optionally, you can sign images that are pushed to the user specified container registry with a customer key. The image tools container uses Cosign to sign and verify image digests after the images are pushed to the customer registry. The installer supports Cosign-native keys and unencrypted RSA or ECDSA PEM private keys with matching public keys. The installer imports RSA or ECDSA PEM keys into Cosign format.

Downloading the Build

To download the installer:

1. Download the media files from MOS as follows:
   a. Log in to MOS.

b. In the **Search** field, search and select **Patches and Product Certification Matrix**.
c. In the **Patch Search** pane:
   i. In the **Platform** field, search and select the platform. For example, Linux x86-64.
   ii. In the **Language** field, search and select the Language. For example, American English.
   iii. In the **Product** field, search and select Siebel CRM.
   iv. In the **Release** field, search and select the release version. For example, Siebel Industry Applications 26.8.
   v. Click **Apply**.

   The Patches page displays the files available for download, for the search. For example, for version 26.8, the following media files are listed: .

   - Siebel CRM 26.8.0.0 2026_08 (Patch)
   - Siebel CRM 26.8 OL Containers and Sample DB (Patch)
   - Siebel CRM 26.8.0.0 Cloud Manager (Patch)
   - Siebel CRM 26.8.0.0 Lift Utility (Patch)

d. In the **Patch Name** field, click the Cloud Manager (Patch) link. For example, Siebel CRM 26.8.0.0 Cloud Manager (Patch).
e. Click **Download**.

You will receive 2 files, for example, `p123XXXXX_268_Linux-x86-64_1of2.zip` and `p123XXXXX_268_Linux-x86-64_2of2.zip`.

> **Note:** Here,the file name is representational and will change with the build. The format of the media file name is `<patch_id>_<release_tag>_<platform>_<file_archive_number>of<total_files>.zip`, where:
>
> - `<patch_id>` is the patch or the bug number. For example, `39713957`.
> - `<release_tag>` is the SCM release version. For example, 26.8 for the SCM release 26.8.
> - `<platform>` is the platform you wish to install SCM on. For example, `Linux-x86-64`.
> - `<file_archive_number>` is a number assigned to a file in a set of media files. For example, `1` is the archive number of the file in the file name `p123XXXXX_268_Linux-x86-64_1of2.zip`.
> - `<total_files>` is the total number of media files. For example, `2` in the file name `P123XXXXX_268_Linux-x86-64_2of2.zip`.

- > **Note:** The file with archive number 1 includes Siebel Installer for SCM, the SCM image, images of SCM dependencies, the SCM Helm chart, and configuration files. The file with archive number 2 includes the SCM third-party images.

2. Extract the media zip file with archive number 1 to the directory (which will be referred to as the "installation directory" in the steps later) from which you want to run the installer, as follows:
   `cd <installation_directory>`

```
unzip p123XXXXX_268_Linux-x86-64_1of2.zip
```

The following files are extracted in the installation directory:

```
drwxrwxrwx 3 root root 3 Oct 22 10:40 ext
drwxrwxrwx 2 root root 4 Oct 22 10:41 resources
drwxrwxrwx 2 root root 2 Oct 22 10:41 logs
drwxrwxrwx 3 root root 3 Oct 22 11:13 archives
-rwxrwxrwx 3 root root 3 Oct 25 09:31 README.html
-rwxrwxrwx 1 root root 186 Oct 25 09:31 runInstaller.sh
drwxrwxrwx 3 root root 3 Oct 29 07:24 jre
-rwxrwxrwx 1 root root 53 Nov 22 06:46 installer.properties
```

Note: Make a note of this installation directory; later you'll run the SCM installation from this directory.

3. Move the media zip file with archive number 2 to the `<installation_directory>/archives/scm_archives` directory. This directory was created when you extracted the media zip file with archive number 1 in step 2:

```
mv p123XXXXX_268_Linux-x86-64_2of2.zip <installation_directory>/archives/scm_archive
```

Note: You must only move the media zip file, do not unzip it.

4. Download the Siebel CRM image matching the Siebel Installer version from MOS, as follows:
   a. Click the **Patches & Updates** tab.
   b. In the **Patch Search** pane, click the **Product or Family (Advanced)** tab.
   c. In the **Product** field, enter Siebel CRM.
   d. In the **Release** field, enter the release version. For example, Siebel Industry Applications 26.8.
   e. In the **Platform** field, select the platform. For example, `Linux x86-64`.
   f. Click **Search**.

      The results pane displays the files available for download for the search. For example, for the installer version 26.8, the following media files are listed:

      - Siebel CRM 26.8.0.0 2026_08 (Patch)
      - Siebel CRM 26.8 OL Containers and Sample DB (Patch)
      - Siebel CRM 26.8.0.0 Cloud Manager (Patch)
      - Siebel CRM 26.8.0.0 Lift Utility (Patch)

   g. In the **Patch Name** field, click the container link. For example, Siebel CRM 26.8 OL Containers and Sample DB (Patch).
   h. Click **Download**. The **File Download** pane lists the downloadable media files for the patch. For example:
      - `p123XXXXX_268_Linux-x86-64_1of2.zip`
      - `p123XXXXX_268_Linux-x86-64_2of2.zip`
   i. Download the media file with the archive number 2. For example, `p123XXXXX_268_Linux-x86-64_2of2.zip`.
   j. Move the downloaded media zip file to the `<installation_directory>/archives/scm_archives` directory. Do not unzip the file. For example:

      ```
      mv p123XXXXX_268_Linux-x86-64_2of2.zip <installation_directory>/archives/scm_archive
      ```

5. (Optional, perform this step only if you need support for Web Tools in languages other than English.) Download the Siebel Safeboot zip file from MOS, as follows:
   a. Click the **Patches & Updates** tab.
   b. In the **Patch Search** pane, click the **Product or Family (Advanced)** tab.
   c. In the **Product** field, enter Siebel CRM.
   d. In the **Release** field, enter the release version. For example, Siebel Industry Applications 26.8.
   e. In the **Platform** field, select the platform. For example, Linux x86-64.
   f. Click **Search**.

      The results pane displays the files available for download for the search. For example, for the version 26.8, the following media files are listed:

      - Siebel CRM 26.8.0.0 2026_08 (Patch)
      - Siebel CRM 26.8 OL Containers and Sample DB (Patch)
      - Siebel CRM 26.8.0.0 Cloud Manager (Patch)
      - Siebel CRM 26.8.0.0 Lift Utility (Patch)
   g. In the **Patch Name** field, click the Siebel CRM patch link. For example, Siebel CRM 26.8.0.0 2026_08 (Patch).
   h. Click **Readme** under the patch details pane. Under the Linux/Unix Archive Files, note the archive number for **Siebel Non-English (US) Safeboot**.
   i. Click **Download**. The File Download pane lists the downloadable media files for the patch. For example:
      - `p123XXXXX_268_Linux-x86-64_9of9.zip`
      - `p123XXXXX_268_Linux-x86-64_8of9.zip`
      - `p123XXXXX_268_Linux-x86-64_7of9.zip`
      - `p123XXXXX_268_Linux-x86-64_6of9.zip`
      - `p123XXXXX_268_Linux-x86-64_5of9.zip`
      - `p123XXXXX_268_Linux-x86-64_4of9.zip`
      - `p123XXXXX_268_Linux-x86-64_3of9.zip`
      - `p123XXXXX_268_Linux-x86-64_2of9.zip`
      - `p123XXXXX_268_Linux-x86-64_1of9.zip`
   j. Click and download the Safeboot archive number file. For example, if the archive number you noted in the earlier step **h** is **3**, then you download the `p123XXXXX_268_Linux-x86-64_3of9.zip` file.
   k. Move the media zip file to the `<installation_directory>/archives/scm_archives` directory. Do not unzip the file.

      ```
      mv p123XXXXX_26_Linux-x86-64_3of9.zip <installation_directory>/archives/scm_archive
      ```

      > **Note:** If any required zip file is not copied to the `<installation_directory>/archives/scm_archives` directory during installation, you must uninstall and perform a fresh installation.

6. (Optional, perform this step only if you want to run the installer as a non-root user.) Configure the non-root user to push the images to the container registry, as follows:

   ```
   sudo /sbin/usermod --add-subuids <first-last> --add-subgids <first-last> <uid>
   podman system migrate
   ```

```
/usr/bin/systemctl --user daemon-reload
```

The variables in the example have the following values:

- `<uid>` is the non-root user ID.
- `<first-last>` is the subordinate user/group IDs range that the non-root user is allowed to use. For example, 20000-265536.

7. (Optional) Sign in to the container registry using Podman and Helm utilities.

> **Note:** If you sign in to the container registry now, which is before running the installer, you need not provide the registry credentials in the SCM configuration details when you run the installer. The credentials with which you've logged in to the container registry are used to push the SCM image, images of SCM dependencies, and SCM Helm chart to the container registry.

Running Siebel Installer

You can run Siebel Installer either in the GUI mode or silent mode.

Running the Installer in the GUI Mode

To run the installer in the GUI mode:

1. Start the installer in the GUI mode as follows:
   ```
   sh runInstaller.sh
   ```
2. Complete the installation as follows:
   a. In the **Installation Location** screen, click **Browse** to select the directory to store the SCM artifacts and click **Next**.
   b. In the **Component Selection** screen, the installation component "Siebel Cloud Manager" and configuration task "Siebel Cloud Manager Configuration" are selected by default. Click **Next** to continue.

   > **Note:** Don't deselect "Siebel Cloud Manager" or "Siebel Cloud Manager Configuration". If you deselect “Siebel Cloud Manager,” the installer will throw an error and you'll not be able to proceed with the installation. If you deselect "Siebel Cloud Manager Configuration", the installer will not upload the SCM image, images of SCM dependencies, and SCM Helm chart to the container registry.

   c. In the **Configuration Details** screen, enter the container registry details required to push the SCM image, images of SCM dependencies, and SCM Helm chart to the container registry:
      - Container Registry URL: The container registry URL to push the SCM image, images of SCM dependencies, and the SCM Helm chart to. For example, `harbor-registry.corpxyz.mytenancy02phx.oraclevcn.com`.
      - Container Registry URL Prefix: The prefix for the container registry URL. It's used to construct the repository path. This field is optional or mandatory based on the registry used. If the registry mandates that the prefix be specified for better organization of projects or namespaces, then this

field is mandatory; else it's optional and can be left blank. For example, if the prefix is `mytenancy` and the suffix is `scmxyz`, then the repository path will be `iad.ocir.io/mytenancy/scmxyz/image-name`.

> **Note:** For OCI container registry, the prefix is mandatory and must be the OCI namespace. Additionally, you can also include an optional suffix. For example, if the namespace is `mytenancy` and the suffix is `scmproject`, then the prefix will be `mytenancy/scmproject`.

- Container Registry UserName: The user name to access the container registry. It is optional if the installer is launched from the session in which the user has already logged in to the container registry.
- Container Registry Password: The password to access the container registry. It is optional if the installer is launched from the session in which the user has already logged in to the container registry.
- Additional Web Tools Languages: A comma-separated list of languages other than English in which you wish to support the Web Tools. For example, JPN, DEU, and so on. For a list of languages supported by Siebel Web Tools and their corresponding language codes, refer to the Article ID 1513102.1 on My Oracle Support.
- Customer Image Signing Enabled: Set this field to `true` to sign images after they are pushed to the customer registry. This field is optional.
- Customer Cosign Private Key: The Cosign private key or importable unencrypted RSA or ECDSA PEM private key. This field is required only when customer signing is enabled.
- Customer Cosign Public Key: The matching public key file. This field is required only when customer signing is enabled.
- Customer Cosign Password: The password that Cosign uses for signing. This field is required only when customer signing is enabled.

d. In the **Summary** screen, review details and click **Install**.
e. Optionally, you can save the installation response messages to a response file. To create a response file:
   i. Click **Save Response File.**
   ii. Go to the required directory.
   iii. Enter the response file name.
   iv. Click **Save**.
f. In the **Installation Progress** screen, as the installation progresses, appropriate messages are displayed. After you receive the message "Post Installation Configuration Complete", click **Next**.
g. In the **Finish Installation** screen, after the installation is complete, the "Installation is successful" message is displayed along with the path and name of the log file. Click **Close**.

Running the Installer in the Silent Mode

You can use the silent mode to perform a fresh installation or configuration. The silent mode installation uses a response file that contains information required by the installer to complete the installation. A sample response file is available in the Siebel Installer location. The `INSTALLATION_TYPE` parameter in the response file allows you to define the type of installation you want to perform.

You can run the installer in the silent mode using one of the following options:

- Option 1: Using the sample response file available in the installer location as follows:

  a. Update the response file:
     - Set the `INSTALLATION_TYPE` parameter to one of the following values:
       - Fresh
       - Reconfiguration

     > **Note:** Reconfiguration only performs the post-install tasks, it doesn't perform the installation of binaries again.

     - Set the values of other parameters as per your environment configuration.

     Here's a sample of the updated response file:

```
 [Installation Details]
INSTALLATION_TYPE="Fresh"
SIEBEL_HOME_LOCATION="/home/siebel"
SELECTED_COMPONENTS="Siebel Cloud Manager"
SELECTED_CONFIGURATION_COMPONENTS="Siebel Cloud Manager"
[Siebel Cloud Manager Configuration]
CONTAINER_REGISTRY_URL=harbor-registry.xxx.xxxx.xxxx.com
CONTAINER_REGISTRY_URL_PREFIX=abc
CONTAINER_REGISTRY_USERNAME=user1
CONTAINER_REGISTRY_PASSWORD=BJUkxRM2ZtUjxxxxxl3Q0t4MkllcldOWGxjVW82QjlNaWdsXXXXXXZ
ADDITIONAL_WEB_TOOLS_LANGUAGES=JPN,DEU
CUSTOMER_IMAGE_SIGNING_ENABLED=true
CUSTOMER_COSIGN_PRIVATE_KEY=/secure/scm/customer.key
CUSTOMER_COSIGN_PUBLIC_KEY=/secure/scm/customer.pub
CUSTOMER_COSIGN_PASSWORD={encrypted-password}
```

     > **Note:**
     > - Passwords in the response file are encrypted. Before you add or update passwords in the response file, you must encrypt the passwords using the Siebel encryption utility as follows:
     >
     > ```
     > jre/bin/java -jar ext/jlib/EncryptString.jar <password>
     > ```
     >
     > - If you omit `CUSTOMER_IMAGE_SIGNING_ENABLED` or set it to `false`, the installer pushesimages as before. You do not need to provide a customer signing key.

  b. Run the installer in the silent mode using the updated response file as follows:
     - Linux:

     ```
     ./runInstaller.sh -silent -responseFile <Location of responsefile>
     ```

     - Windows:

     ```
     runInstaller.bat -silent -responseFile <Location of responsefile>
     ```

- Option 2: Generating the response file using the GUI and then running the installer in silent mode using the generated response file, as follows:

  a. Run Siebel Installer in the GUI mode and navigate through the installation screens providing required information until you reach the Summary screen.

     For more information on running Siebel Installer in GUI mode, see *Running the Installer in the GUI Mode*.
  b. In the Summary screen, create a response file as follows:
     i. Click **Save Response File**.
     ii. Go to the required directory.
     iii. Enter the response file name.
     iv. Click **Save**.
  c. Cancel the installation.
  d. Run Siebel Installer in silent mode using the response file created in step b. For more information on running Siebel Installer in silent mode, see Option 1 discussed in this section.

Siebel Installer Outputs

If you provide a Cosign key, the installer uses the supplied key files and generates the following file:

```
$HOME/mirror-image-manifest.yaml
```

You must keep the supplied Cosign key files for the Helm installation step.

If you provide an RSA or ECDSA key, the installer imports the key into Cosign format and creates the following files:

```
$HOME/customer-cosign.key
$HOME/customer-cosign.pub
$HOME/mirror-image-manifest.yaml
```

For RSA or ECDSA input, you must use the generated `customer-cosign.key` and `customer-cosign.pub` files for Helm installation.

Verifying the SCM Image and SCM Helm Chart in the Container Registry

After the installer is run successfully, you must verify that the SCM image, images of SCM dependencies, and SCM Helm chart are available in the container registry for smooth SCM installation through one of the following ways:

- Open the log file and look for the following messages:

  ```
  Uploaded image and helm chart to customer registry
  Post install configuration step completed for Siebel Cloud Manager Configuration
  Post Installation Configuration Completed.
  ```

- Sign in to the container registry and verify the image from the interface provided by the registry, for example, the GUI.

> **Note:** By default, when you run Siebel Installer, the SCM image, images of SCM dependencies, and SCM Helm chart are pushed to the user container registry.

Installing SCM using Helm

This topic describes the steps to install SCM on a Kubernetes cluster on premises or in the cloud or in your data center on OC3 using Helm.

This topic includes the following sections:

- *Before Installing SCM*
- *Installing SCM*

Before Installing SCM

You must perform the following preinstallation tasks before installing SCM on a Kubernetes cluster:

1. Ensure you've access to the installation directory and container registry provided in Siebel Installer.
2. (Required only when customer image signing is enabled during Siebel Installer setup.) Copy the image trust files that Siebel Installer creates when image signing is enabled into the extracted SCM Helm chart directory, and create the password file as follows:
   a. Go to the `cloudmanager` directory:
      ```
      cd cloudmanager
      ```
   b. Create the `image_trust` directory:
      ```
      mkdir -p image_trust
      ```
   c. Copy the image trust files:
      ```
      cp $HOME/mirror-image-manifest.yaml image_trust/mirror-image-manifest.yaml
      cp $HOME/customer-cosign.pub image_trust/customer-cosign.pub
      cp $HOME/customer-cosign.key image_trust/customer-cosign.key
      ```
   d. Create the password file from the same customer signing password:
      ```
      printf '%s' '{customer-cosign-password}' > image_trust/cosign.password
      ```
   e. Set the file permissions:
      ```
      chmod 600 image_trust/customer-cosign.key image_trust/cosign.password
      chmod 644 image_trust/customer-cosign.pub image_trust/mirror-image-manifest.yaml
      ```

3. Update the `values.yaml` file: The SCM Helm package includes a default `values.yaml` file which determines how SCM will be configured. Before installing SCM, you must update the `values.yaml` file to configure SCM as per your requirements. To update the `values.yaml` file:
   a. Open the `values.yaml` file. You can use the `values.yaml` file in either:
      - The installation directory on the Linux host machine that was used to run Siebel installer, or
      - The SCM Helm chart in your container registry. To use the `values.yaml` in the container registry:
        a. Sign in to the container registry as follows:
           ```
           helm registry login <registry>
           ```
           In this example, `<registry>` is the basename of the container registry.
        b. Pull the SCM Helm chart from the container registry:
           ```
           helm pull oci://<registry>/<repositoryPath> --version <releaseVersion>
           ```
           The variables in the example have the following values:
           - `<registry>` is the container registry basename.
           - `<repositoryPath>` is the SCM Helm chart (`cloudmanager`) repository path.
           - `<releaseVersion>` is the SCM release version.
   b. Unzip the SCM Helm chart zip file as follows:
      ```
      tar -zxf cloudmanager_CM_<releaseVersion>.tgz
      ```
      In this example, `<releaseVersion>` is the SCM release build version that you downloaded.
   c. Update the following sections in the `values.yaml` file:
      - The `image` section with the container registry details (provided in the Siebel Installer configuration tasks) from which the SCM image, images of SCM dependencies, and SCM Helm chart will be used for deployment, as follows:
        ```
        image:
             tag: "<imageTag>"
             imagePullPolicy: IfNotPresent
        ```
        The variables in the example have the following values:
        - `<imageTag>` is the SCM release version.
        - `<imagePullPolicy>` determines when the SCM image is pulled from container registry. It can take the following values: IfNotPresent, Always or Never.
      - The `siebelregistry` section with the image registry details as follows:
        ```
        siebelregistry:
             url: <registryURL>
             registry_prefix: <registryPrefix>
             user: <userName>
             password: <password>
        ```
        > **Note:** If you do not prefer to store the `user` and `password` details in the `values.yaml` file, you can pass these details in the `helm install` command during SCM installation.

        The variables in the example have the following values:
        - `<registryURL>` is the container registry URL to which the SCM image, images of SCM dependencies, and SCM Helm chart were pushed by Siebel Installer.

  - **<registryPrefix>** is the prefix for the container registry URL to which the SCM image, images of SCM dependencies, and SCM Helm chart were pushed by Siebel Installer.
  - **<userName>** is the container registry user name.
  - **<password>** is the container registry user password.

- (Optional) The **securityContext** section with a custom non-root user ID and group settings for the SCM pod:

```
securityContext:
    runAsUser: <uid>
    runAsGroup: 0
    fsGroup: <fs_group_id>
```

  In the above example:

  - **<uid>** is the user ID under which SCM containers and processes run.
  - **<fs_group_id>** is the file system group ID used to provide access to mounted volumes and shared storage.

  You must create a base directory in the SCM mount path and assign ownership to the user ID specified in **securityContext.runAsUser** and assign group ownership to **0**. Ensure that the directory name matches the SCM namespace name in uppercase. For example:

```
sudo mkdir <SCM_NAMESPACE>
sudo chown -R <uid>:0 <SCM_NAMESPACE>
sudo chmod -R g+rwx <SCM_NAMESPACE>
```

  In the above example:

  - **<SCM_NAMESPACE>** is the SCM namespace name in uppercase.
  - **<uid>** is the value configured for **securityContext.runAsUser**.

- (Optional) The **imageTrust** section with the image trust file details, as follows:

```
imageTrust:
    enabled: true
    mirrorManifestFilename: mirror-image-manifest.yaml
    customerPublicKeyFilename: customer-cosign.pub
    customerPrivateKeyFilename: customer-cosign.key
    customerPrivateKeyPasswordFilename: cosign.password
```

  If customer signing is disabled, leave **imageTrust.enabled** set to **false**. You do not need to create the **image_trust** files for Helm installation.

- (Optional) The **resources** section with resource (CPU, memory, and ephemeral storage) allocation for the SCM pod. The default limits and requests values already specified for the resources in the **values.yaml** are sufficient for Siebel CRM deployment, but you can update these values as required as per the size of your Siebel CRM deployment.
- The **storage** section with the network file system (NFS) path for SCM and Siebel CRM deployment as follows:

```
storage:
     nfsServer: <nfsServer>
     nfsPath: <nfsPath>
```

```
storageSize: 200Gi
```

The variables in the example have the following values:

- `<nfsServer>` is the IP address or fully qualified domain name of the NFS server.
- `<nfsPath>` is the export path in the NFS server to access the SCM file system.

- The `sshKeysection` with the public and private key file names required for establishing connection between Git repository and Flux CD operator as follows:
  a. Create a SSH key pair as follows:

```
% ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/<uname>/.ssh/id_ed25519): /Users/<uname>/
sample
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/<uname>/sample
Your public key has been saved in /Users/<uname>/sample.pub
```

  In this example, `<uname>` is the user name.
  b. Copy the private and public key files to the `ssh` directory in the SCM Helm chart home directory (`cloudmanager`).
  c. Update the `sshKey` section with the private and public key file names:

```
sshKey:
     pvtKeyFilename: <privateKeyFilename>
     pubKeyFilename: <publicKeyFilename>
```

  The variables in the example have the following values:

  - `<privateKeyFilename>` is the private key file name.
  - `<publicKeyFilename>` is the public key file name.

- (Optional) The `affinity` section with the details of node affinity scheduling rules for the SCM pod. For more information on the `affinity` parameters, refer to the *Kubernetes API* documentation for the resource pod.

> **Note:** The `affinity` section in the `values.yaml` is applicable only to the SCM pod. For implementing affinity for Siebel pods, see *Assigning Pods to Nodes - Implementing Affinity and Anti-affinity on OKE using Siebel Cloud Manager*.

- The `ociConfig` section with the details of the files required for OCI API authentication to access OCI infrastructure services in an OC3 environment as follows:

> **Note:** You must configure the `caCrtFilename` and `ociCliRcFilename` parameters only when deploying Siebel CRM in OC3.

```
ociConfig:
     ociPvtKeyFilename:<ociPrivateKeyFilename>
     caCrtFilename: <caCertificateFileName>
```

```
ociCliRcFilename : <cliRCFileName>
```

The variables in the example have the following values:

- **<ociPrivateKeyFilename>** is the private key PEM file name. For example, oci_api_key.pem.
- **<caCertificateFileName>** is the CA certificate file name. For example, ca.crt.
- **<cliRCFileName>** is the OCI CLI RC configuration file name.For example, oci_cli_rc.

- The **instanceMetaData** section with the applicable region and compartment OCID values as follows:

```
instanceMetaData:
     vaultEnabled: "False"
     region: <region>
     compartmentOcid: <compartmentOCID>
     ociDeployment: <deploymentType>
```

The variables in the example have the following values:

- **<region>** is the canonical region name. For example, us-ashburn-1.
- **<compartmentOCID>** is the OCID of the compartment used for Oracle Cloud Infrastructure (OCI) calls.
- **<deploymentType>** determines the environment on which you are deploying Siebel CRM. If you are deploying Siebel CRM on:
  - A CNCF certified Kubernetes cluster on premises or in the cloud, set the value of this parameter to "false". This parameter is of string type, so ensure you enclose false in quotes.
  - OC3 in your data center, set the value of this parameter to "oc3".
  - OCI, set the value of this parameter to "public".

- The **userEncryptionKey** section, enable this section and update it only when the **vaultEnabled** parameter is set to **false**.

```
userEncryptionKey:
     uek: "<encryptionkey>"
```

In this example, **<encryptionkey>** is a key which matches the following expression: ^[a-zA-Z0-9]{56,60}$

- The **service** section with the service type that will be used to expose SCM deployment as follows:

```
service:
```

```
serviceType: <servicetype>
```

In this example, `<servicetype>` can be one of the following: ClusterIP, NodePort or LoadBalancer. Since SCM is exposed over https, it requires an SSL certificate.

You can use your SSL certificate or allow SCM to generate it. If you set the `useCustomSSLCertificate` parameter to:

- `true`: You can use your SSL certificate as follows:
  1. Copy your SSL certificate to the `ssl` directory in the SCM Helm chart home directory.
  2. Update the `certificatePath`, `keyPath` and `caCertPath` parameters with the relative path of the certificate under the `scmCustomCert` section.

     > **Note:** You must update the `caCertPath` parameter under the `scmCustomCert` section with the appropriate path only when you set the `useDualTls` parameter to `true` under the `loadBalancer` section.

- `false`: SCM generates the self-signed SSL certificate based on parameters configured under the `scmSelfSignedCert` section.

  Configure other parameters applicable for the service type:

  - `NodePort` and `ClusterIP`: The `customMetadata`, `customLabels` and `customAnnotations` sections are optional. The following is a sample of the service section for `NodePortservice` type:

```
service:
     serviceType: "NodePort"
     nodePort: <valid-unused-port>
     name: "scm-app-svc"
     useCustomSSLCertificate: false
     scmCertSecret:
          name: "scm-app-ssl"
          customMetadata: {}
          customLabels: {}
          customAnnotations: {}
     scmSelfSignedCert:
          country: "US"
          state: "California"
          locality: "San Francisco"
          organization: "Oracle Corporations"
          commonName: "oracle.com"
     scmCustomCert:
          certificatePath: "ssl/scm.crt"
          keyPath: "ssl/scm.key"
          caCertPath: "ssl/ca.crt"
     customMetadata: {}
     customLabels: {}
     customAnnotations: {}
```

    You can optionally set `service.nodePort` to assign a fixed Kubernetes NodePort to the SCM service. Enter an unused integer within the NodePort range configured for

your Kubernetes cluster. If you omit this parameter, Kubernetes assigns a NodePort automatically.

> **Note:** Your cluster administrator can configure the NodePort range at the cluster level. Kubernetes uses the default range of 30000-32767. Self-managed clusters can use a different range with the `kube-apiserver --service-node-port-range` option. For more information:
>
> - Refer *service-node-port-range* description in the Kubernetes *kube-apiserver* documentation
> - Refer to the *Configuring the node port service range* chapter in the RedHat documentation, for OpenShift.

If your cluster uses a custom NodePort range, make sure that the `service.nodePort` value falls within that range. The Kubernetes API server rejects a value outside the configured range.

- `LoadBalancer`: The `customAnnotations` section is mandatory and you must configure other parameters under this section as follows:
  - Configure the `exposedPort` parameter to expose SCM at specific port.

    > **Note:** You must use the same port in the annotations added under the `customAnnotations` section of the `service` section. For example, if `exposedPort` is set to `443`, under the `customAnnotations` section of the `service` section, add the following annotation:
    >
    > ```
    > service.beta.kubernetes.io/oci-load-balancer-ssl-ports: "443"
    > ```

  - Set the `useDualTls` parameter to `true` to enable end-to-end TLS for SCM; in this case, the Load Balancer uses two sets of SSL certificates:
    - Listener certificate that is stored as a Kubernetes secret under the `loadBalancer` section.
    - Backend-set certificate that is stored as a kubernetes secret under the `service` section.
  - If the `useCustomSSLCertificate` parameter under the service section is set to:
    - `true`: Copy your SSL certificate under `ssl` directory in the SCM Helm chart home directory and update the relative path of the certificate under the `customCert` section under the `loadBalancer` section. Additionally, if the `useDualTLs` parameter is set to `true` then you must also configure the `scmCustomCert` section under the `service` section.
    - `false`: SCM generates a self-signed SSL certificate based on parameters configured under the `selfSignedCert` section under `loadBalancer` section.

Additionally, if the `useDualTLs` parameter is set to `true` then you must also configure the `scmSelfSignedCert` section under the `service` section.

Note: The `customAnnotations` section must have annotations with the correct secret name for listener certificates and backend-set certificates.The following is a sample of the `customAnnotations` and `loadBalancer` sections when deploying Siebel CRM in OC3 using `LoadBalancer` as the `serviceType`:

```
service:
     serviceType: "LoadBalancer"
     name: "scm-app-svc"
     useCustomSSLCertificate: false
     customAnnotations:
          oci.oraclecloud.com/load-balancer-type: "lb"
          service.beta.kubernetes.io/oci-load-balancer-tls-secret: scm-lb-ssl
          service.beta.kubernetes.io/oci-load-balancer-internal: "false"
          service.beta.kubernetes.io/oci-load-balancer-shape: "flexible"
          service.beta.kubernetes.io/oci-load-balancer-shape-flex-min: "10"
          service.beta.kubernetes.io/oci-load-balancer-shape-flex-max: "100"
          service.beta.kubernetes.io/oci-load-balancer-subnet1:
               "ocid1.xxxxx.xxx.xx.xxxxxxxxx.............speygundxpjuhu23lorqq"
               oci.oraclecloud.com/oci-load-balancer-listener-ssl-config:
               '{"CipherSuiteName":"oci-default-http2-tls-12-13-ssl-cipher-suite-v1",
               "Protocols":["TLSv1.2","TLSv1.3"]}'
          service.beta.kubernetes.io/oci-load-balancer-ssl-ports: "443"
     loadBalancer:
          exposedPort: 443
          useDualTls: "False"
          certSecret:
               name: "scm-lb-ssl"
               customMetadata: {}
               customLabels: {}
               customAnnotations: {}
          selfSignedCert:
               country: "US"
               state: "California"
               locality: "San Francisco"
               organization: "Oracle Corporations"
               commonName: "oracle.com"
          customCert:
                    certificatePath: "ssl/lb.crt"
                    keyPath: "ssl/lb.key"
```

Note: If you updated the `values.yaml` that you pulled from the container registry, you can push the updated SCM Helm chart in to the container registry after updating the `values.yaml` file as follows:

```
tar -zcf cloudmanager_CM_updated_<releaseVersion>.tgz
helm push cloudmanager_CM_updated_<releaseVersion>.tgz oci://<registry>/
<repositoryPath>
```

The variables in the example have the following values:

- `<registry>` is the container registry basename.
- `<repositoryPath>` is the SCM Helm chart (`cloudmanager`) repository path.
- `<releaseVersion>` is the SCM release version.

Installing SCM

This section describes the steps to install SCM on a Kubernetes cluster on premises or in the cloud or in your data center on OC3 using Helm.

To install SCM using Helm:

1. Go to the SCM Helm chart directory and run the Helm install command as follows:
   ```
   cd cloudmanager
   helm install <releaseName> . -n <namespace> --timeout 30m
   ```
   The variables in the example have the following values:
   - `<releaseName>` is the SCM Helm chart instance identifier.
   - `<namespace>` is the name of the namespace to install SCM in.

   > **Note:** If you didn't include `user` and `password` details in the `siebelregistry` section of `values.yaml`, you must pass these details in the `helm install` command as follows:
   > ```
   > helm install <chartname> . -n <namespace> --set siebelregistry.user=<username> --set
   > siebelregistry.password='<password>'
   > ```

2. Verify that the SCM pod is running and fetch the endpoint URL for SCM using the following command:
   ```
   kubectl get pods -n <namespace>
   ```
3. Build the SCM application URL (when the service type is `NodePort`) as follows:
   a. Get a node IP address:
      ```
      kubectl get nodes –wide
      ```
      > **Note:** The SCM application port is mapped to all active nodes, hence any node IP can be used to build the SCM application URL. You can copy the external IP (if available) or the internal IP as per your Kubernetes configuration.

   b. Get the assigned node port number from the service (Port Range 30000 – 32767):
      ```
      kubectl get svc/scm-app-service -n <namespace>
      ```
   c. Build the SCM application URL using the node IP address and node port as follows:
      ```
      https://<nodeIPAddress>:<nodePortNumber>
      ```
      The variables in the example have the following values:
      - `<nodeIPaddress>` is any active node IP address.
      - `<nodePortNumber>` is the assigned node port number.

   > **Note:** When the `serviceType` is set to `LoadBalancer`, build the SCM application URL as follows:
   > a. Get the external IP and port:
   > ```
   > kubectl get svc -n <namespace>
   > ```
   > b. Build the SCM application URL using the external IP and port number as follows:
   > ```
   > https://<externalIP>:<PortNumber>
   > ```

4. Access the SCM application URL and verify that the swagger page is loading correctly.

Migrating (Lift-And-Shift) Existing Siebel CRM Deployments

The Siebel Lift utility is a command-line utility, developed in Python that is used for migrating an existing Siebel CRM deployment to the cloud or any CNCF certified Kubernetes cluster. The Siebel Lift utility performs the following tasks:

- Creates deployment artifacts from your existing on-premises Siebel CRM deployment and stores them in a staging location.
- Reads the stored artifacts and uploads them to OCI Object Storage to populate the migration pipeline for your Siebel CRM deployment on OCI.

The Siebel Lift utility is available on MOS as Siebel CRM 26.8.0.0 Lift Utility (Patch).

You can use a shared network file storage or an OCI object storage with the Lift utility. For more information, see the "Using a Shared Network File System During Lift-And-Shift" section of *Deploying Siebel CRM using SCM*.

Greenfield deployments of Siebel CRM don't use the Siebel Lift utility.

For details about this Lift utility, see *Downloading and Running the Siebel Lift Utility*. References to OCI Object Storage in the "Downloading and Running Lift Utility" section are applicable primarily when deploying Siebel CRM on OCI or OC3. A shared NFS can be used whenever applicable, see the "Using a Shared Network File System During Lift-And-Shift" section of *Deploying Siebel CRM using SCM*.

Deploying Siebel CRM using SCM

This topic describes the steps to deploy Siebel CRM on a Kubernetes cluster on premises or in the cloud or in your data center on OC3, using BYOR through SCM.

This section includes the following information:

- *Before Deploying Siebel CRM on a Kubernetes Cluster*
- *Deploying Siebel CRM using SCM on a Kubernetes Cluster*
- *Using a Shared Network File System During Lift-And-Shift*

Before Deploying Siebel CRM on a Kubernetes Cluster

Before you deploy Siebel CRM on a Kubernetes cluster, you must complete the following tasks:

1. Ensure the following resources that are required for Siebel CRM deployment are available for smooth deployment of Siebel CRM:
   a. Siebel CRM file system: For the Siebel CRM file system, you can create a NFS share or use the NFS share that you created as part of the prerequisites.
   b. Database: Make a note of the wallet path referenced inside the SCM pod and the TNS connect string alias value. These details will be used in the payload for deploying Siebel CRM on a Kubernetes cluster. In case of TNS without TLS:
      i. Create a `tnsnames.ora` file with the database details in a directory named `wallet`.
      ii. Copy the `wallet` directory to the SCM NFS share `config` directory. For example,

```
<nfsServer>:/<nfsPath>/<namespace>/config/wallet
```

c. Container registry: Make a note of the following details of the container registry that stores the SCM utilities and Siebel CRM image:
   - The registry URL
   - The registry user name and password
   - The registry prefix.

d. Kubernetes cluster: You can use the same cluster you used to deploy SCM, to deploy Siebel CRM. Copy the `Kubeconfig` file of the cluster to the pod or make it accessible to the pod. Make a note of this path, it will be used in the payload for Siebel CRM deployment. For example,

```
<nfsServer>:/<nfsPath>/<namespace>/config/kubeconfig
```

e. Git instance: Make a note of the Git instance details (IP address or hostname, username, access token and root certificate). Copy the certificate to the pod or make it accessible to the pod, this path will be used in the payload for Siebel CRM deployment. For example,

```
<nfsServer>:/<nfsPath>/<namespace>/config/rootCA.crt
```

2. Optionally, you can also generate a custom certificate and use it in the Siebel CRM deployment payload. For more information on generating the custom certificate, see *Using Custom Keystore*.
3. Prepare the Siebel File System and migration directories. For more information, see *Preparing Siebel File System and Migration Directories*.

Deploying Siebel CRM using SCM on a Kubernetes Cluster

After completing the prerequisite tasks for deploying Siebel CRM and ensuring that all resources are available, you can use SCM to deploy Siebel CRM on a Kubernetes cluster. You must prepare a suitable payload and then execute this payload on SCM.

To deploy Siebel CRM on a Kubernetes cluster:

1. Prepare the payload for deployment as follows:
   - Sample payload for creating a Greenfield Siebel CRM deployment on an OCNE cluster with observability:

     Note: You must update the parameters according to your local configuration and prepare the payload.

```
{
     "name":"demo",
     "siebel": {
          "registry_url": "<container_registry_url>",
          "registry_user": "<container_userName>",
          "registry_password": "<container_registry_userPassword>",
          "registry_prefix": "<container_registry_prefix>",
          "database_type": "Vanilla",
          "industry": "Telecommunications",
          "use_self_signed_certificate": true
     },
     "infrastructure": {
          "git": {
               "git_type": "gitlab",
               "gitlab": {
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<gitlab_token>",
                    "git_user": "root",
                    "git_selfsigned_cacert": "/home/opc/certs/rootCA.crt"
               }
          },
          "kubernetes": {
          "kubernetes_type": "BYO_OCNE",
          "byo_ocne": {
```

```
            "kubeconfig_path": "/home/opc/config/kubeconfig"
            "byo_ns": true
        }
    },
    "ingress_controller": {
        "ingress_service_type": "NodePort"
        },
    "mounttarget_exports":{
        "siebfs_mt_export_paths": [
        {
            "mount_target_private_ip": "<nfsServer>",
            "export_path": "<nfsPath>"
        }
        ],
        "migration_package_mt_export_path": {
            "mount_target_private_ip": "<nfsServer>",
            "export_path": "<nfsPath>"
            }
        }
    },
    "database": {
        "db_type": "BYOD",
        "byod": {
            "wallet_path": "/home/opc/config/wallet",
            "tns_connection_name": "<TNS_connect_string>"
        },
        "auth_info": {
            "table_owner_password": "<tableOwnerUserPassword>",
            "table_owner_user": "<tableOwnerUser>",
            "default_user_password": "<plainTextPWD>",
            "anonymous_user_password": "<plainTextPWD>",
            "siebel_admin_password": "<plainTextPWD>",
            "siebel_admin_username": "<adminUser>"
        }
    },
    "observability": {
        "siebel_monitoring": true,
        "siebel_logging": true,
        "enable_oracle_opensearch": true,
        "prometheus": {
            "storage_class_name": "local-storage",
            "local_storage_info": {
                "local_storage": "/mnt/test",
                "kubernetes_node_hostname": "<hostname>"
            }
        },
        "oracle_opensearch": {
            "storage_class_name": "local-storage",
            "local_storage_info": [
            {
                "local_storage": "/mnt/test1",
                "kubernetes_node_hostname": "<hostName>"
            },
            {
                "local_storage": "/mnt/test2",
                "kubernetes_node_hostname": "<hostName>"
            },
            {
                "local_storage": "/mnt/test3",
                "kubernetes_node_hostname": "<hostName>"
            }
            ]
    },
        "monitoring_mt_export_path": {
            "mount_target_private_ip": "<mountTargetIPAddress>",
            "export_path": "/olcne-migration"
```

```
            }
        }
}
```

> **Note:** You can create a similar payload for deploying Siebel CRM on any other CNCF certified Kubernetes cluster by setting the parameter `kubernetes_type` to `BYO_OTHER` and updating the parameter values accordingly.

> **Note:**
>
> - When using an existing namespace (that is when `byo_ns` is set to `true`), you must ensure that the existing namespace name matches the name field in the deployment payload.
> - By default, SCM enables `NetworkPolicy` generation for SCM-managed CRM workloads through the `network_policy.enabled` parameter. For more information on network policies, see *Using Network Policies*.

- Sample payload for creating a Greenfield Siebel CRM deployment on a Kubernetes cluster set up in OC3 with observability:

> **Note:** You must update the parameters according to your local configuration and prepare the payload.

```
{
     "name": "demo",
     "siebel": {
          "registry_url": "<container_registry_url>",
          "registry_user": "<container_userName>",
          "registry_password": "<container_registry_userPassword>",
          "registry_prefix":"<container_registry_prefix>",
          "database_type": "Vanilla",
          "industry": "Telecommunications",
          "use_self_signed_certificate": true
     },
     "infrastructure": {
          "git": {
               "git_type": "gitlab",
               "gitlab": {
                    "git_url": "https://<IP address>",
                    "git_accesstoken": "<gitlab_token>",
                    "git_user": "root",
                    "git_selfsigned_cacert": "/home/opc/config/rootCA.crt"
               }
          },
          "kubernetes": {
               "kubernetes_type": "BYO_OKE",
               "byo_oke": {
               "oke_cluster_id": "ocid1.xxx",
               "oke_endpoint": "PUBLIC"
               }
          },
          "ingress_controller": {
               "ingress_service_type": "loadbalancer",
               "ingress_controller_service_annotations": {
                    "oci.oraclecloud.com/load-balancer-type": "lb",
                    "service.beta.kubernetes.io/oci-load-balancer-internal": "false",
                    "service.beta.kubernetes.io/oci-load-balancer-shape": "flexible",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-min": "11",
                    "service.beta.kubernetes.io/oci-load-balancer-shape-flex-max": "105",
                    "service.beta.kubernetes.io/oci-load-balancer-subnet1":
"ocid1.subnet.oc1.amaaaaaa2x5pucianjvte3dymz2",
```

```
                    "service.beta.kubernetes.io/oci-load-balancer-tls-secret": "lb-tls-
certificate",
                    "service.beta.kubernetes.io/oci-load-balancer-ssl-ports": 443,
                    "oci.oraclecloud.com/oci-load-balancer-listener-ssl-config": "{
                         \"CipherSuiteName\":\"oci-default-http2-tls-12-13-ssl-cipher-suite-v1\",
                         \"Protocols\":[\"TLSv1.2\",\"TLSv1.3\"]}"
               }
          },
          "mounttarget_exports": {
               "siebfs_mt_export_paths": [
               {
                    "mount_target_private_ip": "<NFS server name/IP>",
                    "export_path": "<nfs-path>"
               }
               ],
               "migration_package_mt_export_path": {
                    "mount_target_private_ip": "<NFS server name/IP>",
                    "export_path": "<nfs-path>"
               }
          }
     },
     "database": {
          "db_type": "BYOD",
          "byod": {
               "wallet_path": "/home/opc/config/wallet",
               "tns_connection_name": "<TNS connect string>"
          },
          "auth_info": {
               "table_owner_password": "<Plain Text PWD>",
               "table_owner_user": "<e.g. siebel>",
               "default_user_password": "<Plain Text PWD>",
               "anonymous_user_password": "<Plain Text PWD>",
               "siebel_admin_password": "<Plain Text PWD>",
               "siebel_admin_username": "<e.g. SADMIN>"
          }
     }
}
```

> **Note:** For more information on the parameters in the deployment payload, see *Payload Parameters for Siebel CRM Deployment*.

2. Submit the payload using the `environment` API through a POST request as follows:
   `POST https://<IPAddress>:<PortNumber>/scm/api/v1.0/environment`

   The variables in the example have the following values:

   - `<IPaddress>` is any active IP address.
   - `<PortNumber>` is the assigned port number.

3. Check the status of the workflow using the GET API, self-link for the same will be available in the POST request response. You must ensure that the:
   - Environment status of the Siebel CRM deployment using SCM is "completed".
   - Status of all stages is "passed".
   - Siebel CRM URLs are available in the GET API response once the workflow is complete.

   For information on troubleshooting, see *Troubleshooting Siebel CRM Deployment*.

Using a Shared Network File System During Lift-And-Shift

When you lift an existing Siebel CRM environment and deploy it on premises, you can specify the NFS server endpoint and NFS share directory that contain the lifted Siebel CRM artifacts. You can provide these values in the `nfs` section of the `siebel` block.

To use the `nfs` section in the payload:

> **Note:** You must not configure the `nfs` section if you have configured the `bucket_url` parameter in the Siebel CRM deployment payload.

1. Place all artifacts generated by the Siebel Lift utility in an NFS share directory that the Kubernetes cluster can access and read.
2. Include the `nfs` section in the `siebel` block, as follows

```
"siebel": {
    "registry_url": "<Container_registry_URL>",
    "registry_user": "<Container_user_name>",
    "registry_password": "<Container_registry_password>",
    "registry_prefix": "<Container_registry_prefix>",
    "nfs": {
        "server": "<nfsServer>",
        "path": "<nfsServerPath>"
    }
}
```

   The variables in the example have the following values:

   - `<nfsServer>` is the NFS server endpoint that the Kubernetes cluster running SCM can access.
   - `<nfsServerPath>` is the NFS share directory that contains the complete set of lifted Siebel CRM artifacts.

   For more information, see *Downloading and Running the Siebel Lift Utility*.

Updating SCM Configuration using Helm

This topic describes the steps to update the SCM configuration using Helm.

To update the SCM configuration:

1. Go to the SCM Helm chart directory.
2. Update the `values.yaml` file as required. For more information on updating the `values.yaml` file, see *Installing SCM using Helm*.
3. Run the Helm upgrade command as follows:

```
cd cloudmanager
helm upgrade <releaseName> . -n <namespace>
```

   The variables in the example have the following values:

   - `<releaseName>` is the SCM Helm chart instance identifier
   - `<namespace>` is the SCM namespace.
4. Verify that the SCM pod is running after updating the configuration as follows:

```
kubectl get pods -n <namespace>
```

In the example, `<namespace>` is the SCM namespace.

Reinstalling SCM using Helm

This topic describes how to reinstall the same version of SCM on a Kubernetes cluster using Helm.

To reinstall SCM using Helm:

1. Backup the SCM NFS directory specified in the `nfsPath` parameter of the `Storage` section of the `values.yaml` file.
2. Uninstall SCM. For more information, see *Uninstalling SCM using Helm*.
3. Install SCM using Helm. For more information, see *Installing SCM using Helm*.

Updating SCM using Helm

This topic describes how to update SCM on a Kubernetes cluster using Helm.

To update to a new version of SCM using Siebel installer:

1. Download and run the required version of Siebel Installer for SCM. For more information on downloading and running the Siebel Installer build, see *Downloading and Running Siebel Installer for SCM*.

   > **Note:** If you choose the same installation directory, then Siebel Installer will rename the existing installation directory as `<installationDirectoryName>_pre_<timestamp>` and then creates a new installation directory with the same name.

2. Go to the SCM Helm chart directory and make the following configuration changes in the `values.yaml` file:
   - In the `image` section, update the tag parameter with the new SCM build version.

     > **Note:** The structure of the image section in the `values.yaml` has been updated in release 25.9. If you are upgrading from an SCM version earlier than 25.9, please ensure that you use the updated `values.yaml`.

   - Update all other sections with the values from the previous SCM version's `values.yaml` file. For more information, see *Installing SCM using Helm*.

     > **Note:** You'll find the previous SCM version's `values.yaml` file in the:
     > - Renamed installation directory, if you've given the same installation directory that was used for the previous version of SCM when running Siebel Installer.
     > - Previous SCM version's chart directory, if you've provided a new installation directory when running Siebel Installer for the update.

3. Run the Helm upgrade command as follows:

   ```
   cd cloudmanager
   helm upgrade <releaseName> . -n <namespace>
   ```

   The variables in the example have the following values:
   - `<releaseName>` is the SCM Helm chart instance identifier.

- `<namespace>` is the SCM namespace.

4. Verify that the SCM pod is running after upgrading to the new version.
5. After updating SCM to the new version, run the following command to enable the newly introduced SCM features:

```
kubectl exec -it <scm_pod_name> -n <scm_namepsace> -- bash
cd /home/opc
bash siebel-cloud-manager/scripts/cmapp/migration.sh
```

Choose one of the options presented by the `migration.sh` script. Run the script multiple times, as necessary, for all of the options you require.

> **Note:** You might see GitLab merge conflict errors during migration. In such cases, fix the conflicts manually and try again.

Uninstalling SCM using Helm

This topic describes the steps to uninstall SCM on a Kubernetes cluster using Helm.

To uninstall SCM using Helm:

1. List the installed charts as follows:

```
helm list -n <namespace>
```

2. Uninstall all the charts as follows:

```
helm uninstall <releaseName> -n <namespace>
```

The variables in the example have the following values:

- `<releaseName>` is the SCM Helm chart instance identifier.
- `<namespace>` is the SCM namespace.

Troubleshooting Siebel CRM Deployment

This section provides additional details for troubleshooting Siebel CRM deployment issues.

This topic includes the following sections:

- *Troubleshooting Image Pull Issues*
- *Troubleshooting Helm Chart Uninstallation Issues*

Troubleshooting Image Pull Issues

When you're unable to pull the image and secret creation job fails with the `ImagePullBackoff` error. The following are the likely causes and corrective actions:

Causes	Corrective Actions
The user does not have access to pull images, or incorrect credentials (username and password) are specified in the **`siebelregistry`** section.	Run the **`helm update`** command by updating the correct username and password in the **`siebelregistry`** section.
Image isn't pushed to the URL inthe **`siebelregistry`** section of **`values.yaml`** file.	Run the installer again or update the **`url`** parameter in the **`siebelregistry`** section of **`values.yaml`** file with the correct registry URL.

Troubleshooting Helm Chart Uninstallation Issues

You're unable to uninstall Helm chart. This could be because the clean-up job got stuck as it's unable to pull the image. In this case, delete the namespace to remove all the resources inside the namespace including the Helm references for the releases and try uninstalling again.

Deploying Siebel CRM on OpenShift using SCM

This topic describes the procedure for deploying Siebel CRM on RedHat OpenShift using Siebel Installer for SCM. It contains the following sections:

- *High-Level Steps for Deploying Siebel CRM on OpenShift*
- *Prerequisites for Deploying Siebel CRM on OpenShift*
- *Installing SCM on OpenShift*
- *Deploying Siebel CRM on OpenShift using SCM*
- *Installing Siebel Monthly Update in a Siebel CRM Environment Deployed on OpenShift with Security Hardening Using SCM*

You can deploy Siebel CRM on OpenShift using Siebel Installer for SCM. OpenShift Container Platform is a cloud-based Kubernetes container platform that enables you to develop, deploy, and manage containerized applications. Developed by RedHat, OpenShift Container Platform through its partnership with Oracle, supports running cluster workloads on OCI.

For more information, refer the *OpenShift Container Platform* documentation.

> **Note:** When deploying Siebel CRM on OpenShift using Siebel Installer for SCM, SCM doesn't manage the infrastructure resources. It supports this deployment only through the "Bring Your Own Resources (BYOR)" option.

High-Level Steps for Deploying Siebel CRM on OpenShift

Here are the high-level steps for deploying Siebel CRM on OpenShift:

1. Install the prerequisite tools and make the prerequisite resources available with necessary connectivity and access provisions.
2. Download the Siebel Installer media zip files from My Oracle Support (MOS) and extract.
3. Run the installer and complete the setup and configuration.
4. Verify that the SCM image, images of SCM dependencies, and SCM Helm chart are available in your container registry.

5. Apply the OpenShift Security Context Constraints (SCCs) to the service account in the SCM namespace. SCCs are OpenShift resources that work as a security mechanism, controlling how pods run — specifically, what actions they're allowed to perform and which system resources they can access.
6. Install SCM using Helm.
7. Apply the SCCs to the service accounts in the Siebel CRM namespace.
8. Deploy Siebel CRM on OpenShift using SCM APIs.
9. Verify Siebel CRM deployment on OpenShift.

Prerequisites for Deploying Siebel CRM on OpenShift

You'll need the following for successfully deploying Siebel CRM on OpenShift:

- OpenShift cluster
- Git repositories to store SCM templates, release YAMLs, and Helm charts.
- Other prerequisites required for deploying Siebel CRM on a Kubernetes cluster. For more information, see *Prerequisites for Deploying Siebel CRM on a Kubernetes Cluster*.

Points to note:

- Service accounts will require elevated privileges on specific namespaces to ensure that pods cannot run with excessive permissions. This is critical for securing the cluster.
  - The namespaces require SCCs for specific workloads that cannot operate within restrictive constraints (such as nonroot-v2, restricted-v2, or anyuid). For example, SCM requires elevated permissions to allow containers to:
    - Run Podman commands.
    - Collect host logs. You must grant the `hostmount-anyuid` SCC to the `node-logs-collector` service to securely access and collect system logs from the host. Similarly, you must assign the `node-exporter` SCC to monitoring tools like `node-exporter`.

    Details of the SCCs to be assigned to the service accounts are listed in the *Before Installing SCM on OpenShift* and *Before Deploying Siebel CRM on OpenShift* sections later in the chapter.

  All other workloads are deployed using the most restrictive SCCs possible to limit privileges.
- SCM currently does not handle the setup of OpenShift-native ingress configurations, such as Route objects.

Installing SCM on OpenShift

This section describes the steps for installing SCM on OpenShift using Siebel Installer for SCM. It includes the following steps:

- *Downloading and Running Siebel Installer for SCM*
- *Before Installing SCM on OpenShift*
- *Installing SCM using Helm*

Downloading and Running Siebel Installer for SCM

You must download Siebel Installer for SCM from MOS and run the installer to lay down the SCM image, images of SCM dependencies, and SCM Helm chart that are required for installing SCM on OpenShift. For more information, see *Downloading and Running Siebel Installer for SCM*.

Before Installing SCM on OpenShift

You must perform the following preinstallation tasks:

- Ensure you have access to the OpenShift cluster through command line.
- Ensure that the SCM container image, images of SCM dependencies, and Helm chart are available in the user's container registry.
- Update the `securityContext` section in the SCM Helm chart `values.yaml` file as follows:

```
securityContext:
    runAsUser: <uid>
    runAsGroup: 0
    fsGroup: <fs_group_id>
```

  In the above example:

  - `<uid>` is the user ID allowed for the SCM namespace.
  - `<fs_group_id>` is the supplemental group ID allowed for the SCM namespace.

> **Note:** On OpenShift, `runAsUser` must be within the range specified by the `openshift.io/sa.scc.uid-range` namespace annotation. The `fsGroup` value must be the first ID in the range specified by the `openshift.io/sa.scc.supplemental-groups` namespace annotation. You can view these namespace annotations, as follows:
>
> ```
> oc get namespace <scm_namespace> -o jsonpath='{"openshift.io/sa.scc.uid-range
> \t"}{.metadata.annotations.openshift\.io/sa\.scc\.uid-range}{"\n"}{"openshift.io/
> sa.scc.supplemental-groups\t"}{.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}
> {"\n"}'
> ```
>
> You must also create a base directory in the SCM mount path. The name of the directory must match the SCM namespace name in uppercase. You must assign ownership to the user ID specified in `securityContext.runAsUser`, and assign group ownership to `0`, as follows.
>
> ```
> sudo mount -t nfs -o nfsvers=3,nosuid,resvport -o nolock <mount target ip>:/<export path> <path
>  on your host to mount>
> cd <path on your host to mount>
> sudo mkdir <SCM_NAMESPACE>
> sudo chown -R <uid>:0 <SCM_NAMESPACE>
> sudo chmod -R g+rwx <SCM_NAMESPACE>
> ```

- Assign elevated privileges to the service account in the SCM namespace by applying the following SCCs:

> **Note:** This task applies only if you do not configure the `securityContext` section in the SCM Helm chart `values.yaml` file.

```
oc adm policy add-scc-to-user privileged -z scm-service-account -n <scm_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z scm-service-account -n <scm_namespace>
```

  In the above example, `<scm-namespace>` is the name of the SCM namespace you created for SCM installation.

> **Note:** When you configure the `securityContext` section in the SCM Helm chart `values.yaml` file, you must not apply SCCs. In this configuration, SCM runs by using the `restricted-v2` SCC assigned by OpenShift.

Installing SCM using Helm

You can install SCM on OpenShift using Helm. For more information, see *Installing SCM using Helm*.

Optionally, you can validate the SCM instance after the installation is complete. For example:

```
https://<CM_Instance_IP>:<Port>/scm/api/v1.0/ping
```

If you encounter issues accessing SCM, such as connection refused or timeout errors, review the ingress network policies in the SCM namespace and update them as needed.

Deploying Siebel CRM on OpenShift using SCM

This topic describes how to deploy Siebel CRM on OpenShift using the BYOR option using SCM. It includes the following sections:

- *Before Deploying Siebel CRM on OpenShift*
- *Deploying Siebel CRM using SCM*

Before Deploying Siebel CRM on OpenShift

You must apply the required Security Context Constraints (SCCs) before deploying Siebel CRM by using SCM. The SCCs that you apply depend on the security mode that you choose:

- **Enhanced security** applies when you specify the `security_context` section in the Siebel CRM provisioning payload. In this mode, Siebel CRM workloads run with the custom non-root user ID and group ID defined in the payload. You must apply only the SCCs required for workloads that cannot run with the OpenShift `restricted-v2` SCC. To apply SCCs for enhanced security:
  a. Create a Siebel CRM namespace. You must ensure that the Siebel CRM namespace matches the value of the `env_name` parameter specified in the environment provisioning payload.
  b. Create the custom `nonroot-builder` SCC that allows image build workloads to run with a non-root user while providing the permissions required to build and manage container images. The `nonroot-builder` SCC is required by the `image-builder-sa` service account when the image builder runs without privileged mode. A cluster administrator must create this SCC (`nonroot-builder-scc.yaml`) once per OpenShift cluster:

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
    type: RunAsAny
groups: []
kind: SecurityContextConstraints
metadata:
    name: nonroot-builder
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
runAsUser:
    type: RunAsAny
seLinuxContext:
    type: MustRunAs
```

```
supplementalGroups:
    type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

c. Apply the SCC as follows:

```
oc apply -f nonroot-builder-scc.yaml
```

> **Note:** If you deploy Siebel CRM in multiple namespaces, create the `nonroot-builder` SCC only once at the cluster level. However, you must bind the SCC to the required service accounts in each namespace where Siebel CRM is deployed, as follows:
>
> ```
> oc -n <siebel_namespace> adm policy add-scc-to-user nonroot-builder -z image-builder-sa
> oc -n <siebel_namespace> adm policy add-scc-to-user nonroot-v2 -z image-automation-controller
> oc -n <siebel_namespace> adm policy add-scc-to-user nonroot-v2 -z image-reflector-controller
> oc -n <siebel_namespace> adm policy add-scc-to-user nonroot-v2 -z notification-controller
> oc -n <siebel_namespace> adm policy add-scc-to-user nonroot-v2 -z source-controller
> oc -n <siebel_namespace> adm policy add-scc-to-user anyuid -z <siebel_namespace>-metacontroller
> oc -n <siebel_namespace> adm policy add-scc-to-user anyuid -z <siebel_namespace>-opensearch
> oc -n <siebel_namespace> adm policy add-scc-to-user node-exporter -z node-exporter
> oc -n <siebel_namespace> adm policy add-scc-to-user hostmount-anyuid -z node-logs-collector
> oc -n <siebel_namespace> adm policy add-scc-to-user anyuid -z siebel-metric-exporter
> ```

> **Note:** Service accounts that are not explicitly granted an SCC use the OpenShift `restricted-v2` SCC by default. This SCC provides the least privileged execution model and is the recommended option for workloads that do not require additional permissions.

- **Default security** applies when you do not specify the `security_context` section in the Siebel CRM provisioning payload. In this mode, Siebel CRM workloads run with the default runtime user ID and group ID. You must apply the default SCCs described in this section to the service accounts in the Siebel CRM namespace:
  a. Create a Siebel CRM namespace. You must ensure that the Siebel CRM namespace matches the value of the `env_name` parameter specified in the environment provisioning payload.
  b. Apply the following SCCs to the service accounts in the Siebel CRM namespace:

```
oc adm policy add-scc-to-user nonroot-v2 -z image-automation-controller -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z image-reflector-controller -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z notification-controller -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z source-controller -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z <siebel_namespace>-traefik -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z list-svc-sa -n <siebel_namespace>
oc adm policy add-scc-to-user privileged -z list-svc-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z get-and-create-configmaps -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z ses-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z sai-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z smc-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z cgw-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z connectivity-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z sfs-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z persistence-cleanup-sa -n <siebel_namespace>
```

```
oc adm policy add-scc-to-user anyuid -z image-builder-sa -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z prometheus-adapter -n <siebel_namespace>
oc adm policy add-scc-to-user anyuid -z prometheus -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z prometheus-alertmanager -n <siebel_namespace>
oc adm policy add-scc-to-user anyuid -z siebel-metric-exporter -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z kube-state-metrics -n <siebel_namespace>
oc adm policy add-scc-to-user node-exporter -z node-exporter -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z oracle-db-monitoring -n <siebel_namespace>
oc adm policy add-scc-to-user anyuid -z <siebel_namespace>-opensearch -n <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z <siebel_namespace>-opensearch-dashboards -n
 <siebel_namespace>
oc adm policy add-scc-to-user nonroot-v2 -z log-aggregator -n <siebel_namespace>
oc adm policy add-scc-to-user hostmount-anyuid -z node-logs-collector -n <siebel_namespace>
```

In the above example, `<siebel_namespace>` is the name of the Siebel CRM namespace you created in the previous step.

Note: Oracle recommends adopting the enhanced security model with custom non-root user ID and group ID at the earliest opportunity. This approach provides improved security and aligns with future platform requirements. Support for the default security model is planned to be withdrawn in a future release.

Configuring Siebel File System Directories for a Custom Non-Root User

Before you deploy Siebel CRM on OpenShift with a custom non-root user ID, you must mount the Siebel File System on a host that can access the mount target. You must mount both the Siebel File System export path and the migration export path. This lets you create the required base directories and set the correct ownership.

For more information, see *Preparing Siebel File System and Migration Directories*.

Note: If you use multiple Siebel File Systems or migration file systems, repeat this procedure for each file system.

Deploying Siebel CRM using SCM

You can deploy Siebel CRM on OpenShift by following the same procedure you use to deploy Siebel CRM on a Kubernetes cluster. However, OpenShift deployments require the **BYO namespace** option.

In the environment provisioning payload:

- Set the `byo_ns` parameter to `true` to use the BYO namespace option.
- Set `kubernetes_type` to `BYO_OPENSHIFT`.
- Provide the OpenShift cluster kubeconfig path in the `byo_openshift` section.

Siebel CRM is deployed into the namespace that you create and manage.

When deploying Siebel CRM on OpenShift, you can optionally specify a custom security context in the environment provisioning payload. This configuration allows Siebel CRM and observability workloads to run with user and group IDs that are permitted by the OpenShift namespace.

To configure a custom security context in the environment provisioning payload, add the `security_context` section under:

- The `siebel` section to configure Siebel CRM workloads.
- The `observability` section to configure observability workloads, if observability is enabled.

When a custom security context is configured, the deployed workloads run with the specified user and group IDs instead of the default runtime IDs. Ensure that the configured values are allowed by the OpenShift namespace security policies.

Sample `security_context` section:

```
{
    "siebel": {
        "security_context": {
            "run_as_user": <uid>,
            "run_as_group": <gid>,
            "fs_group": <fs_group_id>
        }
    },
    "infrastructure": {
        "kubernetes": {
            "kubernetes_type": "BYO_OPENSHIFT",
            "byo_openshift": {
                "kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml",
                "byo_ns": true
            }
        }
    },
    "observability": {
        "security_context": {
            "run_as_user": <uid>,
            "run_as_group": <gid>,
            "fs_group": <fs_group_id>
        }
    }
}
```

Note:

- If you specify `security_context`, you must configure the following parameters: `run_as_user`, `run_as_group`, and `fs_group`.
- On OpenShift, `run_as_user` must be within the range specified by the `openshift.io/sa.scc.uid-range` namespace annotation. The `fs_group` value must be the first ID in the range specified by the `openshift.io/sa.scc.supplemental-groups` namespace annotation. To view these namespace annotations, execute the following command:

  ```
  oc get namespace <siebel_namespace> -o jsonpath='{"openshift.io/sa.scc.uid-range\t"}
  {.metadata.annotations.openshift\.io/sa\.scc\.uid-range}{"\n"}{"openshift.io/sa.scc.supplemental-groups
  \t"}{.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}{"\n"}'
  ```

  You must use a value from the `uid-range` annotation for `run_as_user`. Use the first ID from the `supplemental-groups` annotation for `fs_group`.

For more information, see *Deploying Siebel CRM using SCM* and *Using a BYO Namespace*.

For more information about the payload parameters, see *Payload Parameters for Siebel CRM Deployment*.

After Siebel CRM is deployed on OpenShift, you can install Siebel monthly updates by following the procedure described in *Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM*.

Installing Siebel Monthly Update in a Siebel CRM Environment Deployed on OpenShift with Security Hardening Using SCM

This topic describes the procedure to install the latest monthly update in a Siebel CRM environment deployed on OpenShift with security hardening using SCM.

> **Note:** These steps do not include repository upgrade steps. Repository upgrade steps are optional and are the same as the steps used for on-premises Siebel CRM deployments.

Starting with version 26.6, SCM and Siebel support custom user ID and group ID for pods. You can optionally use custom user ID and group ID to enhance the security posture of deployed environments by enabling SCM and Siebel CRM workloads to run with user and group IDs that comply with the security policies of the target Kubernetes cluster.

To install a Siebel monthly update with enhanced security for SCM and Siebel CRM workloads:

1. Complete the prerequisites for updating to version 26.6 before installing the monthly update. For more information, see *Prerequisites for Updating to Version 26.6*.
2. Apply the monthly update. For detailed steps to install a Siebel monthly update, see *Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM*.
3. (Optional) Execute the post-update security hardening steps after the environment is updated to version 26.6 or later. For more information, see *Post-Update Security Hardening*.

Prerequisites for Updating to Version 26.6

You must complete the following steps before you run the monthly update to version 26.6 on OpenShift:

1. Create a custom non-root builder SCC as follows:

   > **Note:** Perform this step only when you want to apply the OpenShift hardened-security posture for the `image-builder` service account.

```
# nonroot-builder-scc.yaml
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
    type: RunAsAny
groups: []
kind: SecurityContextConstraints
metadata:
    name: nonroot-builder
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
runAsUser:
    type: RunAsAny
```

```
seLinuxContext:
    type: MustRunAs
supplementalGroups:
    type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

2. Apply the custom SCC:

```
kubectl apply -f <path_to_nonroot-builder-scc.yaml>
```

3. Add required SCCs to Siebel and observability service accounts:

> **Note:** Existing service accounts should already have the required SCC grants from the current deployment. No additional action is needed for them. But for the monthly update to version 26.6, add the required SCC grants only for the new service accounts introduced in version 26.6.

```
oc adm policy add-scc-to-user nonroot-v2 -z ses-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z sai-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z smc-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z cgw-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z connectivity-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z sfs-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z persistence-cleanup-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-builder -z image-builder-sa -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z prometheus-alertmanager -n <namespace>
oc adm policy add-scc-to-user anyuid -z siebel-metric-exporter -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z oracle-db-monitoring -n <namespace>
oc adm policy add-scc-to-user nonroot-v2 -z log-aggregator -n <namespace>
oc adm policy add-scc-to-user hostmount-anyuid -z node-logs-collector -n <namespace>
```

Post-Update Security Hardening

You must perform the steps in this section only after the monthly update is complete and the environment runs version 26.6 or later. These hardening steps are optional. You must apply them when you want to move to the hardened-security posture for OpenShift.

To harden security for environment version 26.6 or later:

1. Complete the post-update security hardening procedure. For more information, see *Post-Update Security Hardening for SCM and Siebel Pods*.
2. Remove the privileged SCC grant from SCM:

```
oc -n <scm_namespace> adm policy remove-scc-from-user privileged -z scm-service-account
```

3. Remove unused SCC grants from the default service account:

```
oc adm policy remove-scc-from-user nonroot-v2 -z default -n <namespace>
oc adm policy remove-scc-from-user privileged -z default -n <namespace>
oc adm policy remove-scc-from-user hostmount-anyuid -z default -n <namespace>
```

4. Remove old SCC grants for Siebel and observability that are no longer required:

```
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z list-svc-sa
oc -n <siebel_namespace> adm policy remove-scc-from-user privileged -z list-svc-sa
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z default
oc -n <siebel_namespace> adm policy remove-scc-from-user hostmount-anyuid -z default
oc -n <siebel_namespace> adm policy remove-scc-from-user privileged -z default
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z get-and-create-configmaps
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z <siebel_namespace>-traefik
oc -n <siebel_namespace> adm policy remove-scc-from-user anyuid -z prometheus
```

```
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z prometheus-adapter
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z prometheus-alertmanager
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z kube-state-metrics
oc -n <siebel_namespace> adm policy remove-scc-from-user nonroot-v2 -z <siebel_namespace>-opensearch-
dashboards
```

5 Migrating Siebel CRM Deployments Using Siebel Lift Utility

About this chapter

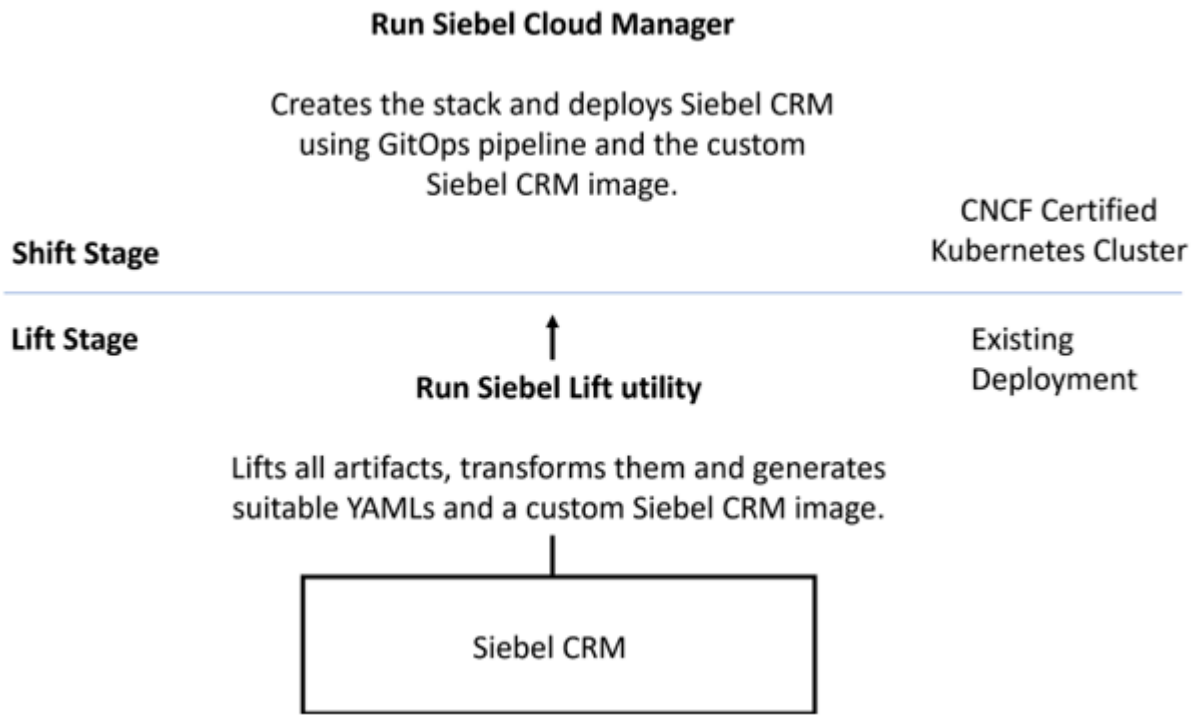
This chapter describes the procedure for migrating an existing Siebel CRM deployment to the cloud or any CNCF certified Kubernetes cluster using Siebel Lift Utility. It contains the following topics:

- *Overview*
- *Lifting Existing Siebel CRM Deployments*
- *Shifting Existing Siebel CRM Deployments*

Overview

You can migrate an existing Siebel CRM environment to any cloud or a CNCF-certified Kubernetes cluster using SCM in two simple steps:

1. Lift: In this step, you lift an existing Siebel CRM environment using the Siebel Lift utility that is available on My Oracle Support (MOS) as Siebel CRM 26.8.0.0 Lift Utility (Patch). The Siebel Lift utility:
   a. Builds deployment kits consisting of artifacts derived from your existing deployment of Siebel CRM.
   b. Moves the deployment kits to a storage location such as OCI object storage or an NFS shared directory.
2. Shift: After the Siebel Lift utility creates and uploads Siebel CRM deployment kits, you specify the deployment parameters in the deployment payload and submit a REST request to SCM to shift your environment. SCM completes the deployment of Siebel CRM according to configurations specified in the deployment payload.



Lifting Existing Siebel CRM Deployments

This topic describes the steps for lifting an existing Siebel CRM deployment using the Siebel Lift utility that is available on MOS as Siebel CRM 26.8.0.0 Lift Utility (Patch).

> **Note:** The Siebel Lift utility is only for deployments that involve performing the "lift" operations of creating and uploading deployment kits from an existing deployment of Siebel CRM. If you are performing a green field deployment of Siebel CRM, then you do not need to download, install, or use the Siebel Lift utility.

This topic includes the following sections:

- *Requirements for Siebel Lift Utility*
- *Downloading and Running the Siebel Lift Utility*
- *Lifting a Siebel CRM Environment Running on Siebel CRM Compliant Operating System*

Requirements for Siebel Lift Utility

The following are some requirements for running the Siebel Lift utility:

- You can run the Siebel lift utility on Oracle Enterprise Linux and on Microsoft Windows.
- You can run the Siebel lift utility in silent mode or in interactive mode (using a menu-driven command-line interface).

- (Optional) If your Siebel Lift utility installation method requires Python, install Python 3.14.x. Use Python 3.14.4 on Windows and a supported Python 3.14.x version on Linux 8 or Linux 9. For more information, see *Installing and Configuring Python (for Non-Container Mode)*.
- You will need software such as 7-Zip and so on for extracting the utility from the ZIP file or TAR file you downloaded.
- The user running the Siebel Lift utility must have access to the files exported by the utility, else the upload process will fail.
- The Siebel Lift utility currently supports U.S. English (ENU).
- The Siebel CRM on-premises environment must be running when you run the Siebel Lift utility on it.

  For more information, see *Downloading and Running the Siebel Lift Utility*.

Downloading and Running the Siebel Lift Utility

The Siebel Lift utility is a command-line utility, developed in Python, that is used for migrating an existing Siebel CRM deployment to the cloud or any CNCF certified Kubernetes cluster. The Siebel Lift utility performs the following tasks:

- Creates deployment artifacts from your existing on-premises Siebel CRM deployment and stores them in a staging location.
- Reads the stored artifacts and uploads them to OCI Object Storage to populate the migration pipeline for your Siebel CRM deployment on OCI.

> **Note:** You need the Siebel Lift utility only when you perform a **lift** operation, which creates and uploads deployment kits from an existing on-premises Siebel CRM deployment.If you are performing a greenfield deployment of Siebel CRM or Siebel Component Services on OCI, you do not need to download, install, or use the Siebel Lift utility.

The Siebel Lift utility is available on My Oracle Support (MOS) as Siebel CRM 26.8.0.0 Lift Utility (Patch).

After you download the Siebel Lift utility and perform preparatory steps, you run the utility to do the following:

1. Introspect, validate, and archive the required Siebel CRM artifacts from the Siebel CRM on-premises environment. This step is also referred to as creating deployment kits.

   These artifacts include application artifacts (the Siebel configuration, Web files, the Siebel File System, and others) and Siebel database artifacts. The artifacts are archived in TAR files and placed in a staging location in the on-premises environment. The archived artifact files are also referred to as deployment kits.
2. Upload the deployment kits into OCI Object Storage.

> **Note:** You can perform one or both of these operations (create and upload) in a single execution of the Siebel Lift utility, or perform creation of artifacts in one execution and upload of artifacts in another.

The staging location is where the deployment kits will be created in the required format at the time of executing the Siebel Lift utility. When you execute the utility to create deployment kits, the staging location must be empty in order to avoid any duplication. The staging location must only contain artifacts that need to be uploaded. The staging location must be accessible from the computer where you execute the utility.

You can download, install, and run the Siebel Lift utility in container mode or in non-container mode.

In either container mode or non-container mode, you can run the Siebel Lift utility in silent mode or in interactive mode. When you use silent mode, you specify a response file that you have prepared. Similarly, when you run the utility in interactive mode, you can use input files to provide values for database settings or OCI settings.

For more information about requirements for the Siebel Lift utility, see *Requirements and Limitations*.

> **Note:** If you use the tasks in *Downloading and Running the Siebel Lift Utility (Container Mode)* (Linux only), then do not perform the tasks for non-container mode.

This topic contains the following information:

- *Downloading the Siebel Lift Utility*
- *Downloading and Running the Siebel Lift Utility (Container Mode)*
- *Downloading and Running the Siebel Lift Utility (Non-Container Mode)*
- *Data Not Migrated by Siebel Lift Utility*
- *Troubleshooting Siebel Lift Utility Execution*

Downloading the Siebel Lift Utility

To download the Siebel Lift utility:

1. Log in to My Oracle Support (MOS).
2. In the **Search** field, search for and select **Patches and Product Certification Matrix**.
3. In the **Patch Search** pane, enter the search criteria:
    - In the **Platform** field, search for and select the platform, such as **Linux x86-64**.
    - In the **Language** field, search for and select the language, such as **American English**.
    - In the **Product** field, search for and select **Siebel CRM**.
    - In the **Release** field, search for and select the release version, such as **Siebel Industry Applications 26.8**.
4. Click **Apply**.

    The Patches page displays the files available for the search. For example, for installer version 26.8, the following media files are listed:

    - Siebel CRM 26.8.0.0 2026_08 (Patch)
    - Siebel CRM 26.8 OL Containers and Sample DB (Patch)
    - Siebel CRM 26.8.0.0 Cloud Manager (Patch)
    - Siebel CRM 26.8.0.0 Lift Utility (Patch)
5. In the **Patch Name** field, click the Siebel Lift Utility (Patch) link, such as **26.8.0.0 Siebel Lift Utility**.
6. Click **Download**. You receive one file, for example, `p123xxxxx_268_Linux-x86-64.zip`.

Downloading and Running the Siebel Lift Utility (Container Mode)

You can use the procedures in this topic to download and run the Siebel Lift utility using container mode. This option is recommended. It is the default option on Linux, though it can also be used on Windows, where you are running Linux containers on Windows. This topic is part of *Downloading and Running the Siebel Lift Utility*.

> **Note:** If you use the tasks in this topic (for container mode), then do not perform the tasks for non-container mode.

Note: Separate procedures are provided for running the Siebel Lift utility in silent mode or in interactive mode. It is recommended to run the Siebel Lift utility in silent mode. For information about the settings equivalent to the response file options, see *Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)*.

This topic contains the following sections:

- *Downloading the Siebel Lift Utility (for Container Mode)*
- *Running the Siebel Lift Utility in Silent Mode (for Container Mode)*
- *Running the Siebel Lift Utility in Interactive Mode (for Container Mode)*

Downloading the Siebel Lift Utility (for Container Mode)

Use the following procedure to download the Siebel Lift utility for container mode.

To download the Siebel Lift utility for container mode (Linux only)

1. Download the Siebel Lift utility media file, for example, `P123XXXXX_268_Linux-x86-64.zip`. For more information, see *Downloading the Siebel Lift Utility*.
2. Extract the package and use the `SiebelLiftUtility/container` directory.
3. Extract the contents of the ZIP file. This file contains the following contents:
   - **execute_lift_container.sh.** A shell script used to run the container.
   - **volumemounts.ini.** Contains local paths for the container volume mounts.
   - **oci_config_template.ini.** Contains OCI credentials details (for interactive mode only). To be populated only if OCI object storage is used during "lift-and-shift".
   - **oracle_db_config_template.ini.** Contains Oracle Database details (for interactive mode only).
   - **lift_utility_responsefile_template.resp.** A response file template used for silent mode execution of the utility.
   - **tnsnames.ora.** A template of the `tnsnames.ora` file, containing database connection credentials details.
   - **images/siebelifttool_<version>.tar.gz.** Contains the Siebel Lift utility container image archive.
4. Install and configure Podman, the software for managing containers.
5. Update the `volumemounts.ini` file with the required values. In particular, note the following:
   - While executing the Siebel Lift utility from the container, provide the mounted volume path of the container at the location indicated by STAGING_LOCATION (as specified in `volumemounts.ini`), such as `/liftstage`.
   - If you will run the Siebel Lift utility in silent mode, then keep the response file template (`lift_utility_responsefile_template.resp`) at the location indicated by TEMPLATES_FILE_LOCATION (as specified in `volumemounts.ini`), such as `/templates`.
   - If you will run the Siebel Lift utility in interactive mode, and you want to provide the OCI and database related configuration details via files, then keep the `oci_config_template.ini` and `oracle_db_config_template.ini` template files at the location indicated by TEMPLATES_FILE_LOCATION (as specified in `volumemounts.ini`), such as `/templates`.
   - If you need to access the log files outside the container, then also update the local mounted path value for the variable LIFT_TOOL_LOG_LOCATION (as specified in volumemounts.ini).

6. Update the `tnsnames.ora` file with the required values.

   While executing the Siebel Lift utility from the container to generate database artifacts, create a directory named `tns` inside the local mounted path for `/templates`. Copy the provided `tnsnames.ora` file to this `tns` folder and update it with the database connection details as needed.

   > **Note:** Alternatively, you can use actual database connect string details as input instead of the TNS profile name while running the Siebel Lift utility. In this case, you do not need the provided tnsnames.ora file.

7. While executing the Siebel Lift utility from the container to generate database artifacts (deployment kits), provide the database client location, which is `/usr/lib/oracle/12.2/client`.

Running the Siebel Lift Utility in Silent Mode (for Container Mode)

Use the following procedure to run the Siebel Lift utility in silent mode.

To run the Siebel Lift utility in silent mode (for container mode)

- Run a command like the following in a shell:

```
sh execute_lift_container.sh -m silent -v volumemounts.ini -f lift_utility_responsefile_template.resp -s
 <smc_password> -d <database_password> -a <DNS1:IP1>,<DNS2:IP2>
```

  In the sample command, the flag:

  - -m is the mode of the execution (silent or interactive).
  - -v is the volume mount file location.
  - -f is the response file location required for silent mode execution.
  - -s is the SMC Authenticated Password required for silent mode execution.
  - -d is the table owner password required for silent mode execution.
  - -a is the comma-separated list of DNS:IP hosts mappings. This is an optional parameter.

  > **Note:** It is recommended to run the Siebel Lift utility in silent mode. For information about the settings equivalent to the response file options, see *Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)*.

Running the Siebel Lift Utility in Interactive Mode (for Container Mode)

Use the following procedure to run the Siebel Lift utility in interactive mode.

To run the Siebel Lift utility in interactive mode (for container mode)

- Run a command like the following in a shell:

```
sh execute_lift_container.sh -m interactive -v volumemounts.ini -a <DNS1:IP1>,<DNS2:IP2>
```

  In the sample command, the flag:

  - -m is the mode of the execution (silent or interactive).
  - -v is the volume mount file location.
  - -f is the response file location required for silent mode execution.

- -s is the SMC Authenticated Password required for silent mode execution.
- -d is the table owner password required for silent mode execution.
- -a is the comma-separated list of DNS:IP hosts mappings. This is an optional parameter.

> **Note:** For more information about interactive mode, see *Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)*.

> **Note:** When running the Lift utility in container mode, make sure that the user group 1000 has the necessary access to all the volume mounts. Run the following commands to grant required access to user group 1000:
>
> ```
> chown -R 1000:1000 <all_volume_mount_folders>
> chmod -R g+rwx <all_volume_mount_folders>
> ```

Downloading and Running the Siebel Lift Utility (Non-Container Mode)

You can use the procedures in this topic to download and run the Siebel Lift utility in non-container mode.

This topic contains the following sections:

- *Downloading the Siebel Lift Utility (for Non-Container Mode)*
- *Installing and Configuring Python (for Non-Container Mode)*
- *Running the Siebel Lift Utility in Silent Mode (for Non-Container Mode)*
- *Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)*

Downloading the Siebel Lift Utility (for Non-Container Mode)

Use the following procedure to download the Siebel Lift utility for non-container mode.

Prerequisites for running Siebel Lift Utility (for Non-Container Mode):

You must have OpenSSL version 3.0.5 or higher for the encryption/decryption mechanism to work.

To download the Siebel Lift utility (for non-container mode)

1. Download the Siebel Lift utility media file, for example, `P123XXXXX_268_Linux-x86-64.zip`. For more information, see *Downloading the Siebel Lift Utility*.
2. Extract the contents of the ZIP file, and then use the `SiebelLiftUtility/non-container` directory.

Installing and Configuring Python (for Non-Container Mode)

If you are using non-container mode for downloading and running the Siebel Lift utility, then you must use one of the procedures below to install and configure Python. Do this before you run the Siebel Lift utility.

To install and configure Python on Windows (for non-container mode)

1. On Windows, you download Python from *https://www.python.org/downloads*.

   Next, you will install various Python modules with the required settings by using the appropriate command and the `requirements.txt` file for the Python installation. If you are installing a later version of Python, then make sure to use the compatible versions of the dependent modules.
2. Configure proxy settings by entering a command like the following:

   ```
   export/set https_proxy=https://proxy-name:port
   ```

For example: `https_proxy=https://www-proxy-hqdc.com:80`

3. Use a command like the following to install Python. The versions defined in the `requirements.txt` settings are compatible with the version of Python you install. Run this command from the folder in which you have extracted the siebelliftutility.zip file. Run this command from the extracted `SiebelLiftUtility/non-container` directory:

```
pip install -r requirements.txt
```

To install and configure Python on Linux (for non-container mode)

1. Install and configure Python 3.14.x from Software Collection Library (SCL). Make sure that the Python library versions specified in `requirements.txt` are compatible with your platform. For detailed instructions to install and configure Python on Oracle Linux 8, refer online documentation.

   For Oracle Enterprise Linux 9, install and configure Python 3.14.x from SCL, as follows:

   a. (Optional) If you use a proxy, add the following entry under the existing [main] section in `/etc/yum.conf`. Replace the host name and port with your proxy values.
   ```
   proxy=http://proxy.example.com:3128
   ```

   b. Enable the required repositories and install Python 3.14 and `pip`:
   ```
   sudo yum install -y dnf-plugins-core
   sudo yum config-manager --enable ol9_baseos_latest ol9_appstream
   sudo yum --refresh install -y python3.14 python3.14-pip
   python3.14 --version
   ```

   This command installs the available Python 3.14.x package, which might be newer than Python 3.14.4. If `yum` reports `No match`, make sure that your repositories or corporate mirror include Oracle Linux 9.8 or later packages.

   c. Go to the extracted `SiebelLiftUtility/non-container` directory:
   ```
   cd /path/to/SiebelLiftUtility/non-container
   ```

   d. (Optional) Set the proxy variables if curl or pip requires a proxy:
   ```
   export http_proxy="http://proxy.example.com:3128"
   export https_proxy="$http_proxy"
   ```

   e. Create and activate a Python virtual environment:
   ```
   python3.14 -m venv .venv-py314
   source .venv-py314/bin/activate
   python –version
   ```

   f. Install and verify the required Python libraries:
   ```
   python -m pip install -r requirements.txt
   python -m pip check
   ```

2. If you create Siebel database artifacts, also called deployment kits, verify that you have the required Oracle Database client version. For more information, see *Requirements and Limitations*.
3. Add Python, Python scripts, 7-Zip or another extraction tool, and the Oracle Database home to the PATH environment variable.
4. Verify the setup by running commands like the following on the OCI terminal or command prompt:
   ```
   python --version
   7z
   ```

```
oci -v
```

> **Note:** If any of these commands fail, then check the PATH variable definitions made in Step 3 of this procedure and update them as needed. If you are behind a firewall, then you might also need to modify the HTTP and HTTPS proxy settings for your environment. Set the environment variables `http_proxy` and `https_proxy` to suitable proxy servers.

Running the Siebel Lift Utility in Silent Mode (for Non-Container Mode)

Use the following procedure to run the Siebel Lift utility in silent mode.

The response file `lift_utility_responsefile_template.resp` is available in the directory where you extracted the utility. You can modify a copy of this file according to your requirements.

> **Note:** It is recommended to run the Siebel Lift utility in silent mode. For information about the settings equivalent to the response file options, see the procedure for running Siebel Lift utility in interactive mode, in *Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)*.

To run the Siebel Lift utility in silent mode (for non-container mode)

1. Modify the response file as needed.

   The response file you specify must contain settings corresponding to those described in the procedure for running the Siebel Lift utility in interactive mode.
2. Enter the following command in a command window or shell:

   ```
   python siebel_lift_utility.py -f <response_file> -sp <smc_password> -dp <table_owner_password>
   ```

   In this command, use the arguments to specify a response file, the password for Siebel Management Console (SMC), and the table owner password. The `-sp` and `-dp` flags are mandatory where SMC_CONFIGURATION and DATABASE (in the section ARTIFACT_TYPE) are set to YES in the response file.

Running the Siebel Lift Utility in Interactive Mode (for Non-Container Mode)

Use the following procedure to run the Siebel Lift utility in interactive mode.

> **Note:** It is recommended to run the Siebel Lift utility in silent mode. For more information, see *Running the Siebel Lift Utility in Silent Mode (for Non-Container Mode)*.

To run the Siebel Lift utility in interactive mode (for non-container mode)

1. Enter the following command in a command window or shell:

   ```
   python siebel_lift_utility.py
   ```

   Follow the instructions and provide the required inputs on each screen. The steps that follow describe the available options.
2. In the welcome screen, select the task you want to perform. You can choose to create deployment kits, upload deployment kits to OCI Object Storage, or both.
3. If you are creating deployment kits, then select the artifact type for creating deployment kits. You can select application tier artifacts, database tier artifacts, or both.

4. If you selected application tier artifacts, then select one or more of the following options, which are mandatory for deployment: SMC configuration profiles, Siebel File System, or custom Web files. You can also select encryption key file or other artifacts (files not included in other options).
5. If you selected SMC configuration profiles as one of the application tier artifact selections, then specify the details to allow introspection of the SMC configuration data.

   Specify the SMC host name, the SMC HTTPS port, the application context root name, the authenticated user name, and the authenticated user password. For more information about these options, see *Siebel Installation Guide*.
6. If you included Siebel File System as one of the application tier artifact selections, then specify the Siebel File System path locations in a single comma-separated value. (This value would correspond to the Siebel File System parameter in the Siebel CRM on-premises environment.)
7. If you selected custom Web files as one of the application tier artifact selections, then specify the locations of these files (custom files, custom images, and custom scripts) in the application container directory for your existing deployment.

   - For Siebel CRM 21.2 or later, specify `SIEBEL_ROOT /applicationcontainer_external` (on the Siebel Enterprise installation location where you're running Siebel Application Interface) as the parent directory for the three types of files.
   - For Siebel CRM 21.1 and earlier, specify `SIEBEL_ROOT /applicationcontainer` (on the Siebel Application Interface installation location) as the parent directory for the three types of files.

   For more information about the Web artifact locations, see *Siebel Installation Guide* for your Siebel CRM installed version.
8. If you selected database tier artifacts in Step 3 of this procedure, then select the database type. (In this release, Oracle Database is the only selection.)
9. Specify the database configuration information by doing one of the following:

   - Directly specify information such as the following:
     - **Oracle Home location.** Specify the installation location of the Oracle Client.
     - **Table owner user.** Specify the table owner user.
     - **Table owner user password.** Specify the table owner user password.
     - **TNS profile name.** Specify the TNS profile name from the tsnnames.ora file.
     - **Database directory creation flag.** (Optional) Used to map the directory location in the database. If a directory is already created or mapped by the Database Administrator, then specify N. If a directory wasn't already created, then specify Y.
     - **Number of parallel transactions.** (Optional) Used to increase the performance of this utility. This parameter depends on the CPU configuration: if there are more than one CPU, then you can specify a value greater than 1 (the default is 1). It is recommended to pass the value 16 to make the utility run faster.
     - **Database directory name.** Used to create the directory name to store the database dump files.
     - **Application user extraction flag.** (Optional) Used to extract the Siebel users and their access. Specify Y (the default) to extract all Siebel users. Specify N to extract only SADMIN and GUESTUSER.
   - Alternatively, you can prepare a database configuration file containing these settings, such as `db_config.ini`, and specify the location of this file.

10. If you are uploading deployment kits to OCI Object Storage, then specify configuration information for OCI access by doing one of the following:

    - Directly specify information such as OCI region name, OCI tenancy ID, OCI compartment ID, OCI user ID, OCI private key file location, OCI passphrase, OCI fingerprint, and OCI bucket name.

      You can find most of these settings from the Profile menu after logging into OCI. If necessary, first create API keys. Then you can download the OCI private key file to a location accessible to the Siebel Lift utility.
    - Alternatively, you can prepare an OCI configuration file containing these settings, such as `oci_config.ini`, and specify the location of this file. Structure the file like the following:

```
# <KEY=VALUE> (Do not change the KEY name. Only update the VALUE.)
[OCI_CONFIGURATION_DETAILS]
# [NOTE: An Oracle Cloud Infrastructure region. Example: us-example-1]
OCI_REGION_NAME=
# [NOTE: OCID of the tenancy. Example: ocid1.tenancy.oc1..<unique_ID>]
OCI_TENANCY_ID=
# [NOTE: OCID of the compartment. Example: ocid1.compartment.oc1..<unique_ID>]
OCI_COMPARTMENT_ID=
# [NOTE: OCID of the user. Example: ocid1.user.oc1..<unique_ID>]
OCI_USER_ID=
# [NOTE: Full path and filename of the private key. The key pair must be in PEM format. Example:
 oci_api_key.pem]
OCI_PRIVATE_KEY_FILE_LOCATION=
# [NOTE: Passphrase used for the key, if it is encrypted.]
OCI_PASSPHRASE=
# [NOTE: Fingerprint for the public key that was added to OCI user.]
OCI_FINGER_PRINT=
# [NOTE: Name of the bucket to be created or already exist inside the compartment of the OCI
 object store. The deployment kit gets upload under this bucket.]
OCI_BUCKET_NAME=
```

11. Complete and exit your Siebel Lift utility session. The deployment kits will be created, uploaded to OCI Object Storage, or both, according to your selections.

    When you create deployment kits, an `export.log` file is created, which you can review. Creating Siebel database artifacts creates a very large file named like `EXPORT_01.DMP`.

    > **Note:** Before you proceed with Siebel CRM deployment steps described in *Deploying Siebel CRM on OCI*, to make sure deployment is successful, make sure that the deployment artifacts have been successfully uploaded.

12. Review log files such as `siebel_lift_utility.log`, `setup.log`, or `createdirectory.log` to confirm successful execution or to help you troubleshoot issues you might encounter.

    Also review *Requirements and Limitations* and other information about the Siebel Lift utility. For issues in creating database artifacts, review your specified number of parallel transactions or other settings.

Data Not Migrated by Siebel Lift Utility

The Siebel Lift utility does not migrate the following data source parameters of the custom `ServerDataSrc` named subsystem:

- `DSConnectString`
- `DSTableOwner`
- `DSDLLName`
- `DSGatewayAddress`

- **DSEnterpriseServer**
- **DSOLAPServer**
- **CFGOLEAutomationDLL**
- **CFGScriptingDLL**
- **ConfiguratorDLLName**
- **AnonUser**
- **AnonPwd**

You must configure these parameters manually. For more information on how to configure named subsystem parameters, see the "Use Cases for Setting Parameters" table under the *Use Cases for Setting Parameters* section.

Troubleshooting Siebel Lift Utility Execution

This topic includes troubleshooting information for Siebel Lift utility execution. This topic is part of *Downloading and Running the Siebel Lift Utility*.

Create Directory Error

When executing `siebel_lift_utility.py`, you might encounter that the execution fails and error messages like the following are seen in the `siebel_lift_utility.log/setup.log/createdirectory.log` file.

```
2021-10-16 19:56:44,971 - __main__ - INFO - Lift Execution Started.
2021-10-17 07:54:20,106 - __main__ - INFO - Successfully captured the required inputs.
2021-10-17 07:54:20,106 - __main__ - INFO - Response File creation started.
2021-10-17 07:54:24,105 - __main__ - INFO - Response file :<_io.TextIOWrapper name='response_files\
\responsefile_17102021_075420.resp' mode='a' encoding='cp1252'> successfully created.
2021-10-17 07:54:24,120 - __main__ - INFO - Starting export DB execution.
2021-10-17 07:54:24,120 - __main__ - INFO - Export DB process is in progress. Please wait.....
2021-10-17 07:54:24,120 - __main__ - INFO - ['ExportDB.exe', '-s', 'C:\\app\\client\\Administrator\\product
\\19.0.0\\client_1\\bin', '-t', 'SIEBEL', '-p', 'SIEBEL', '-d', 'ORCL', '-f', 'C:\\CM', '-m', '', '-n', '',
 '-l', WindowsPath('C:/CM/logs')]
2021-10-17 07:54:24,135 - __main__ - ERROR - Error in executing database export utility. (<class
 'FileNotFoundError'>, FileNotFoundError(2, 'The system cannot find the file specified', None, 2, None),
 <traceback object at 0x000001A1826BAD40>)
Setup.log:
Sun Oct 17,2021 [12:20:14] : Stage 2 of 3 : Creating Directory
Sun Oct 17,2021 [12:20:16] : Stage 2 of 3 : Error during directory creation
Createdirectory.log:
CREATE OR REPLACE DIRECTORY EXPORT_DIR AS 'C:\CM'
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

Root Cause: In the above step, the `create directory` has failed because the Table Owner (TBLO) user does not have the privilege to create the directory.

Solution: Make sure to have Create Directory privilege for the TBLO user (in this case, it is SIEBEL) before running the utility. Otherwise, have the directory created by the DBA so that you can choose the Create Directory option as N and proceed. If you have the permission to grant the required privilege, then run the following command after connecting to the database:

```
grant create any directory to <TBLO user>;
```

Introspection Error Codes

When the Siebel Lift utility performs introspection on data about the topology and configuration of the Siebel CRM environment, different codes are returned, indicating a successful run or an error encountered. The following table describes the possible codes returned. If introspection failed, then rerun the introspection.

Error Codes for Siebel Lift Utility Introspection

Status	Error Code	Message
Success	0	Introspection successfully completed.
Success	2	Introspection successfully completed with trivial errors &/or warnings.
Error	1	Introspection failed. Rerun the utility.

Lifting a Siebel CRM Environment Running on Siebel CRM Compliant Operating System

This topic describes the procedure to run the Siebel Lift utility against a Siebel CRM environment hosted on another machine that has a Siebel CRM compatible operating system running on it; for example, appropriate flavors and versions of Oracle Solaris, HP-UX, IBM AIX, Linux and Windows.

> **Note:** For representational purposes this section will refer only to Oracle Solaris, but the same steps apply to all Siebel CRM compatible operating systems mentioned above.

Prerequisites for lifting a Siebel environment running on Oracle Solaris:

> **Note:** We recommend running the Siebel Lift utility in container mode on Oracle Enterprise Linux (OEL) machines to lift a Siebel CRM environment running on another Oracle Solaris machine.

- Create a shared file path for the staging location. This shared path must be accessible from both the Oracle Solaris machine on which Siebel CRM is installed and the OEL machine from where the Siebel Lift utility will be run, for example, `/net/liftsharedpath/`.
- Create the following sub folders under the shared file path to store the different artifacts:
  - A folder to store the Siebel CRM file system.
  - A templates folder to store template files such as `lift_utility_responsefile_template.resp`, and so on.
  - A folder to store custom scripts, custom files, custom images, and so on.
  - A folder to store the encryption key file.
  - An additional folder to store other file types.
- Copy the Siebel CRM artifact from the Oracle Solaris machine (running Siebel CRM) to the respective sub folders in the shared file path.

Lifting a Siebel CRM Environment Running on Oracle Solaris

To lift a Siebel CRM environment running on Oracle Solaris:

1. Download the Siebel Lift utility on an OEL machine that has access to the shared file path. For more information on downloading the Siebel Lift utility, see *Downloading the Siebel Lift Utility (for Container Mode)*.

2. Run the Siebel Lift utility on the OEL machine in container mode. For more information on running the Siebel Lift utility in container mode, see *Running the Siebel Lift Utility in Silent Mode (for Container Mode)*.

Shifting Existing Siebel CRM Deployments

After the Siebel Lift utility finishes running, it creates and uploads the deployment kits. Now, you can use these deployment kits to shift or deploy your Siebel CRM environment.

To deploy Siebel CRM:

1. Configure the deployment parameters in the deployment payload.
2. Submit the payload to SCM using a REST request.

SCM retrieves the Siebel CRM artifacts from either the object storage or the NFS shared directory and deploys Siebel CRM based on your payload configuration. You can choose to deploy Siebel CRM:

- On OCI: When deploying Siebel CRM on OCI, you can either:
  - Allow SCM to create the infrastructure resources or
  - Bring your own resources. In this case, you can bring all the required infrastructure resources, or bring some and let SCM create the remaining required infrastructure resources.

  For more information, see *About this Chapter*.
- On a CNCF certified Kubernetes cluster, such as OCNE or Red Hat Open Shift, or on Oracle Cloud Compute @Customer (OC3) in your data center. In this case, you must bring all the infrastructure resources that are required to deploy Siebel CRM. For more information, see *Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer*.

Note: Before you proceed with Siebel CRM deployment, ensure that the deployment artifacts have been successfully uploaded.

After the Siebel CRM deployment is complete, you can use SCM to manage your Siebel CRM environment.

6 Deploying Open Integration on a Kubernetes Cluster Using SCM

About this Chapter

This chapter describes how to deploy Open Integration using Siebel Cloud Manager (SCM) on a Kubernetes cluster. It contains the following topics:

- *Deploying Open Integration Using SCM*
- *Rerunning the Open Integration Deployment*
- *Deleting an Open Integration Deployment*
- *Viewing all Open Integration Deployments*
- *Performing Open Integration Incremental Updates*
- *Troubleshooting Open Integration Deployment Issues*

Open Integration provides tooling for the creation and execution of REST APIs, enabling you to expose Siebel processes externally. With Open Integration, you can provide API access to both standard and customized Siebel CRM components (such as applets, views, business services, and workflows), thereby streamlining integration with external applications and systems. For more information on Open Integration, refer the *Siebel REST API Guide* and *Siebel Installation Guide*.

You can deploy Open Integration on a Kubernetes Cluster using SCM. SCM enables you to automate the deployment, configuration, and management of services across the Kubernetes environment. It simplifies the deployment of Open Integration on a Kubernetes cluster by automating key tasks such as version control, environment setup, and configuration changes.

Open Integration uses a Git-based deployment model in SCM. Customer-specific runtime configuration is stored in a customer Git repository. During deployment, SCM retrieves the configuration by using a registered Git repository, a Git branch, and a Git directory.

To deploy Open Integration on a Kubernetes cluster, ensure that an SCM instance version 25.12 or later is available on the cluster. For more information on installing SCM on a Kubernetes cluster, see the *Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer* chapter.

Deploying Open Integration Using SCM

This section describes the steps for deploying Open Integration on a Kubernetes cluster using SCM. It includes the following sections:

- *High-Level Steps for Deploying Open Integration on a Kubernetes Cluster Using SCM*
- *1. Creating Infrastructure Resources*
- *2. Verifying Infrastructure Resources*
- *3. Registering a Customer Git Repository*
- *4. Uploading Starter Templates*

- *5. Uploading Sync Utility Files*
- *6. Preparing Open Integration Files in Git*
- *7. Deploying Open Integration using SCM*
- *8. Verifying the Open Integration Deployment Status*

High-Level Steps for Deploying Open Integration on a Kubernetes Cluster Using SCM

The following are the high-level steps for deploying Open Integration on a Kubernetes cluster using SCM:

1. Create the required infrastructure resources for deploying Open Integration by submitting a POST request to the `/infrastructure` API.
2. Verify that the infrastructure resources are successfully created and active by submitting a GET request to the `/infrastructure` API.
3. Register customer Git by submitting a POST request to the `/git_repositories` API.
4. Upload starter templates by submitting a POST request to the `/openintegration/git_template_upload` API.
5. Synchronize the configuration files with the cluster by uploading the modified files by submitting a POST request to the `/syncutilities/upload` API.
6. Prepare Open Integration in Git. Create config.json and profile.json from the starter templates, then update and commit deployment files.
7. Deploy Open Integration by submitting a POST request to the `/openintegration` API using the infrastructure resources and templates created.
8. Retrieve the deployment status using the deployment ID by submitting a GET request to the `/openintegration` API.

1. Creating Infrastructure Resources

You must create the infrastructure resources required for deploying Open Integration. This includes setting up infrastructure resources such as the Kubernetes cluster, Git repository, registry, Coherence Operator, and so on. This step initiates the creation of the infrastructure resources on which Open Integration will run. You can initiate the creation of infrastructure resources using the `/infrastructure` API:

- Method: POST
- Endpoint: `/infrastructure`
- Description: Initiates the creation of the required infrastructure resources based on the input payload and returns an infrastructure ID, which will be used during the Open Integration deployment.
- Sample API call request:

```
POST https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/infrastructure
Authorization: Basic Auth
Content-Type: application/json
```

- Request payload structure:

```
{
    "kubernetes": {
        "kubernetes_type": "BYO_OKE",
        "namespace": "<infra-namespace>",
        "byo_oke": {
                "oke_cluster_id": "<oke-cluster-ocid>",
                "oke_endpoint": "PRIVATE"
            }
        },
```

```
        "git": {
            "git_type": "byo_git",
            "byo_git": {
                "git_protocol_type": "https",
                "git_scm_repo_url": "https://<git-host>/<group>/<gitops-repo>.git",
                "git_scm_repo_branch": "<gitops-branch>",
                "git_scm_flux_folder": "<flux-folder>",
                "git_helm_repo_url": "https://<git-host>/<group>/<helm-repo>.git",
                "git_helm_repo_branch": "<helm-branch>",
                "git_user": "<git-user>",
                "git_accesstoken": "<git-token>",
                "git_selfsigned_cacert": "",
                "skip_flux_bootstrap": false
            }
        },
        "registry": {
            "registry_url": "<registry-host>",
            "registry_prefix": "<registry-prefix>",
            "registry_user": "<registry-user>",
            "registry_password": "<registry-password>"
        },
    }
```

Optional Security Context

You can optionally specify a `security_context` section at the top-level of the infrastructure provisioning payload when infrastructure resources and operators must run with custom non-root user ID and group ID. For example:

```
{
    "security_context": {
        "run_as_user": <uid>,
        "run_as_group": <gid>,
        "fs_group": <fs_group_id>
    }
}
```

If you specify `security_context`, you must configure the following parameters:

- `run_as_user`
- `run_as_group`
- `fs_group`

On OpenShift, `run_as_user` must be within the range specified by the `openshift.io/sa.scc.uid-range` namespace annotation. The `fs_group` value must be the first ID in the range specified by the `openshift.io/sa.scc.supplemental-groups` namespace annotation.

To determine valid values, inspect the namespace annotations and select a user ID from the `uid-range` annotation and the first group ID from the `supplemental-groups` annotation.

- Sample response:

```
{
   "data": {
      "infra_details": {
         "components_info": {
         "git": {
            "byo_git": {
               "git_accesstoken": "********",
               "git_helm_repo_branch": "DEMO_HELM",
               "git_helm_repo_url": "https://devops. xxxxx.oci.oraclecloud.com/namespaces/siebeldev/
projects/DEMO/repositories/DEMO_HELM",
```

```
                "git_protocol_type": "http",
                "git_scm_flux_folder": "flux1",
                "git_scm_repo_branch": "DEMO_SCM",
                "git_scm_repo_url": "https://devops.scmservice.xxxxx.oci.oraclecloud.com/namespaces/
siebeldev/projects/DEMO/repositories/DEMO_SCM",
                "git_user": "**********",
                "skip_flux_bootstrap": false
             },
          "git_type": "byo_git"
          },
             "kubernetes": {
                "byo_oke": {
                   "oke_cluster_id": "ocid1.cluster.oc1.xxxxx......",
                   "oke_endpoint": "PRIVATE"
                },
                "kubernetes_type": "BYO_OKE"
             },
             "registry": {
                "registry_password": "********",
                "registry_url": "iad.ocir.io",
                "registry_prefix": "siebel_openint",
                "registry_user": "******"
             }
          },
          "infra_id": "7OZMKQ",
          "infra_status": "creation-in-progress",
          "resource_status": {
             "git": "not-validated",
             "kubernetes": "not-validated",
             "registry": "not-validated"
          },
          "stages_info": [
          {
             "end_time": "Thu, 22 May 2025 06:54:38 GMT",
             "log_api_link": "",
             "log_location": "/home/opc/siebel/infrastructures/7OZMKQ/helm_setup.log",
             "name": "Setup Helm Chart for validation",
             "previous_status": "",
             "stage_name": "setup_validation_helmchart",
             "start_time": "Thu, 22 May 2025 06:54:27 GMT",
             "status": "passed"
          },
          {
             "end_time": "Thu, 22 May 2025 06:54:38 GMT",
             "log_api_link": "",
             "log_location": "/home/opc/siebel/infrastructures/7OZMKQ/helm_setup.log",
             "name": "Check Job status",
             "previous_status": "",
             "stage_name": "check_job_status_helmchart",
             "start_time": "Thu, 22 May 2025 06:54:27 GMT",
             "status": "in-progress"
          },
          {
             "end_time": "Thu, 22 May 2025 06:54:38 GMT",
             "log_api_link": "",
             "log_location": "",
             "name": "Collect Infra artifacts",
             "previous_status": "",
             "stage_name": "collect_infra_artifacts",
             "start_time": "Thu, 22 May 2025 06:54:27 GMT",
             "status": ""
          }
          ],
          "updated_values": {}
       }
       },
```

```
   "message": "Infrastructure details retrieved successfully.",
   "status": "success"
}
```


2. Verifying Infrastructure Resources

After creating the infrastructure resources, you must verify that all resources have been successfully created and are functioning as expected. This step ensures that all components are properly provisioned and that the necessary connections between them are established. You can verify the infrastructure resources using the `/infrastructure` API:

- Method: GET
- Endpoints:
  - `/infrastructure`
  - `/infrastructure/{infra_id}`
  - `/infrastructure/{infra_id}/logs/{infra_stage_name}`
- Description: Verifies that the infrastructure resources have been properly provisioned and are running without any issues. It returns infrastructure details, resource status, and stage status. The stages include:
  - `setup_validation_helmchart`
  - `check_job_status_helmchart`
  - `collect_infra_artifacts`
- Sample API call request:

```
GET https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/infrastructure/<infra_id> HTTP/1.1
Authorization: Basic Auth
```

- Sample API call response:


```
{
   "data": {
      "infra_details": {
            "components_info": {
               "git": {
                  "byo_git": {
                     "git_accesstoken": "********",
                     "git_helm_repo_branch": "DEMO_HELM",
                     "git_helm_repo_url": "https://devops.scmservice.xxxxx.oci.oraclecloud.com/
namespaces/siebeldev/projects/DEMO/repositories/DEMO_HELM",
                     "git_protocol_type": "http",
                     "git_scm_flux_folder": "flux1",
                     "git_scm_repo_branch": "DEMO_SCM",
                     "git_scm_repo_url": "https://devops.scmservice.xxxxx.oci.oraclecloud.com/
namespaces/siebeldev/projects/DEMO/repositories/DEMO_SCM",
                     "git_user": "**********",
                     "skip_flux_bootstrap": false
                  },
                  "git_type": "byo_git"
               },
               "kubernetes": {
                  "byo_oke": {
                     "oke_cluster_id": "ocid1.cluster.oc1.xxxxx......",
                     "oke_endpoint": "PRIVATE"
                  },
                  "kubernetes_type": "BYO_OKE"
               },
               "registry": {
               "registry_password": "********",
```

```
                    "registry_url": "iad.ocir.io",
                    "registry_user": "******"
                }
            },
            "infra_id": "7OZMKQ",
            "infra_status": "creation-in-progress",
            "resource_status": {
                "git": "not-validated",
                "kubernetes": "not-validated",
                "registry": "not-validated"
            },
            "stages_info": [
                {
                    "end_time": "Thu, 22 May 2025 06:54:38 GMT",
                    "log_api_link": "",
                    "log_location": "/home/opc/siebel/infrastructures/7OZMKQ/helm_setup.log",
                    "name": "Setup Helm Chart for validation",
                    "previous_status": "",
                    "stage_name": "setup_validation_helmchart",
                    "start_time": "Thu, 22 May 2025 06:54:27 GMT",
                    "status": "passed"
                },
                {
                    "end_time": "Thu, 22 May 2025 06:54:38 GMT",
                    "log_api_link": "",
                    "log_location": "/home/opc/siebel/infrastructures/7OZMKQ/helm_setup.log",
                    "name": "Check Job status",
                    "previous_status": "",
                    "stage_name": "check_job_status_helmchart",
                    "start_time": "Thu, 22 May 2025 06:54:27 GMT",
                    "status": "in-progress"
                },
                {
                    "end_time": "Thu, 22 May 2025 06:54:38 GMT",
                    "log_api_link": "",
                    "log_location": "",
                    "name": "Collect Infra artifacts",
                    "previous_status": "",
                    "stage_name": "collect_infra_artifacts",
                    "start_time": "Thu, 22 May 2025 06:54:27 GMT",
                    "status": ""
                }
            ],
            "updated_values": {}
        }
        },
    "message": "Infrastructure details retrieved successfully.",
    "status": "success"
}
```


3. Registering a Customer Git Repository

You must now register the customer Git repository that contains the Open Integration runtime files. SCM stores reusable Git connection metadata and returns a generated `git_id` that you can use in subsequent Open Integration deployment requests. You can register a customer Git repository using the `/git_repositories`API:

- Method:POST
- Endpoint:`/git_repositories`

- Description: Validates access to the customer Git repository by using `git ls-remote` and stores the repository metadata under `/home/opc/siebel/git/repositories`.
- Supported options:

Option	Description	Required Parameters
HTTPS Git	Use a repository that authenticates with a user name and access token.	`git_protocol_type`, `git_url`, `git_user`, `git_accesstoken`
HTTPS Git with self-signed CA	Use a repository that is secured by a private or self-signed certificate authority (CA).	HTTPS Git parameters plus `git_selfsigned_cacert`
SSH Git	Use a repository that authenticates with an SSH private key.	`git_protocol_type`, `git_url`, `git_ssh_private_key`

- Sample API call request for HTTPS:

```
{
    "git": {
        "git_protocol_type": "https",
        "git_url": "https://<git-host>/<group>/<repo>.git",
        "git_user": "<git-user>",
        "git_accesstoken": "<git-token>",
        "git_selfsigned_cacert": ""
    }
}
```

- Sample API call request for SSH:

```
{
    "git": {
        "git_protocol_type": "ssh",
        "git_ssh_host_key_sha256": {
                "devops.scmservice.xxxxxx.oci.oraclecloud.com": "SHA256:m2uQAR8WI+o+EffY/1GR6XXXXXXXX"
        },
        "git_url": "git@<git-host>:<group>/<repo>.git",
        "git_ssh_private_key": "/home/opc/.ssh/openint_git_key"
    }
}
```

- Sample response:

```
{
    "data": {
        "git_id": "<git_id>",
        "git_url": "https://<git-host>/<group>/<repo>.git",
        "git_protocol_type": "https"
    },
    "message": "Git repository registered successfully.",
    "status": "success"
}
```

4. Uploading Starter Templates

You can upload the Open Integration starter templates from SCM to a registered customer Git repository. This step creates the initial template files, including `config_template.json`, `profile_template.json`, and other starter artifacts in

the customer Git repository. You can upload the started templates to the Git repository using the `/openintegration/git_template_upload` API:

- Method: POST
- Endpoint: `/openintegration/git_template_upload`
- Description: Pushes the starter templates from the SCM `static_templates` directory to the specified customer Git branch and directory. SCM creates a commit only when one or more files are added or changed.
- Sample API call request for default upload:

```
{
    "git": {
        "git_id": "<git_id>"
    }
}
```

  By default, SCM uploads the templates to the `main` branch and the `openint-templates` directory.

- Sample API call request for a custom branch or directory upload:

```
{
    "git": {
        "git_id": "<git_id>",
        "git_template_branch": "<git_template_branch>",
        "git_template_directory": "< git_template_directory >"
    }
}
```

In the above sample payload:

- `<git_id>` is the identifier of the customer Git repository previously registered with SCM.
- `<git_template_branch>` is the Git branch to which SCM uploads the starter templates. If omitted, SCM uses the `main` branch. It is an optional parameter.
- `<git_template_directory>` is the target directory in the customer Git repository. If omitted, SCM uses the `openint-templates` directory.

5. Uploading Sync Utility Files

You can upload local runtime files that SCM must read during deployment, such as Siebel keystore and truststore files to the SCM-managed storage. The deployment payload must reference file paths that exist on the SCM host. You can upload these files to the SCM container by using the `/syncutilities/upload` API:

- Method: POST
- Endpoint: `/syncutilities/upload`
- Description: Uploads local runtime configuration files to SCM-managed storage and returns the absolute file paths.
- Sample API call request:

```
POST /syncutilities/upload HTTP/1.1
Host: https://<scm-host>/scm/api/v1.0
Authorization: Bearer <token>
Content-Type: multipart/form-data
Form field: file[] = /path/to/siebelserverkeystore.jks
Form field: file[] = /path/to/siebelclientkeystore.jks
Form field: file[] = /path/to/siebeltruststore.jks
Form field: file[] = /path/to/aiegressserverkeystore.jks
```

```
Form field: file[] = /path/to/kafkakeystore.jks
Form field: file[] = /path/to/kafkatruststore.jks
Form field: file[] = /path/to/kafkaoauthtruststore.jks
```

- Sample API call response:

```
{
   "data": {
      "rejected_files": [],
      "sync_id": "SCM_FileSync_<date>_<id>",
      "synced_files": [
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/siebelserverkeystore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/siebelclientkeystore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/siebeltruststore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/aiegressserverkeystore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/kafkakeystore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/kafkatruststore.jks",
            "/home/opc/syncUtility/SCM_FileSync_<date>_<id>/kafkaoauthtruststore.jks"
      ]
   },
   "message": "All files successfully synced.",
   "status": "success"
}
```

Note:

- The upload request uses the multipart form-data field `file[]`.
- SCM stores uploaded files in the `/home/opc/syncUtility/{sync_id}` directory.
- Supported file extensions include `.jks`, `.crt`, `.txt`, `.ini`, `.key`, `.sso`, `.p12`, `.properties`, `.pem`, `.ora`, and `.json`
- SCM rejects empty files and files larger than 1 MB.

6. Preparing Open Integration Files in Git

You must now create deployment-ready Open Integration files from the starter templates stored in the customer Git repository. You must complete the following procedure before you call the `POST /openintegration` API:

1. Create a new deployment branch from the branch that contains the starter templates. You must avoid modifying the starter template branch directly unless your project process requires it.
2. Create `config.json` by using `config_template.json` as the starting point.
3. Create `profile.json` by using `profile_template.json` as the starting point.
4. Update `config.json` with the configuration required for the target Open Integration runtime.
5. Update `profile.json` with the configuration required for the target Siebel and Open Integration environment.
6. Add or update any required integration configurations, scripts, mappings, domain configurations, and related starter kit artifacts.
7. Commit the changes to the deployment branch.
8. Push the deployment branch to the customer Git repository.

Note:

- The deployment directory must contain committed Open Integration files, including `config.json` and `profile.json`.
- Retain the starter template files for reference unless your project process requires their removal after creating the deployment files.
- Do not store Git access tokens, SSH private keys, keystore passwords, registry passwords, or other customer secrets in the Git repository.
- Store keystore and truststore files in SCM-accessible runtime storage locations, such as Sync Utility output directories, rather than in the customer Git repository.

7. Deploying Open Integration using SCM

You can now deploy Open Integration using the infrastructure resources that you created using the configuration files stored in customer Git. This step initiates the deployment of the Open Integration services.

Optionally, you can set up the Coherence server for Open Integration by configuring the `coherence` section in the payload. You can deploy a new Coherence server or use your existing Coherence server during the Open Integration deployment.

In the deployment request, reference the registered `git_id`, the target Git branch, the target directory, the uploaded keystore and truststore file paths, and the required Coherence settings.

You can deploy Open Integration using the `/openintegration` API endpoint:

- Method: POST
- Endpoint: `/openintegration`
- Description: Initiates the deployment of Open Integration using the infrastructure resources and configuration files created in the earlier steps. It creates the Open Integration deployment, generates GitOps resources, creates runtime secrets, runs the image-builder job, and deploys the runtime.
- Sample payload for:
  - Deploying Open Integration:

```
{
   "name": "openint1",
   "infra_id": "7OZMKQ",
   "openintegration": {
      "git": {
         "git_id": "<git_id>",
         "git_openint_branch": "<customer-branch>",
         "git_openint_directory": "<customer-openint-directory>"
      },
      "siebel_server_keystore_file_path": "/home/opc/siebel/siebelkeystore.jks",
      "siebel_client_keystore_file_path": "/home/opc/siebel/siebelkeystore_client.jks",
      "siebel_truststore_file_path": "/home/opc/siebel/siebeltruststore.jks",
      "siebel_server_keystore_password": "server",
      "siebel_truststore_password": "siebel",
```

```
        "siebel_client_keystore_password": "client"
    }
}
```

- Deploying Open Integration with a new Coherence server:

```
{
    "name": "openint1",
    "infra_id": "<infra_id>",
    "openintegration": {
    "base_url": "<>",
    "git": {
        "git_id": "<git_id>",
        "git_openint_branch": "<customer-branch>",
        "git_openint_directory": "<customer-openint-directory>"
    },
    "siebel_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelserverkeystore.jks",
    "siebel_client_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelclientkeystore.jks",
    "siebel_truststore_file_path": "/home/opc/syncUtility/<sync_id>/siebeltruststore.jks",
    "siebel_server_keystore_password": "<server-keystore-password>",
    "siebel_client_keystore_password": "<client-keystore-password>",
    "siebel_truststore_password": "<truststore-password>"
    },
    "coherence": {
        "coherence_cluster_name": "<coherence-cluster-name>",
        "use_existing": "false",
        "enable_tls": "true"
    }
}
```

- Deploying Open Integration using an existing Coherence server:

```
{
    "name": "openint1",
    "infra_id": "<infra_id>",
    "openintegration": {
    "git": {
        "git_id": "<git_id>",
        "git_openint_branch": "<customer-branch>",
        "git_openint_directory": "<customer-openint-directory>"
    },
    "siebel_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelserverkeystore.jks",
    "siebel_client_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelclientkeystore.jks",
    "siebel_truststore_file_path": "/home/opc/syncUtility/<sync_id>/siebeltruststore.jks",
    "siebel_server_keystore_password": "<server-keystore-password>",
    "siebel_client_keystore_password": "<client-keystore-password>",
    "siebel_truststore_password": "<truststore-password>"
    },
    "coherence": {
        "coherence_cluster_name": "<existing-coherence-cluster-name>",
        "namespace": "<existing-coherence-namespace>",
        "use_existing": "true",
        "wka_endpoint": "<existing-coherence-wka-endpoint>"
    }
}
```

- DeployingOpen Integration using Coherence server with optional EPS keystore certificates:

```
{
    "name": "openint1",
    "infra_id": "<infra_id>",
    "openintegration": {
          "git": {
          "git_id": "<git_id>",
          "git_openint_branch": "<customer-branch>",
          "git_openint_directory": "<customer-openint-directory>"
       },
```

```
            "siebel_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/
siebelserverkeystore.jks",
            "siebel_client_keystore_file_path": "/home/opc/syncUtility/<sync_id>/
siebelclientkeystore.jks",
            "siebel_truststore_file_path": "/home/opc/syncUtility/<sync_id>/siebeltruststore.jks",
            "siebel_server_keystore_password": "<server-keystore-password>",
            "siebel_client_keystore_password": "<client-keystore-password>",
            "siebel_truststore_password": "<truststore-password>",
            "eps": {
               "ai_egress_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/
aiegressserverkeystore.jks",
               "ai_egress_server_keystore_password": "<ai-egress-password>",
               "kafka_keystore_file_path": "/home/opc/syncUtility/<sync_id>/kafkakeystore.jks",
               "kafka_keystore_password": "<kafka-keystore-password>",
               "kafka_truststore_file_path": "/home/opc/syncUtility/<sync_id>/kafkatruststore.jks",
               "kafka_truststore_password": "<kafka-truststore-password>",
               "kafka_oauth_truststore_file_path": "/home/opc/syncUtility/<sync_id>/
kafkaoauthtruststore.jks",
               "kafka_oauth_truststore_password": "<kafka-oauth-truststore-password>"
            }
         },
         "coherence": {
            "coherence_cluster_name": "<existing-coherence-cluster-name>",
            "namespace": "<existing-coherence-namespace>",
            "use_existing": "true",
            "wka_endpoint": "<existing-coherence-wka-endpoint>"
         }
      }
```

Optional Security Context

You can optionally specify the `security_context` section at the top level of the infrastructure provisioning payload when infrastructure resources and operators must run with custom non-root user ID and group ID. For example:

```
{
   "name": "<openint_name>",
   "infra_id": "<infra_id>",
   "security_context": {
      "run_as_user": <uid>,
      "run_as_group": <gid>,
      "fs_group": <fs_group_id>
   }
}
```

> **Note:** This is a separate Open Integration configuration. Include it explicitly even if the infrastructure provisioning payload also contains a `security_context` section.

The specified values are applied to the following Open Integration resources:

- Open Integration image builder
- Open Integration application workloads
- Coherence chart resources

If you specify `security_context`, you must provide all three parameters:

- `run_as_user`
- `run_as_group`
- `fs_group`

On OpenShift, `run_as_user` must be within the range specified by the `openshift.io/sa.scc.uid-range` namespace annotation. The `fs_group` value must be the first ID in the range specified by the `openshift.io/sa.scc.supplemental-groups` namespace annotation.

To determine valid values, inspect the namespace annotations and select a user ID from the `uid-range` annotation and the first group ID from the `supplemental-groups` annotation.

- Sample API call request:

```
POST https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration
 Authorization: Basic Auth
Content-Type: application/json
```

- Sample response:

```
{
   "data": {
      "deploy_id": "<deploy_id>",
      "deploy_status": "creation-in-progress",
      "infra_id": "<infra_id>",
      "name": "openint1",
      "namespace": "openint1",
      "openintegration": {
         "git": {
            "git_id": "<git_id>",
            "git_openint_branch": "<customer-branch>",
            "git_openint_directory": "<customer-openint-directory>"
      },
      "siebel_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelserverkeystore.jks",
      "siebel_client_keystore_file_path": "/home/opc/syncUtility/<sync_id>/siebelclientkeystore.jks",
      "siebel_truststore_file_path": "/home/opc/syncUtility/<sync_id>/siebeltruststore.jks",
      "siebel_server_keystore_password": "<server-keystore-password>",
      "siebel_client_keystore_password": "<client-keystore-password>",
      "siebel_truststore_password": "<truststore-password>",
      "eps": {
         "ai_egress_server_keystore_file_path": "/home/opc/syncUtility/<sync_id>/
aiegressserverkeystore.jks",
         "ai_egress_server_keystore_password": "<ai-egress-password>",
         "kafka_keystore_file_path": "/home/opc/syncUtility/<sync_id>/kafkakeystore.jks",
         "kafka_keystore_password": "<kafka-keystore-password>",
         "kafka_truststore_file_path": "/home/opc/syncUtility/<sync_id>/kafkatruststore.jks",
         "kafka_truststore_password": "<kafka-truststore-password>",
         "kafka_oauth_truststore_file_path": "/home/opc/syncUtility/<sync_id>/kafkaoauthtruststore.jks",
         "kafka_oauth_truststore_password": "<kafka-oauth-truststore-password>"
      }
   },
   "stages": [
      {
         "name": "Pre Deploy",
         "stage_name": "pre_deploy",
         "status": ""
      },
      {
         "name": "Prepare GitOps",
         "stage_name": "prepare_gitops",
         "status": ""
      },
      {
         "name": "Flux Setup",
         "stage_name": "flux_setup",
         "status": ""
      }
   ]
   },
   "message": "success",
```

```
        "status": "success"
    }
```

Optional Open Integration Base URL

You can use `openintegration.base_url` to specify the URL that Open Integration exposes to its clients. For a NodePort deployment, use an accessible Kubernetes node host name or IP address and the Open Integration NodePort. For example:

```
{
   "openintegration": {
      "base_url": "https://100.95.243.197:32082/openintegration/v1.0"
   }
}
```

If you omit this field or leave it blank, SCM uses `https://openint-service.<deployment-namespace>.svc.cluster.local:8433`.

SCM resolves the value before deployment and writes it to the Open Integration Helm Release as the BASE_URL environment variable.

Note:

- EPS JKS file-password pairs are independently optional. When you provide a file path, provide its matching password.
- For an existing Coherence cluster, provide `namespace` and `wka_endpoint`.

Certificate SAN Requirement

You must ensure that the server certificate includes the Open Integration endpoint used by the integration in its Subject Alternative Name (SAN):

- Service URL: `DNS:openint-service.<namespace>.svc.cluster.local`
- Node host name URL: `DNS:<node-dns-name>`
- Node IP URL: `IP:<node-ip-address>`

Validation Rules

The following validation rules apply when you deploy Open Integration using SCM:

- The `name` parameter is required in the payload. The value must contain 3 to 12 alphanumeric characters and start with a letter.
- The value of the `infra_id` parameter must refer to a valid SCM infrastructure record.
- The value of `openintegration.git.git_id` parameter must be a registered 6-character uppercase alphanumeric Git ID.
- The value of `git_openint_branch` parameter must be a valid Git branch name.
- The value of `git_openint_directory` parameter must be a relative directory inside the customer Git repository.
- The `siebel_server_keystore_file_path`, `siebel_client_keystore_file_path`, and `siebel_truststore_file_path` parameters are required in the payload.
- The three JKS password parameters are required in the payload.

- If you specify the `security_context` parameter, the `run_as_user`, `run_as_group`, and `fs_group` values are required and must be integers.
- The `openintegration.base_url` parameter is optional. If you omit the value or leave it blank, SCM uses the internal Open Integration Service URL.
- If you specify the `openintegration.base_url` parameter, the value must be an absolute HTTP or HTTPS URL that contains a host name or IP address. The value must not contain whitespace or embedded credentials.
- If the `openintegration.base_url` value includes a port, the port must be valid and non-zero.
- The `config_json_file_path` and `profile_json_file_path` parameters are not supported in V2.

8. Verifying the Open Integration Deployment Status

You can now verify the status of the Open Integration application. This allows you to monitor the progress of the deployment, verify that it is fully deployed, and diagnose any issues that may occur during the deployment process. You can query the deployment status using the `/openintegration` API with the deployment ID as the parameter:

- Method: GET
- Endpoint: `/openintegration/<deploy_id>`
- Description: Helps you to retrieve the status of the Open Integration deployment.
- Sample API call request:

```
GET https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/<deploy_id>
Authorization: Basic Auth
```

  In the example above, `<deploy_id>` is the unique ID that was generated when the Open Integration deployment was initiated.

- Sample API call response:

```
{
   "data": {
      "ansible_chart_name": "ansible-test6-ntntd1",
      "app_dir": "/home/opc/siebel/microservices/siebel_openint/NTNTD1",
      "coherence": {
         "coherence_cluster_name": "siebelcache",
         "namespace": "openint2",
         "use_existing": false
      },
      "deploy_id": "NTNTD1",
      "deploy_status": "completed",
      "infra_id": "GXWODN",
      "name": "test6",
      "namespace": "openint2",
      "openintegration": {
         "git": {
            "git_id": "FSDCDF",
            "git_openint_branch": "main",
            "git_openint_directory": "siebel_openint"
         },
         "siebel_server_keystore_file_path": "/home/opc/siebel/siebelkeystore.jks",
         "siebel_client_keystore_file_path": "/home/opc/siebel/siebelkeystore_client.jks",
         "siebel_truststore_file_path": "/home/opc/siebel/siebeltruststore.jks",
         "siebel_server_keystore_password": "server",
         "siebel_truststore_password": "siebel",
         "siebel_client_keystore_password": "client"
      },
      "product_name": "siebel_openint",
      "self_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1",
```

```
        "stages": [
        {
           "end_time": "Wed, 21 May 2025 12:17:39 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
prepare_gitops",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/prepare_gitops.log",
           "name": "Prepare GitOps",
           "previous_status": "",
           "stage_description": "Generate Flux definitions, Prepare helmchart to git and Prepare GitOps",
           "stage_name": "prepare_gitops",
           "start_time": "Wed, 21 May 2025 12:17:14 +0000",
           "status": "passed"
        },
        {
           "end_time": "Wed, 21 May 2025 12:18:47 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
pre_deploy",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/pre_deploy.log",
           "name": "Pre Deploy",
           "previous_status": "",
           "stage_description": "Create pre requisites and other preparations",
           "stage_name": "pre_deploy",
           "start_time": "Wed, 21 May 2025 12:17:40 +0000",
           "status": "passed"
        },
        {
           "end_time": "Wed, 21 May 2025 12:19:02 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
flux_setup",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/flux_setup.log",
           "name": "Flux Setup",
           "previous_status": "",
           "stage_description": "Do flux Setup and Perform flux bootstrap",
           "stage_name": "flux_setup",
           "start_time": "Wed, 21 May 2025 12:18:48 +0000",
           "status": "passed"
        },
        {
           "end_time": "",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
verify_deployment",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/
verify_deployment.log",
           "name": "Verify Deployment",
           "previous_status": "",
           "stage_description": "Verify HR Deployment",
           "stage_name": "verify_deployment",
           "start_time": "Wed, 21 May 2025 12:22:03 +0000",
           "status": "passed"
        }
     ]
     },
     "message": "success",
     "status": "success"
}
```


Runtime Configuration Checks

Use the following checks to verify the Open Integration runtime configuration:

- The application sets `OI_LOG_LEVEL` and `DLQ_LOG_LEVEL` to `ERROR` by default.
- The runtime `log` directory, `/openintv2Installer/openint-deploy/target/logs`, is writable and ephemeral. Log4j can create `siebel-openint.log` in this directory. The log contents are removed when the pod is recreated.

- SCM writes `Openint.cache.enabled` and `Openint.cache.tlsEnabled` in the customer `profile.json` file as JSON boolean values.
- The Coherence configuration includes `openIntRest_*` and `oi-distributed-rest-sessions` for REST-session caching. Before you use this feature, make sure the deployed Coherence runtime includes the required Open Integration cache classes.

Rerunning the Open Integration Deployment

You can rerun an existing Open Integration deployment when you want to implement configuration changes, update the application, or recover from failure. You can rerun the deployment using the `/openintegration` API with the deployment ID as the parameter:

- Method: PUT
- Endpoint: `/openintegration/<deploy_id>`
- Description: Helps you to rerun an existing Open Integration deployment when the customer Git repository content has changed or when deployment recovery is required. When you rerun the deployment, you cannot update the parameters in the payload.
- Sample API call request:

```
PUT https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/<deploy_id> HTTP/1.1
Authorization: Basic Auth
Content-Type: application/json
```

  In the example above, `<deploy_id>` is the ID of the deployment that you want to rerun.

- Sample API call response:

```
{
   "data": {
      "ansible_chart_name": "ansible-test6-ntntd1",
      "app_dir": "/home/opc/siebel/microservices/siebel_openint/NTNTD1",
      "coherence": {
         "coherence_cluster_name": "siebelcache",
         "namespace": "openint2",
         "use_existing": false
      },
      "deploy_id": "NTNTD1",
      "deploy_status": "Rerun-In-Progress",
      "infra_id": "GXWODN",
      "name": "test6",
      "namespace": "openint2",
      "openintegration": {
         "git": {
            "git_id": "FSDCDF",
            "git_openint_branch": "main",
            "git_openint_directory": "siebel_openint"
         },
         "siebel_server_keystore_file_path": "/home/opc/siebel/siebelkeystore.jks",
         "siebel_client_keystore_file_path": "/home/opc/siebel/siebelkeystore_client.jks",
         "siebel_truststore_file_path": "/home/opc/siebel/siebeltruststore.jks",
         "siebel_server_keystore_password": "server",
         "siebel_truststore_password": "siebel",
         "siebel_client_keystore_password": "client"
      },
      "product_name": "siebel_openint",
      "self_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1",
      "stages": [
```

```
        {
           "end_time": "Wed, 21 May 2025 12:17:39 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
prepare_gitops",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/prepare_gitops.log",
           "name": "Prepare GitOps",
           "previous_status": "",
           "stage_description": "Generate Flux definitions, Prepare helmchart to git and Prepare GitOps",
           "stage_name": "prepare_gitops",
           "start_time": "Wed, 21 May 2025 12:17:14 +0000",
           "status": "passed"
        },
        {
           "end_time": "Wed, 21 May 2025 12:18:47 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
pre_deploy",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/pre_deploy.log",
           "name": "Pre Deploy",
           "previous_status": "",
           "stage_description": "Create pre requisites and other preparations",
           "stage_name": "pre_deploy",
           "start_time": "Wed, 21 May 2025 12:17:40 +0000",
           "status": "passed"
        },
        {
           "end_time": "Wed, 21 May 2025 12:19:02 +0000",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
flux_setup",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/flux_setup.log",
           "name": "Flux Setup",
           "previous_status": "",
           "stage_description": "Do flux Setup and Perform flux bootstrap",
           "stage_name": "flux_setup",
           "start_time": "Wed, 21 May 2025 12:18:48 +0000",
           "status": "passed"
        },
        {
           "end_time": "",
           "log_api_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/NTNTD1/logs/
verify_deployment",
           "log_location": "/home/opc/siebel/microservices/siebel_openint/NTNTD1/logs/
verify_deployment.log",
           "name": "Verify Deployment",
           "previous_status": "",
           "stage_description": "Verify HR Deployment",
           "stage_name": "verify_deployment",
           "start_time": "Wed, 21 May 2025 12:22:03 +0000",
           "status": "in-progress"
        }
     ]
     },
     "message": "success",
     "status": "success"
}
```


Deleting an Open Integration Deployment

You can delete an existing Open Integration deployment using the `/openintegration` API:

- Method: DELETE

- Endpoint: `/openintegration/<deploy_id>`
- Description: Helps you to delete an Open Integration deployment.
- Sample API call request:

```
DELETE https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/<deploy_id>
Authorization: Basic Auth
```

  In the example above, `<deploy_id>` is the ID of the deployment that you want to delete.

- Sample API call response:

```
{
   "data": {
      "archive_dir": "/home/opc/siebel/microservices/siebel_openint/archive/H417J9",
      "deleted": true,
      "deploy_id": "H417J9"
   },
   "message": "Deletion of deploy_id H417J9 of siebel_openint successfull",
   "status": "success"
}
```

Viewing all Open Integration Deployments

You can view all the Open Integration instances deployed in the system using the `/openintegration` API:

- Method: GET
- Endpoint: `/openintegration`
- Description: Helps you to view all the Open Integration instances deployed in the system.
- Sample API call request:

```
GET https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration
Authorization: Basic Auth
```

- Sample API call response:

```
{
   "data": [
      {
      "deploy_id": "3TZE18",
      "deploy_status": "Rerun-In-Progress",
      "infra_id": "GXWODN",
      "name": "test7",
      "path": "/home/opc/siebel/microservices/siebel_openint/3TZE18",
      "self_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/3TZE18"
   },
   {
      "deploy_id": "JBE2XZ",
      "deploy_status": "Rerun-In-Progress",
      "infra_id": "GXWODN",
      "name": "test7",
      "path": "/home/opc/siebel/microservices/siebel_openint/JBE2XZ",
      "self_link": "https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/openintegration/JBE2XZ"
   },
   ],
   "message": "success",
   "status": "success"
}
```

Performing Open Integration Incremental Updates

You can perform an incremental Open Integration update after the customer Git changes are complete and pushed. This updates the infrastructure Helm chart so Flux reconciliation restarts the image builder job, builds a new application image, and starts the updated application.

To perform an incremental Open Integration update:

1. Complete all required Open Integration changes in the customer Git repository.
2. Commit and push the changes to the branch used by the deployment.
3. Go to the infrastructure Helm chart repository. You can do this from the customer Helm chart repository checkout or from the SCM pod path:
   ```
   cd /home/opc/siebel/infrastructures/<infra_id>/<helmchart_repo>/
   git pull
   ```
4. Change to the image builder chart directory for the deployment:
   ```
   cd <deploy_name>-<deploy_id>/siebel-openint-image-builder
   ```
5. Edit the `Chart.yaml` file:
   ```
   vi Chart.yaml
   ```
6. Increment the chart version value to `version: 0.1.2`.
7. Save the file, then commit and push the Helm chart change:
   ```
   git add .
   git commit -m "<message>"
   git push
   ```

An incremental update performs the following:

- The Helm chart version change stages the incremental update for reconciliation.
- Flux reconciliation detects the updated chart version.
- The `siebel-openint-image-builder` job restarts and builds a new Open Integration application image from the latest committed customer Git content.
- The updated Open Integration application starts by using the new image.

Note:

- Ensure that all customer Git changes are committed and pushed before you increment the chart version.
- Increment only the chart version that is required to trigger the image builder update.
- Do not commit credentials, tokens, keystore passwords, registry passwords, or other secrets when updating customer Git or infrastructure Helm charts.

Updating Only Git Details in an Existing Deployment

You can use this procedure to update only the customer Git details in an existing Open Integration deployment.

To update only Git details:

1. Go to the infrastructure SCM chart repository, pull the latest changes, and open the image-builder Helm release:
```
cd /home/opc/siebel/infrastructures/<infra_id>/<scm_repo>/
git pull
cd <deploy_name>-<deploy_id>/apps/base/siebel-<deploy_name>/
vi siebel-openint-image-builder-helmrelease.yaml
```
2. Update the customer Git values under `spec.values`.
3. Commit and push the Helm repository change:
```
git add .
git commit -m "<message>"
git push
```

Flux reconciles the Helm release, restarts the image-builder job, and builds the application from the updated customer Git source.

Updating OI_LOG_LEVEL and DLQ_LOG_LEVEL

The runtime sets OI_LOG_LEVEL and DLQ_LOG_LEVEL to ERROR by default. You can change either value for an existing deployment by editing the Open Integration application Helm release values.

To change the values of OI_LOG_LEVEL and DLQ_LOG_LEVEL:

1. Go to the infrastructure SCM chart repository, pull the latest changes, and open the application HelmRelease:
```
cd /home/opc/siebel/infrastructures/<infra_id>/<scm_repo>/
git pull
cd <deploy_name>-<deploy_id>/apps/base/siebel-<deploy_name>/
vi siebel-openint-app-helmrelease.yaml
```
2. Under `spec.values.env`, set the required values:
```
env:
   - name: OI_LOG_LEVEL
     value: "INFO"
   - name: DLQ_LOG_LEVEL
     value: "DEBUG"
```
3. Commit and push the Helm repository change:
```
git add .
git commit -m "<message>"
git push
```

Flux reconciles the application HelmRelease. The Open Integration application pod restarts with the new log levels.

Updating BASE_URL After Deployment

You can update the Open Integration runtime URL after deployment by updating the Open Integration application Helm release in the infrastructure SCM Git repository.

To update the Open Integration runtime URL after deployment:

1. Go to the infrastructure SCM chart repository, pull the latest changes, and open the application HelmRelease:
```
cd /home/opc/siebel/infrastructures/<infra_id>/<scm_repo>/
git pull
cd <deploy_name>-<deploy_id>/apps/base/siebel-<deploy_name>/
```

```
vi siebel-openint-app-helmrelease.yaml
```

2. Under `spec.values`, update `baseUrl` and the `BASE_URL` entry under `env` to the same value:

```
spec:
   values:
      baseUrl: "https://100.95.243.197:32082/openintegration/v1.0"
      env:
         - name: OI_LOG_LEVEL
           value: "ERROR"
         - name: DLQ_LOG_LEVEL
           value: "ERROR"
         - name: BASE_URL
           value: "https://100.95.243.197:32082/openintegration/v1.0"
```

3. Commit and push the HelmRelease change:

```
git add siebel-openint-app-helmrelease.yaml
git commit -m "Update OpenIntegration BASE_URL"
git push
```

4. Wait for Flux reconciliation and verify the value in the running container.

```
kubectl rollout status deployment/siebel-open-integration -n <openint_namespace>
kubectl exec -n <openint_namespace> deploy/siebel-open-integration -- printenv BASE_URL
```

   Flux reconciles the Open Integration application Helm release. The application pod restarts with the updated `BASE_URL`, and the `printenv BASE_URL` command returns the new URL.

Note: You must:

- Provide an absolute HTTP or HTTPS URL without whitespace or embedded credentials. If you specify a port, use a valid, non-zero port.
- Make sure the server certificate SAN covers the host name or IP address when you use HTTPS.
- Set `openintegration.base_url` in the initial deployment payload when the value must remain stable across reruns. A later SCM rerun regenerates resources from the saved deployment payload and can restore the original `BASE_URL`.

Troubleshooting Open Integration Deployment Issues

The following table lists the common Open Integration deployment issues and possible resolutions:

Issue	Resolution
Infrastructure creation fails	Verify registry credentials, GitOps repository access, Helm repository access, Kubernetes permissions, namespace, OCI access, and operator prerequisites.
Git registration fails	Verify the URL and protocol, credentials, SSH key path, optional CA certificate path, and network access from the SCM host.
Template upload returns no new commit	The target files already match the SCM starter templates in the selected branch and directory.

Issue	Resolution
Template upload fails	Confirm the **git_id**, branch name, relative directory, repository write permission, and customer Git network access.
Sync upload rejects files	Check the file extension, file size, and whether the file is empty.
Deployment payload is rejected	Confirm **name**, **infra_id**, **git_id**, branch, relative directory, keystore and truststore paths, passwords, and Coherence fields.
Image builder cannot clone customer Git	Check **siebel-openint-git-secrets**, the **/git-secrets** mount, the generated **git.json**, and the image-builder pod logs.
Image builder cannot find Open Integration files	Confirm that **git_openint_branch** and **git_openint_directory** point to committed files in customer Git.
Rerun does not pick up changes	Confirm that the changes were committed and pushed to the saved branch and directory before you submit the bodyless **PUT** request.
Incremental update does not start a new image builder job	Confirm that the customer Git changes were pushed, **Chart.yaml** was incremented under **<deploy_name>-<deploy_id>/siebel-openint-image-builder**, and the Helm chart change was committed and pushed.

Security Considerations:

- Do not include actual Git access tokens, SSH private keys, certificate authority (CA) certificates, Java KeyStore (JKS) passwords, registry passwords, or customer secrets in documentation examples, sample payloads, or source-controlled files.
- For HTTPS Git repositories, ensure SCM mounts access tokens as secret files at runtime and does not store or embed token values in **git.json**.
- For SSH Git repositories, ensure SCM mounts the SSH private key as **/git-secrets/ssh_private_key**.
- For HTTPS repositories that use a self-signed or private CA certificate, ensure SCM mounts the CA certificate as **/git-secrets/https_trust_source**.
- Ensure that the generated **git.json** file contains runtime metadata and references to secret file locations and does not contain raw secret values.
- Keystore and truststore paths specified in the deployment payload must be accessible to SCM and should reference approved runtime storage locations, such as Sync Utility output paths.
- Store sensitive files and credentials only in approved secret-management or runtime-storage locations. Do not commit them to customer Git repositories or infrastructure Helm chart repositories.

7 Monitoring Siebel CRM Deployments

Monitoring Siebel CRM Deployments

This chapter describes how you can monitor the backend of a Siebel CRM deployment created using Siebel Cloud Manager (SCM) using the **Siebel CRM Observability - Monitoring and Log Analytics** module. This module helps you align your Siebel CRM deployment with cloud-native best practices.

The observability stack uses best of breed tools such as Prometheus, Grafana, Oracle OpenSearch, Fluentd, OCI services, and other supporting components.

The **Siebel CRM Observability - Monitoring and Log Analytics** module includes the following components:

- Siebel CRM Observability – Monitoring
- Siebel CRM Observability – Log Analytics

This feature is optional. You can enable and manage it by using SCM.

This chapter contains the following topics:

- *Metrics Information Categories*
- *Siebel CRM Monitoring Architecture*
- *Key Software Components for Monitoring*
- *Visualization Components for Monitoring*
- *Configuring the Siebel CRM Observability – Monitoring Solution*
  - *Enabling the Solution*
  - *Few Parameters for Prometheus Configuration for Siebel CRM Monitoring*
  - *Alert Notifications*
  - *Custom Siebel CRM Metrics*
  - *Additional Node Exporter Metrics*
- *Dashboards for Siebel CRM Monitoring*
- *Siebel CRM Observability - Data Visualization and Metrics Monitoring for Oracle Databases*

Benefits of the Observability Solution

Some of the benefits a modern observability solution like this can provide are:

- Reduction of administrative overhead and skills dependency for production support:
  - A near real-time, at-a-glance view of system health of your Siebel CRM backend is available.
  - Can automatically detect anomalous behavior and generate notifications.
- Helps in improving user experience:
  - Makes metrics available to configure automatic dynamic scaling of Siebel Servers.
  - Provides information to monitor system under heavy load.

- Helps track the impact of configuration changes on system behavior.
- Makes available data to predict system failures before they occur.

- Enables planning investments based on data:
  - Infrastructure investment planning is aided by having accurate information on the demand on resources.
  - Make available stored data to analyze past performance of systems.

Metrics Information Categories

This observability solution will capture end to end metrics for all the infrastructure deployed as part of a Siebel CRM deployment done by SCM.

Here are the categories of metric information that are captured and displayed on sample dashboards delivered as part of this solution:

- *Node Metrics*
- *Container Metrics*
- *Kubernetes Metrics*
- *Traefik Ingress Metrics*
- *OCI Infrastructure Service Metrics*
- *Metrics for Java-based Services in Siebel CRM*
- *Siebel Server Metrics*
- *Oracle Database Metrics*

Node Metrics

Capture and propagate hardware-level metrics from each node in the cluster. For example, CPU usage, memory consumption, disk I/O, network statistics, and so on. Node Exporters are used to collect these metrics.

- For more information on node exporters, refer *https://github.com/prometheus/node_exporter*.
- For a detailed list of all available metrics, refer to the "Prometheus Server Node Metrics" section in *Monitoring & Alerting Capabilities in an Oracle Private Cloud Appliance X9-2 (PDF)*.

Container Metrics

The Siebel Observability solution captures and propagates container-level metrics like CPU and memory usage, file system statistics, network activities, and so on. cAdvisor is used to collect metrics.

- For more information on cAdvisor, refer *https://github.com/google/cadvisor/tree/master*.
- More details about cAdvidor and all available metrics, refer *https://github.com/google/cadvisor/blob/master/docs/storage/prometheus.md*.

Kubernetes Metrics

Information about the state and health of various Kubernetes objects like pods, deployments, services, and so on are captured. For example, Kubernetes objects CPU, memory consumption, disk I/O, network statistics among others.

"kube-state-metrics", which is a service that listens to the Kubernetes API server and generates metrics about the state of the objects, are used.

- For more information on kube-state-metrics, refer *https://github.com/kubernetes/kube-state-metrics/tree/main*.
- For a detailed list of all available metrics, refer *https://github.com/kubernetes/kube-state-metrics/blob/main/docs/metrics/workload/pod-metrics.md*.

Traefik Ingress Metrics

Metrics are also collected for Traefik Ingress Controllers. These help in the monitoring and management of ingress traffic by measuring, for example, total number of client requests, sum of response duration per ingress, request processing time, upstream service latency per ingress, and so on. This observability solution uses Traefik native exporters for collecting metrics for Prometheus to scrape.

For more information on Traefik exporters, refer *https://doc.traefik.io/traefik/reference/install-configuration/observability/metrics/#prometheus* .

OCI Infrastructure Service Metrics

In this category, the observability – monitoring solution relays data about the health, capacity, and performance of cloud resources on OCI. A wide array of metrics for OCI services in use, like database, network, vault, and so on, are available. Metrics and Alarms features of OCI Monitoring Service are used to collect these.

For more information, refer *https://docs.oracle.com/en-us/iaas/Content/Monitoring/Concepts/monitoringoverview.htm*.

Metrics for Java-based Services in Siebel CRM

For metrics categories belong to Java services used in Siebel CRM, Java Management Extension (JMX) exporters are used. JMX in Gateway, Siebel Server, AI and SMC pods help in collecting necessary metrics.

For more information on JMX exporters, refer *https://github.com/prometheus/jmx_exporter*.

Siebel Server Metrics

This category of metrics help in monitoring important aspects of the Siebel application servers. Siebel Server related metrics are collected through custom monitoring agents. The following out-of-the-box metrics are collected:

Max_Tasks, Total_Tasks, Active_Tasks, Total_Siebel_Servers, Active_Siebel_Servers, Active_Processes, Max_Mts_Process, Active_Mts_Process, Active_Sessions, Siebel_Server_State, SIEBEL_COMPONENT_INFO, SIEBEL_COMPGRP_INFO, SIEBEL_TASK_INFO, SIEBEL_PROCESS_INFO, SIEBEL_SESSION_INFO, and so on.

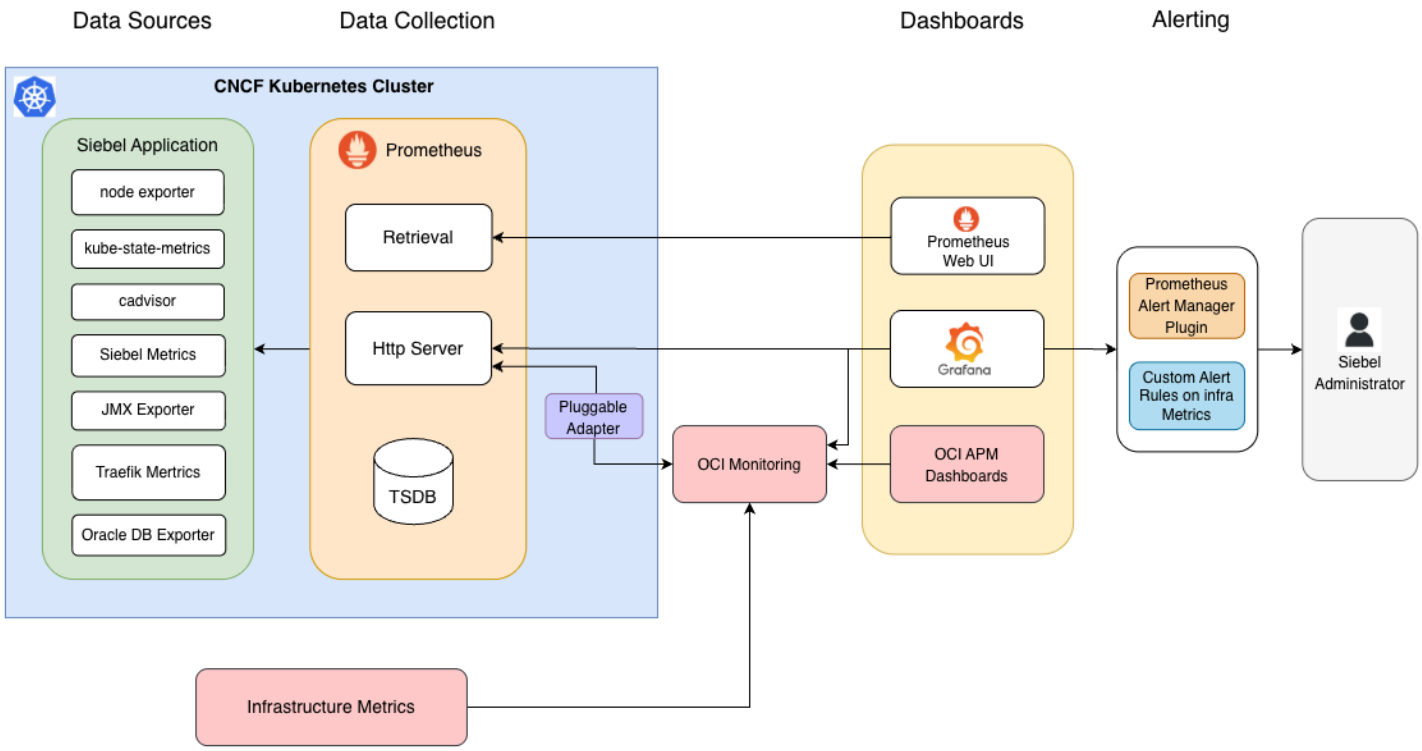
The framework allows collection and propagation of more Siebel information to be collected as metrics by processing server manager commands.

Oracle Database Metrics

You can monitor the key performance, health, and operational statistics of Oracle Databases integrated with Siebel CRM using Oracle Database metrics. Oracle Database Metrics Exporter (Oracle DB Exporter) collects metrics from database queries in real time and makes these metrics available to Prometheus for aggregation. You can visualize the metrics in Grafana or OCI dashboards, enabling proactive issue detection, performance optimization, and generation of business insights.

Siebel CRM Monitoring Architecture

The following solution architecture diagram shows all the software components that capture, transform, propagate, store and display metrics data of all necessary elements of a Siebel Server deployment on OKE by SCM.



A large array of metrics related to networks, disks, nodes, pods, containers, database, Kubernetes and Siebel deployment components are collected and transmitted for viewing and analysis. cAdvisor is the metrics exporter for running containers. Other categories of metrics exporters include node exporters, kube-state -metrics exporters, Siebel metrics exporter, JMX exporters and Traefik Ingress metrics exporter.

These metrics are scraped/collected and stored in time series database in Prometheus, which, along with OCI infrastructure service metrics, feeds into Prometheus Web UI, Grafana and OCI monitoring console for viewing the collected metric. Information from these flows are also used by Prometheus Alert Manager plugin, which, in addition to OCI alerting services, can be configured to notify users of incidents that match defined criteria.

Key Software Components for Monitoring

The following key software components are used for monitoring:

- *Prometheus*
- *OCI Monitoring*
- *Exporters*

Prometheus

Prometheus is one of the most important components of the Siebel CRM Observability - Monitoring solution. It is one of the most widely used open-source monitoring software in the world today and is built in the Go language. Prometheus joined the *Cloud Native Computing Foundation* in 2016 as the second hosted project, after *Kubernetes*.

Metrics collected from various infrastructure components like nodes, pods, containers, load balancer, Siebel application, OKE, mount targets, and so on are sent to Prometheus for storage in time series database (that is metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels). Alert rules configurations are also done in Prometheus. A Prometheus component called *Alertmanager* evaluates alert rules and sends notifications to target channels like email, mobile, and so on.

Prometheus also provides a web-based UI, metrics and query endpoints and a query language PromQL for analysis.

It offers extensive integration capabilities - due to its popularity, a plethora of exporters are available open-source to collect metrics from many different software. This can be another very important consideration for using Prometheus in any organization.

For more information, refer *https://prometheus.io/*.

OCI Monitoring

The Siebel CRM Observability - Monitoring solution also offers extensive capabilities through OCI Monitoring solution. For all the OCI and Siebel CRM services, insights are available at your fingertips. The solution implements Prometheus-OCI adapter to send data in Prometheus to OCI Monitoring for use with other various OCI monitoring and dashboarding solutions.

For more information, refer *https://docs.oracle.com/en-us/iaas/Content/Monitoring/home.htm*.

Exporters

Exporters are the components in play for collecting and exposing the metrics for ingestion into Prometheus. They work on all targets being monitored like nodes, load balancer, and so on and convert the information to suitable formats for Prometheus to ingest.

Visualization Components for Monitoring

The Siebel CRM Observability - Monitoring solution includes usage of various dashboard visualization tools.

This solution offers the following options for metrics data visualization and dashboarding:

- *Grafana*
- *Oracle APM Dashboard* (OCI Application Performance Monitoring)
- *Prometheus UI*

Grafana

Grafana is a popular Open Source visualization and dashboarding tool that connects to Prometheus to retrieve metrics data. It enables you query, visualize, alert on, and explore your metrics easily. Grafana also provides users tools to turn time-series database (TSDB) data into insightful graphs and visualizations.

Note that Grafana server/service is not distributed by Oracle. JSONs of various sample dashboards are provided by Oracle, to be imported into Grafana that is managed by the user. Features described in this document were tested with Grafana version 9.4.3.

For more information, refer *https://grafana.com/grafana/*.

Oracle APM Dashboard

OCI Application Performance Monitoring (APM) Dashboard is a web-based interface to configure and manage Oracle OCI Monitoring that helps in creating dashboards, setting up alarms, and exporting metrics data. Few sample dashboards are provided from Oracle Siebel.

For more information, refer *https://docs.oracle.com/en-us/iaas/management-dashboard/home.htm*.

Prometheus UI

Prometheus UI is an expressions browser visual interface directly on Prometheus – the most important component in the offering. It's invaluable in the technical exploration of metrics and configuration checks and hence an essential tool for troubleshooting and ad-hoc analysis. Prometheus UI enables exploring and alerting based on metrics, executing queries and gain insights into the monitored systems' performance. It supports PromQL for advanced selection and aggregation of time series data and display in real time.

For more information, refer *https://prometheus.io/docs/visualization/browser/*.

Configuring the Siebel CRM Observability – Monitoring Solution

This section contains the following topics:

- *Enabling the Solution*
- *Disabling OCI Monitoring for Siebel CRM*
- *Few Parameters for Prometheus Configuration for Siebel CRM Monitoring*
- *Alert Notifications*
- *Custom Siebel CRM Metrics*
- *Additional Node Exporter Metrics*

Enabling the Solution

The Observability Monitoring feature can be enabled to monitor Siebel CRM environments deployed by SCM on Oracle Kubernetes Engine on OCI. This feature exposes APIs for easy enabling/disabling. Various aspects of the Observability feature like Monitoring, Logging Analytics, Alerts, OCI Logging Analytics, Oracle OpenSearch, and so on can be individually turned on/off.

To enable only Monitoring during a Siebel CRM deployment, section like this to be appended in Siebel CRM deployment payload. For example:

API: POST on `/scm/api/v1.0/environment`

```
{
```

```
        <Siebel deployment payload>
        "observability": {
           "siebel_monitoring": true,
           "oci_config": {
              "oci_config_path": "/home/opc/siebel/oci-config/config1",
              "oci_private_api_key_path": "/home/opc/siebel/oci-config/mykey.pem",
              "oci_config_profile_name": "DEFAULT"
              },
        "monitoring_mt_export_path": {
           "mount_target_private_ip": "10.0.255.YY",
           "export_path": "/devXX-monitoring"
              }
        }
    }
```

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

When monitoring is enabled during Siebel deployment, the `monitoring_mt_export_path` parameter needs to be provided only when BYOR choice (Use existing resources) was selected during SCM installation.

To enable monitoring for a pre-existing Siebel CRM deployment done by SCM, the following sample can be used:

API: POST on `/scm/api/v1.0/environment/<ENV_ID>/observability`

```
    {
       "observability": {
          "siebel_monitoring": true,
          "oci_config": {
             "oci_config_path": "/home/opc/siebel/oci-config/config1",
             "oci_private_api_key_path": "/home/opc/siebel/oci-config/mykey.pem",
             "oci_config_profile_name": "DEFAULT"
          },
       "monitoring_mt_export_path": {
          "mount_target_private_ip": "10.0.255.YY",
          "export_path": "/devXX-monitoring"
        }
       }
    }
```

The `monitoring_mt_export_path` parameter is required when monitoring is enabled for a pre-existing Siebel CRM deployment done by SCM.

It returns a `RUN_ID` upon success.

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

The status of the enabled features can be checked with GET for specific `RUN_IDS`, or you can get a broader response with an upper level URI ending in the term "observability".

```
/scm/api/v1.0/environment/<ENV_ID>/observability/<RUN_ID>

/scm/api/v1.0/environment/<ENV_ID>/observability/
```

Re-runs can be done with PUT API with RUN_ID at the end. Note that reruns are idempotent.

```
/scm/api/v1.0/environment/<ENV_ID>/observability/<RUN_ID>
```

To use the Siebel CRM Observability – Monitoring solution, Siebel Cloud Manager version needs to be updated to CM_24.6.0 or later using commands like following. Refer to the appropriate section for details on update process. Though SCM needs to be updated, Siebel CRM version need not be updated always for using this feature, as limited backward compatibility for Siebel CRM versions below 24.6 is supported.

```
ssh opc@<CM_IP>
bash start_cmserver.sh CM_24.6.0
```

To run observability workloads with a custom non-root user ID and group ID, include a `security_context` section under the `observability` section in the deployment payload, in addition to the configuration required to enable monitoring.

```
{
   "observability":{
   "security_context": {
      "run_as_user": <uid>,
      "run_as_group": <gid>,
      "fs_group": <fs_group_id>
      }
   }
}
```

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

Note: On OpenShift, `run_as_user` must be within the range specified by the `openshift.io/sa.scc.uid-range` namespace annotation. The `fs_group` value must be the first ID in the range specified by the `openshift.io/sa.scc.supplemental-groups` namespace annotation. To view these namespace annotations, run:

```
oc get namespace <siebel_namespace> -o jsonpath='{"openshift.io/sa.scc.uid-range\t"}
{.metadata.annotations.openshift\.io/sa\.scc\.uid-range}{"\n"}{"openshift.io/sa.scc.supplemental-groups\t"}
{.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}{"\n"}'
```

In the above example:

- Use a value from the `uid-range` annotation for `run_as_user`.
- Use the first ID from the `supplemental-groups` annotation for `fs_group`.

Note: If you specify `observability.security_context` and use local storage for the Prometheus persistent volume, update the ownership of the local storage directory on the Kubernetes worker node before enabling observability. For more information, see *Observability: Local Storage Prerequisite for Custom Non-Root User ID Security*.

To enable Alerting along with Monitoring, deployment payload to contain section like the following in addition to the section for enabling monitoring:

```
{
   "observability": {
   ......
   "send_alerts": "true",
   "alertmanager_email_config": {
      "smtp_host": "smtp.us-ashburn-1.oraclecloud.com",
      "smtp_port": "587",
      "smtp_from_email": "no-reply@oraclesiebel.com",
      "smtp_auth_username": "ocid1.user.oc1.......",
      "smtp_auth_password": "ocid1.vaultsecret.oc1.uk-london-1.....",
      "to_email": "test1@oracle.com,test2@oracle.com "
      }
   }
}
```

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

Therefore, to enable monitoring functionality along with alerting in Siebel CRM Observability, a payload like the following can be used (for a non-BYO use case) along with Siebel CRM deployment payload:

```
{
   <other Siebel CRM deployment payload elements>
   "observability": {
      "siebel_monitoring": true,
      "oci_config": {
         "oci_config_path": "/home/opc/siebel/oci-config/config1",
         "oci_private_api_key_path": "/home/opc/siebel/oci-config/mykey.pem",
         "oci_config_profile_name": "DEFAULT"
         }
      "send_alerts": "true",
      "alertmanager_email_config": {
         "smtp_host": "smtp.us-ashburn-1.oraclecloud.com",
         "smtp_port": "587",
         "smtp_from_email": "no-reply@oraclesiebel.com",
         "smtp_auth_username": "ocid1.vaultsecret.oc1.uk-london-1.....",
         "smtp_auth_password": "ocid1.vaultsecret.oc1.uk-london-1.....",
         "to_email": "test1@oracle.com,test2@oracle.com"
         }
   }
}
```

Observability: Local Storage Prerequisite for Custom Non-Root User ID Security

If you specify `security_context` under the `observability` section and use local storage for the Prometheus persistent volume, update the ownership of the local storage directory on the Kubernetes worker node before enabling observability.

Run the following command on the worker node that hosts the local storage directory:

```
chown <run_as_user>:<fs_group> <local_storage_directory_path>
```

In this command:

- `<run_as_user>` is the value configured for `observability.security_context.run_as_user` in the deployment payload.
- `<fs_group>` is the value configured for `observability.security_context.fs_group` in the deployment payload.
- `<local_storage_directory_path>` is the path to the local storage directory used by the Prometheus persistent volume.

Disabling OCI Monitoring for Siebel CRM

You can disable OCI monitoring that was enabled earlier through the `enable_oci_monitoring` parameter. Setting the value of `enable_oci_monitoring` to `false` prevents Prometheus from sending metrics to the OCI monitoring service.

To disable OCI monitoring for Siebel, include the `enable_oci_monitoring` parameter in the Siebel CRM deployment payload as follows:

```
{
     "observability": {
          "enable_oci_monitoring": false
     }
}
```

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

Note: Disabling OCI monitoring will not delete the OCI Application Performance Management (APM) dashboard that was created when OCI monitoring was enabled.

Few Parameters for Prometheus Configuration for Siebel CRM Monitoring

Prometheus has a vast number of configuration options which can be used depending on specific business requirements. Of those, we want to highlight a few that need careful balance between processing and storage needs vs latency of how soon metrics information is available in Prometheus from the exporters:

- Scraping Interval: Defines how often Prometheus collects metrics.
- Retention Policy: Dictates how long to store metrics in the time series database of Prometheus.
- Evaluation Interval: Defines how often Prometheus evaluates rules.

These parameters are available in the SCM Git Repository in the file `/flux-crm/apps/base/siebel_observability/prometheus.yaml`.

In the sample section below, inside `prometheus.yaml` file, these define config variables global in scope that is, parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections. Individual target specific configurations are also possible to be configured. Refer to Prometheus documentation for more details.

```
values:
   server:
      global:
         scrape_interval: 1m
         scrape_timeout: 10s
         evaluation_interval: 1m
      retention: "15d"
```

A scrape_config section specifies a set of targets and parameters describing how to scrape them. In the general case, one scrape configuration specifies a single job. In advanced configurations, this may change.

Targets may be statically configured via the `static_configs` parameter. Includes parameters like target IP address, scrape interval and more. Defined in

- Git repository: `<namespce>-helmcharts`
- File: `<namespce>-helmcharts/prometheus/templates/configMap.yaml` under `prometheus.yml` section

Here is a sample section containing these parameters:

```
prometheus.yml: |-
   scrape_configs:
      - job_name: jmx-exporter
        scrape_interval: 5s
        kubernetes_sd_configs:
        - role: endpoints
          namespaces:
            names:
            - {{ .Release.Namespace }}
        relabel_configs:
        - action: keep
          source_labels:
          - __meta_kubernetes_endpoint_port_name
          regex: jmx-metrics
```

Targets can be dynamically discovered using one of the supported service-discovery mechanisms. For example, Kubernetes service discovery, DNS-based service discovery and so on.

> **Note:** Whenever you update a file in a Helm chart, you must increment the `version` field in the corresponding `Chart.yaml` file. This ensures that the deployed state is reconciled with the declared state defined in your YAML files.

For the latest Prometheus configuration options, refer *https://prometheus.io/docs/prometheus/latest/configuration/configuration/*.

Alert Notifications

The Siebel CRM Observability – Monitoring solution offers ability to generate alert notifications based on predefined conditions incorporating various metrics collected.

Alerting can be handled with Prometheus for all involved resources as it's where the collected metrics reside. It is also possible to configure alerting with OCI Notifications service. For details on using OCI Notification services, refer official documentation available at *https://docs.public.oneportal.content.oci.oraclecloud.com/en-us/iaas/Content/Notification/home.htm*.

Broadly, this is how alert notification is functionally handled with Prometheus based configurations:

- Alerting rules defined in Prometheus servers get evaluated and when necessary conditions are fulfilled, alerts get send to Alertmanager configured.
- The Alertmanager can be configured to manage alerts using, among others, actions like:
  - Grouping alerts of similar nature into a single notification.
  - Inhibition or suppressing certain alerts if certain other alerts are already firing.
  - Silencing or muting alerting for specific time periods, and so on.
- The Alertmanager can send notifications to:
  - Email systems
  - On-call notification systems
  - Chat platforms

Alert notifications using Prometheus in the Siebel CRM Observability - Monitoring solution involve:

- Creating alerting rules in Prometheus.
- Configuring Prometheus to connect to and notify the Alertmanagers.
- Setup and configuration of the Alertmanager, which ultimately sends notification to target channels.

Alerting Rules in Prometheus

Rules for alert trigger conditions are defined in the `prometheus/templates/configMap.yaml` file under `prometheus.rules` key in the Helm charts Git repository.

Specific metrics and thresholds can be configured by following the Prometheus documentation.

The rules are written in PromQL (Prometheus Query Language).

Here is a sample of the alert rule block to be used in Prometheus that will get evaluated to true when container CPU usage is above 60% - the evaluation checks the value over period blocks of 15 minutes.

```
prometheus.rules: |-
   groups:
   - name: siebel alerts
      rules:
      - alert: ContainerHighCpuUtilization
         expr: (sum(rate(container_cpu_usage_seconds_total{name!=""}[15m]))
            BY (instance, name) * 100) > 60
         for: 2m
         labels:
            severity: critical
         annotations:
            summary: |
               Container High CPU utilization (instance {{ "{{" }}
               $labels.instance }})
            description: |
               " Container CPU utilization is above 60%\n
               VALUE = {{ "{{" }} $value }}\n LABELS = {{ "{{" }} $labels }}"
```

A few of the notations used as example above are briefly explained below. For more details, refer Prometheus documentation available at *https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/*.

- `Groups`: Alerting rules exist in a rule group. Rules within a group are run sequentially at a regular interval, with the same evaluation time.
- `alert`: The name of the alert. It must be a valid label value. It's a string type.
- `expr`: It is string type PromQL expression to evaluate. In every evaluation cycle this is evaluated at the current time, and all resultant time series become pending/firing alerts.
- `for`: The optional *for* clause causes Prometheus to wait for a certain duration between first encountering a new expression output vector element and counting an alert as firing for this element. In this case, Prometheus will check that the alert continues to be active during each evaluation for 2 minutes before firing the alert.
- `labels`: The *labels* clause allows specifying a set of additional labels to be attached to the alert. Any existing conflicting labels will be overwritten. The label values can be templated.
- `annotations`: The annotations clause specifies a set of informational labels that can be used to store longer, additional information such as alert descriptions or runbook links. The annotation values can be templated.

Label and annotation values can be templated using console templates. The $labels variable holds the label key/value pairs of an alert instance. The $value variable holds the evaluated value of an alert instance.

Refer Prometheus documentation on to how to use PromQL and all other options available to set up rules in Prometheus server that meet your business requirements.

Target Alertmanager endpoints are also defined in the same file `<Helm charts repository name>/prometheus/templates/configMap.yaml` but under `prometheus.yml`. Among various options available (refer Prometheus official documentation for all options), Alertmanager's URL and any routing or grouping configurations are noteworthy.

A sample configuration is provided below.

```
prometheus.yml: |-
   global:
      {{ .Values.server.global | toYaml | trimSuffix "\n" | indent 6 }}
      {{- if .Values.alerting }}
      rule_files:
         - /etc/prometheus/prometheus.rules
      alerting:
         alertmanagers:
         - scheme: http
            static_configs:
            - targets:
               - "prometheus-alertmanager.{{ .Release.Namespace }}.svc:9093"
```

```
{{- end }}
```

Above is a configuration for Prometheus to inform Alertmanager when the rule previously defined in Prometheus (under `prometheus.rules` key) evaluates to True.

The rule files can be reloaded at runtime by sending SIGHUP to the Prometheus process. The changes are only applied if all rule files are well-formatted.

In the above sample:

- `alerting` section to define the alerting target Alertmanager.
- `Scheme` may contain values http or https. Because the alertmanager pods are within the same cluster, Siebel CRM Observability - Monitoring solution uses http which is the default scheme.
- `.Values` contains values defined in `values.yaml`.
- `.Release.Namespace` is a variable containing the namespace of the current Helm release in "Helm Templating Language".

Note: Whenever you update a file in a Helm chart, you must increment the `version` field in the corresponding `Chart.yaml` file. This ensures that the deployed state is reconciled with the declared state defined in your YAML files.

Prometheus Alertmanager Configurations

Details of Prometheus Alertmanager configurations are available at *https://prometheus.io/docs/alerting/latest/configuration/*. In this document, we will touch upon a very small set of considerations to keep in mind.

Alertmanager configurations are defined in the `prometheus-alert-manager/templates/AlertManagerConfigmap.yaml` file under `config.yml` key in the Helm charts Git repository.

The Alertmanager is configured using a YAML-based configuration file. Essential configuration components and parameters include:

- **Global Configurations**
  - `resolve_timeout`: This global setting defines the default duration after which an alert will be considered resolved if no more firing alerts are received for it.

  Example snippet:

```
global:
   resolve_timeout: 5m
   smtp_smarthost: {{ .Values.email_config.smtp_host }}:{{ .Values.email_config.smtp_port }}
   smtp_from: {{ .Values.email_config.smtp_from }}
   smtp_auth_username: {{ .Values.email_config.smtp_auth_username }}
   smtp_auth_password: {{ .Values.email_config.smtp_auth_password }}
```

- **Route Configurations**
  - `receiver`: Specifies the default receiver for alerts.
  - `group_by`: Groups alerts by specific labels. In this example, alerts are grouped by alertname and severity.
  - `group_wait`: Specifies how long to wait before grouping alerts. New alerts within this window will be grouped together.
  - `group_interval`: Defines the interval at which groups of alerts are evaluated for sending.
  - `repeat_interval`: Specifies how often to repeat notifications for the same alert group.
  - `routes`: Defines routing rules. In this example, alerts with a severity label set to "critical" are sent to the 'urgent-email' receiver, while others are sent to the 'normal-email' receiver.

  Example snippet:

```
route:
   receiver: alert-emailer
   group_by: ['alertname', 'priority']
   group_wait: 10s
   group_interval: 5m
   repeat_interval: 30m
   routes:
      - receiver: alert-emailer
      matchers:
         - severity="critical"
```

- **Receiver Configurations**
  - `receivers`: specify different receivers for alerts. Each receiver can have various configurations based on the notification channel, such as email, Slack, or other integrations.

  Example snippet:

```
receivers:
- name: alert-emailer
   email_configs:
      - to: "team@example.com"
```

Note: Whenever you update a file in a Helm chart, you must increment the `version` field in the corresponding `Chart.yaml` file. This ensures that the deployed state is reconciled with the declared state defined in your YAML files.

It is recommended to point Prometheus to a list of all Alertmanagers instead of load-balancing.

Note: To update the Alertmanager email configuration, you must update the SCM Git repository file `flux-crm/apps/base/siebel_observability/prometheus_alertmanager.yaml`. After you update the file, reconcile Flux and restart the Alertmanager deployment. This ensures that the running Alertmanager pod loads the updated ConfigMap.

Custom Siebel CRM Metrics

In addition to the metrics already being collected and streamed to dashboards, additional Siebel metrics collection options are supported that can be obtained by processing server manager commands. For example, metrics for Process, sessions, statistics and any other metric by processing server manager commands.

This topic contains the following sections:

- *Enable Metrics for Processes, Sessions, and Statistics*
- *Add Custom Siebel Metrics*

Enable Metrics for Processes, Sessions, and Statistics

Configuration needs to be changed in the SCM Git repositiory in the file `flux-crm/apps/base/siebel_observability/prometheus.yaml` under "metrics→additional_siebel_metrics" key as follows:

```
additional_siebel_metrics:
   process: true
   session: true
   statistics:
      server: false
      component:
         component_list: [ "EAIObjMgr_enu", "FINSObjMgr_enu"]
```

Add Custom Siebel Metrics

One can add Custom Siebel Metrics using server manager (srvrmgr) command output.

“metrics→custom_metrics→extension1" key is available out-of-the-box for this purpose as a sample.

An example is shown below to send the value of "PA_Value" from srvrmgr command output to Prometheus as metric "custom_MaxTasks". This will process server manager command "list param maxtasks for comp SWToolsObjMgr_enu" and send the PA_VALUE as value of metric named "custom_MaxTasks".

```
extension1:
- name: MaxTasks
     cmd: "list param maxtasks for comp SWToolsObjMgr_enu"
     value_column: "PA_VALUE"
     description: "Max Task of Siebel Webtools" prometheus_type: "Gauge"
     value_column: "PA_VALUE"
     type: "Gauge"
- name: MaxThreads
     cmd: "list param MaxThreads for comp EnergyObjMgr_enu"
     value_column: "PA_VALUE"
     description: "My Description 1"
     type: "Gauge"
- name: NumRetries
     cmd: "list param NumRetries for comp EnergyObjMgr_enu"
     value_column: "PA_VALUE"
     description: "My description 2"
     type: "Gauge"
```

Any other metrics obtained by processing srvrmgr commands can be included, to be collected and streamed to dashboard. Before including these commands in the `prometheus.yaml` file, it is suggested that you verify the accuracy of the commands used as well as the results returned.

For display in the dashboard, make necessary new dashboards or edit existing ones.

Additional Node Exporter Metrics

It is important to know that many collectors of Prometheus Node Exporter are disabled by default as a matter of practice. For more information, refer *https://github.com/prometheus/node_exporter#disabled-by-default*.

In Siebel CRM Observability - Monitoring solution, these can be enabled by providing a `--collector.<name>` flag under "`args`" section of `node-exporters-daemonset.yaml` and after that one must increment the "`version`" in the file `<namespace>-helmcharts/node-exporters/Chart.yaml`. This will deploy the necessary collector on OKE as part of GitOps.

Dashboards for Siebel CRM Monitoring

The Siebel Observability Monitoring solution offers two choices for viewing metrics on dashboards – Grafana, which is the Open Source option, and OCI Console which is an OCI-native solution.

Several sample dashboards are provided with both options. These are in English.

These leading dashboard solutions offer easy customizability. While intuitive, for most impactful use, refer to their official documentation for details.

This topic covers:

- *Using Grafana Dashboards*
- *Using OCI Dashboards*
- *Troubleshooting Common Dashboard Issues*

Using Grafana Dashboards

For more details about Grafana Dashboards, see *Key Software Components for Monitoring*. Note that Grafana server/service is not shipped by Oracle and must be hosted and managed by the users. Features described in this document were tested with Grafana version 9.4.3.

The Grafana sample dashboards are provided as JSONs which can be downloaded after enabling the monitoring feature and available via GET API: `/scm/api/v1.0/environment/<ENV_ID>/observability/download/sample_grafana_dashboards.zip`.

Here's a list of sample Grafana dashboards from Oracle for complete backend monitoring:

- Node Exporter Dashboard
- Kube-State-Metrics Dashboard
- Cadvisor Dashboard
- Jmx Dashboard
- Traefik Ingress Dashboard
- Siebel Server Dashboard
- Siebel Components Dashboard
- Siebel Block Volume and File System Storage Dashboard

The downloaded JSONs can be imported into Grafana for use with two steps:

1. Add SCM provisioned Prometheus endpoint as a Data Source in Grafana as follows:
   a. Log in to Grafana
   b. Add DataSource
   c. Select Prometheus

d. Add Prometheus URL from SCM
e. Choose GET as Https method
f. Save

2. Select the same data source as Prometheus DataSource during import of JSONs.

Visualizing OCI Infrastructure Metrics in Grafana

Follow these steps below to visualize OCI Infrastructure Metrics in Grafana.

For more information, refer:

- *https://grafana.com/grafana/plugins/oci-metrics-datasource/*
- *https://github.com/oracle/oci-grafana-metrics/blob/master/docs/using.md*

1. In Grafana, go to **Configurations > Plugins** and install the **Oracle Cloud Infrastructure Metrics** plugin.
2. Click **Create an Oracle Cloud Infrastructure Metrics Data Source** and provide appropriate "Connection Details" and save the "Datasource".
3. You can now use OCI Infrastructure metrics to build custom Dashboards in Grafana. You can verify the metrics from the metrics **Explore** tab in Grafana.

Using OCI Dashboards

OCI APM (Application Performance Monitoring) Dashboards are part of Oracle Cloud's observability and monitoring functionalities.

These can be accessed under APM Dashboard services in OCI Console. Sample dashboards from Siebel CRM Observability – Monitoring solution is available under SCM compartment and named "Siebel CRM - <namespace>".

Access steps:

1. Log in to OCI.
2. Navigate to Application Performance Monitoring Dashboards Service.
3. Change compartment to SCM compartment.
4. Sample dashboard is named "Siebel CRM - <namespace>".

These and other dashboards can be customized for your specific business requirements. Broad customization steps for OCI dashboard include:

- Identify different metrics from Metric Explorer service that meet your requirements.
- Utilize Monitoring Query Language from OCI to generate right output. For more information, refer *https://docs.oracle.com/en-us/iaas/Content/Monitoring/Reference/mql.htm*).
- Build new Dashboards based on the output from previous step.

Troubleshooting Common Dashboard Issues

NGINX Dashboard Loading Issue

The NGINX dashboard does not load if the `nginx_ingress_controller_requests` metric is missing or appears only with a status of 404 in the Prometheus UI.

To resolve this issue:

1. Inspect the NGINX Ingress Controller pod arguments as follows:

a. Execute the `kubectl describe` command to describe the NGINX Ingress Controller pod.
b. Verify whether the `metrics-per-host` argument was set to a value other than `false` in the `Containers.controller.Args` section or missing when the pod started.

2. If this argument is not set to `false` or is missing, perform the following steps to resolve the issue:
   a. Update the NGINX Ingress Helm chart as follows:
      i. Navigate to the environment Helm chart directory:
      ```
      cd /home/opc/siebel/<env_id>
      ```
      ii. Open the NGINX Ingress Helm chart.
      iii. Increment the chart version in `Chart.yaml` (for example, from 4.12.2 to 4.12.3).
      iv. Add the `perHost` parameter in `values.yaml` under `controller.metrics` as follows:
      ```
      perHost: false
      ```
      v. Update the template argument in the `ingress-nginx/templates/_params.tpl` file as follows:
      ```
      {{- if .Values.controller.metrics.enabled }}
      - --enable-metrics={{ .Values.controller.metrics.enabled }}
      - --metrics-per-host={{ .Values.controller.metrics.perHost }}
      {{- end }}
      ```
   b. Update the NGINX Ingress Flux release manifest as follows:
      i. Navigate to your environment Flux project in your environment directory.
      ```
      cd /home/opc/siebel/<env_id>/<Cloud manager repository name>
      ```
      ii. Edit the `infrastructure/nginx/release.yaml` file to set the NGINX release metric as follows:
      ```
      spec:
         ...
         metrics:
            enabled: true
            perHost: false
         ...
      ```
      iii. Commit the changes and push them to the repository:
      ```
      git add .
      git commit -m "perHost is set to false"
      git push
      ```
      Wait for the reconciliation process to complete and for the changes to be applied.
3. Verify that the NGINX dashboard loads as expected.

Siebel CRM Observability - Data Visualization and Metrics Monitoring for Oracle Databases

Oracle Database monitoring is the process of continuously observing the performance, health, availability, and resource utilization of Oracle databases. It ensures that the database runs efficiently, prevents downtime, and quickly detects and address issues before they impact users or business processes.

The Siebel CRM Observability and Monitoring solution is enhanced with the Oracle Database Metrics Exporter for Prometheus. It is integrated with SCM to provide unified visibility into Oracle databases deployed on Kubernetes – both on premises and in the cloud. It provides production ready, OpenTelemetry and Prometheus compatible metrics, and supports custom metrics that can be defined using a TOML (Tom's Obvious, Minimal Language) configuration file. You can monitor database health, operational behavior, and business-state metrics from live queries in real time and visualize this data in dashboards like Grafana. This supports faster troubleshooting, better decision-making, and a more

resilient Siebel CRM environment. It provides Oracle database metrics from metadata and also allows you to query any table and expose the data as metrics for monitoring in dashboards.

> **Note:** The Oracle Database Metrics Exporter for Prometheus is referred to as **Oracle DB Exporter** throughout this document.

This topic includes the following sections:

- *Prerequisites for Enabling Oracle Database Monitoring for BYOD*
- *Enabling Oracle Database Monitoring for Siebel CRM*
- *Disabling Oracle Database Monitoring in Siebel CRM*
- *Accessing Oracle Database Monitoring Metrics*
- *Configuring Oracle DB Exporter to Run Custom SQL Queries*
- *Troubleshooting Oracle DB Exporter Database Access Issues*

Prerequisites for Enabling Oracle Database Monitoring for BYOD

This section lists the prerequisites for enabling Oracle Database monitoring when using the BYOD option.

You must grant appropriate permissions to Oracle DB Exporter users before enabling Oracle Database monitoring. In a BYOR deployment, ensure that the database user used by Oracle DB Exporter to connect to the Oracle Database has the required privileges, based on the types of metrics being collected. The database users used by Oracle DB Exporter will need the following permissions:

Task	Permissions
To fetch built-in default metrics.	• CREATE SESSION • SELECT_CATALOG_ROLE
To run custom SQL queries against Siebel CRM table owner schema.	1. CREATE SESSION 2. SELECT access on the Siebel table owner schema

To create database users and grant permissions, connect to Oracle Database using the database admin user and do the following:

1. Create a user account to fetch database metrics:
   ```
   SQL> create user <metrics_user> identified by <password>;
   ```
2. Grant CREATE SESSION and SELECT_CATALOG_ROLE permissions to `<metrics_user>`:
   ```
   SQL> grant 'SELECT_CATALOG_ROLE' to <metrics_user>;
   SQL> grant CREATE SESSION to <metrics_user>;
   ```
3. Create a user to run custom queries:
   ```
   SQL> create user <custom_query_user> identified by <password>;
   ```
4. Grant CREATE SESSION to `<custom_query_user>` created in step 3:
   ```
   SQL> grant CREATE SESSION to <custom_query_user>;
   ```
5. Grant SELECT privileges on all Siebel CRM tables after the Siebel CRM database import completes; that is, after the `import_siebel_db` stage of Siebel CRM environment provisioning completes:
   ```
   SQL>
   BEGIN
      For t IN (SELECT table_name FROM all_tables where owner = '<table_owner_user in UPPERCASE>') LOOP
   ```

```
        EXECUTE IMMEDIATE 'GRANT SELECT ON ' || '<table_owner_user in UPPERCASE>' || '.' || t.table_name
 || ' TO ' || '<custom_query_user>';
   END LOOP;
END;
/
```

Enabling Oracle Database Monitoring for Siebel CRM

This section explains how to enable Oracle Database monitoring during a new Siebel CRM deployment or for an existing Siebel CRM environment. It includes the following subsections:

- *Enabling Oracle Database Monitoring during Siebel CRM Deployment*
- *Enabling Oracle Database Monitoring on an Existing Siebel CRM Deployment*
- *Enabling Oracle Database Monitoring on an Existing Siebel CRM Deployment with Siebel Monitoring Enabled*

Enabling Oracle Database Monitoring during Siebel CRM Deployment

To enable Oracle Database monitoring during Siebel CRM deployment, you must update the Siebel CRM deployment payload as follows:

- Set the `enable_oracle_db_monitoring` parameter to `true`.
- Configure the `auth_info` section based on the database type.

The following `observability` section snippet shows how to enable database monitoring when the database type is set to ATP or DBCS_VM:

```
{
   <Siebel deployment payload>
   "observability":
   {
      "siebel_monitoring": true,
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_password": "<metrics user's password>",
         "data_as_metrics_exporter_password": "<custom query user's password>"
      }
   }
}
```

The following `observability` section snippet shows how to enable database monitoring when the database type is set to BYOD:

> **Note:** You must ensure that all the prerequisites listed in the *Prerequisites for Enabling Oracle Database Monitoring for BYOD* section are met.

```
{
   <Siebel deployment payload>
   "observability":
   {
      "siebel_monitoring": true,
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_username": "<metrics user's username>",
         "db_metrics_exporter_password": "<metrics user's password>",
         "wallet_path": "<db wallet path>",
         "tns_connection_name": "<tns connection name>",
```

```
            "data_as_metrics_exporter_username": "<custom query user's username>",
            "data_as_metrics_exporter_password": "<custom query user's password>"
         }
      }
   }
```

For more information on the parameter, see *Payload Parameters for Siebel CRM Deployment*.

Enabling Oracle Database Monitoring on an Existing Siebel CRM Deployment

You can enable Oracle Database monitoring on an existing Siebel CRM environment that was deployed using SCM by submitting a POST request using the `observability` API.

To enable Oracle Database monitoring on an existing environment:

1. Create the JSON payload:
   - When the database type is set to ATP or DBCS_VM as shown below:

```
{
   "observability":
   {
      "siebel_monitoring": true,
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_password": "<metrics user's password>",
         "data_as_metrics_exporter_password": "<custom query user's password>"
      }
   }
}
```

   - When the database type is set to BYOD as shown below:

> **Note:** You must ensure that all the prerequisites listed in the *Prerequisites for Enabling Oracle Database Monitoring for BYOD* section are met.

```
{
   "observability":
   {
      "siebel_monitoring": true,
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_username": "<metrics user's username>",
         "db_metrics_exporter_password": "<metrics user's password>",
         "wallet_path": "<db Wallet path>",
         "tns_connection_name": "<tns connection name>",
         "data_as_metrics_exporter_username": "<custom query user's username>",
         "data_as_metrics_exporter_password": "<custom query user's password>"
      }
   }
}
```

2. Submit the payload using a POST request to the `observability` API, as shown below:

```
POST https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/environment/<ENV_ID>/observability
```

For more information on the parameters, see *Payload Parameters for Siebel CRM Deployment*.

Enabling Oracle Database Monitoring on an Existing Siebel CRM Deployment with Siebel Monitoring Enabled

You can enable Oracle Database monitoring on an existing Siebel CRM environment that already has Siebel CRM monitoring enabled by submitting a POST request using the `observability` API. In this case, since Siebel CRM monitoring is already enabled, you need not set the `siebel_monitoring` parameter to `true`.

To enable Oracle Database monitoring on an existing environment that has Siebel CRM monitoring enabled:

1. Create the JSON payload:
   - When the database type is set to ATP or DBCS_VM as shown below:

```
{
   "observability":
   {
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_password": "<metrics_user's password>",
         "data_as_metrics_exporter_password": "<custom_query_user's password>"
      }
   }
}
```

   - When the database type is set to BYOD as shown below:

> **Note:** You must ensure that all the prerequisites listed in the *Prerequisites for Enabling Oracle Database Monitoring for BYOD* section are met.

```
{
   "observability":
   {
      "enable_oracle_db_monitoring": "true",
      "auth_info": {
         "db_metrics_exporter_username": "<metrics user's username>",
         "db_metrics_exporter_password": "<metrics user's password>",
         "wallet_path": "<db Wallet path>",
         "tns_connection_name": "<tns connection name>",
         "data_as_metrics_exporter_username": "<custom query user's username>",
         "data_as_metrics_exporter_password": "<custom query user's password>"
      }
   }
}
```

2. Submit the payload using a POST request to the `observability` API, as shown below:

```
POST https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/environment/<ENV_ID>/observability
```

For more information on the parameters, see *Payload Parameters for Siebel CRM Deployment*.

Disabling Oracle Database Monitoring in Siebel CRM

You can disable Oracle Database monitoring by setting value of the `enable_oracle_db_monitoring` parameter to `false`.

To disable Oracle Database monitoring:

1. Create the JSON payload as shown below:

```
{
   "observability": {
      "enable_oracle_db_monitoring": false
```

```
    }
}
```

2. Submit the payload using a POST request to the `observability` API, as shown below:
   `POST https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/environment/{env_id}/observability`

For more information about the parameters, see *Payload Parameters for Siebel CRM Deployment*.

Accessing Oracle Database Monitoring Metrics

You can access the Oracle Database monitoring metrics through Prometheus dashboard and visualize the metrics through Grafana dashboard.

This topic contains the following sections:

- *Prometheus Dashboard*
- *Sample Grafana Dashboard*

Prometheus Dashboard

You can access the Prometheus dashboard either through the load balancer URL or a node port, depending on the ingress service type. The Prometheus dashboard URL is available in the `urls` section of the Siebel CRM deployment API response and in the GET response of the `environment` API. You can use the URL ending with `/Prometheus` to access the Prometheus dashboard. All metrics prefixed with `oracledb_` represent the Oracle Database monitoring metrics in the Prometheus dashboard.

Sample Grafana Dashboard

You can download and access the sample Grafana dashboard through the API GET request as follows:

- Oracle Database dashboard:
  `GET https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/environment/{env_id}/observability/download/Oracle-Database-Dashboard.json`
- Order Management dashboard for the telecommunications industry:
  `GET https://<SCM_instance_IP>:<port_num>/scm/api/v1.0/environment/{env_id}/observability/download/Oracle-Order-Management-Database-Dashboard.json`

Configuring Oracle DB Exporter to Run Custom SQL Queries

In addition to collecting a standard set of default metrics, Oracle DB Exporter also allows you to define and collect custom metrics by running your own SQL queries. This helps you monitor application-specific performance indicators or business data that are not captured by the default database metrics.

You can use custom SQL queries to tailor your observability solution by monitoring and collecting business critical metrics, such as Siebel version, transaction counts, or any custom key performance indicators, from Oracle Database. These metrics can then be visualized through Prometheus and Grafana dashboards.

To execute custom SQL metrics using Oracle DB Exporter:

1. Go to the `oracle-db-monitoring` Helm chart directory:
   `cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/oracle-db-monitoring`

2. Go to the templates directory:
```
cd templates
```
3. Edit the `custom-metric-config.yaml` file to define a new metric. Under the `custom-metric.toml` section, add the new metric block as shown below:
```
[[metric]]
context = <context>
labels = <labels>
metricsdesc = { <metricdescription> }
request = <SQLQuery>
databases = <databases>
```

   The variables in the example have the following values:

   - `[[ metric ]]` is used to define a new metric block. You can have multiple `[[ metric ]]` blocks in the YAML file for different queries.
   - `<context>` is used to group related metrics together. Oracle DB Exporter will prefix metrics with this context. For example:
     ```
     context = "siebel_app_version"
     ```
   - `<labels>` is used to define extra Prometheus labels to attach to the metric. For example:
     ```
     labels = [ "version" ]
     ```
   - `<metricdescription>` is used to map the metric name to a user-friendly description. For example:
     ```
     metricsdesc = { siebel_app_version="Siebel application version" }
     ```
   - `<SQLQuery>` is used to specify the SQL query that Oracle DB Exporter will run against the Oracle Database. For example:
     ```
     request = ''' select 1 as siebel_app_version, APP_VER as version from
      {{.Values.oracleDBMonitoring.table_owner_user }}.S_APP_VER where ROWNUM = 1 '''
     ```

     In the SQL query above, `{{ .Values.oracleDBMonitoring.table_owner_user }}` is a Helm template variable that is used to dynamically insert the table owner user during deployment, based on the value set for `oracleDBMonitoring.table_owner_user` in the Oracle Database monitoring Helm chart's `values.yaml` file. You must not change the value of this parameter; Oracle DB Exporter automatically will replace this placeholder with the correct schema value during deployment.
   - `<databases>` is used to define the target of the SQL query. You can assign one of the following values to this parameter:
     - `ForDataAsMetricsExport`: Use this value when your custom query targets Siebel application tables.
     - `ForDBMetricsExport`: Use this value when your custom query targets database objects other than Siebel tables.

     > **Note:** If you set the database value incorrectly, it may cause connection or privilege issues during metric collection.

     Example of `[[metric]]` block:

     ```
     [[metric]]
      context = "siebel_app_version"
      labels = [ "version" ]
      metricsdesc = { siebel_app_version="Siebel application version" }
      request = '''
      select 1 as siebel_app_version, APP_VER as version
      from {{ .Values.oracleDBMonitoring.table_owner_user }}.S_APP_VER
      where ROWNUM = 1
      '''
     ```

```
databases = [ "ForDataAsMetricsExport" ]
```

> **Note:** Prometheus is designed to ingest and store numeric time-series data. It does not natively support string values as the output for metrics. When you want to capture string information, such as a Siebel CRM application version, you must encode this information in a way that Prometheus can handle. In the above custom metric definition, version is specified as a label, Prometheus stores a metric like `oracledb_siebel_app_version_siebel_app_version{version="25.9"}` with the value set to `1`. The string value ("25.9") is attached as a label to the metric.

4. Save the `custom-metric-config.yaml` file.
5. Increment the chart version as follows:
   a. Go to the `oracle-db-monitoring` Helm chart directory.
   b. Open the `Chart.yaml` file and increment the version number. For example, update the version 0.1.0 to 0.1.1.
6. Commit the changes to the Git repository:

```
git add .
git commit -m "Oracle DB monitoring helmchart changes."
git push
```

7. Verify that the latest commit listed by Flux matches the commit you just pushed. This confirms that Flux has detected and synchronized your changes:

```
flux get source git -n <env_namespace>
```

8. After the reconciliation is completed, exec into the Oracle DB Exporter pod and verify that `configMap` includes the new custom metric as follows:

```
kubectl exec -it <oracle-db-monitoring pod> -n <namespace> -- cat /oracle/observability/custom-metric.toml
```

9. Verify the output in Prometheus as follows:
   a. Open the Prometheus user interface.
   b. Run the query for your new metric to view the SQL query output as reported by Oracle DB Exporter.

```
oracledb_<context>_<key of metricsdesc>
```

Troubleshooting Custom Metrics Issues

If you don't see your custom metrics in Prometheus, or encounter errors during configuration, follow these steps to resolve the issue:

1. Ensure that the syntax in your `custom-metric.toml` file is correct.
2. Ensure your SQL query returns at least one numeric column per metric (required for Prometheus ingestion). If you need to report a string value, use a constant numeric value (like `1`) as the metric and return the string field as a label.
3. Ensure that the metrics users have the required privileges assigned, in case of BYOD. For more information, see *Prerequisites for Enabling Oracle Database Monitoring for BYOD* .
4. Ensure that the `custom-metric-config.yaml` file changes are saved and the chart version has been incremented. After committing the changes, ensure that the GitOps or Helm reconciliation completes without errors.
5. Check the container logs for any ERROR messages related to scraping custom metrics:

```
kubectl logs <oracle-db-monitoring pod> -n <namespace>
```

6. Log in to the Oracle DB Exporter pod and view the `custom-metric.toml` file to confirm your changes:

```
kubectl exec -it <oracle-db-monitoring pod> -n <namespace> -- cat /oracle/observability/custom-metric.toml
```

Troubleshooting Oracle DB Exporter Database Access Issues

If the Oracle DB Exporter is unable to connect to the database, or if the Grafana dashboard displays the database status as "Dead," you must review the `oracle-db-monitoring` pod logs for the error "ORA-12154: TNS: could not resolve the connect identifier specified" or other database connection issues. If this error appears in the logs, perform the following steps to resolve the issue:

1. Verify that the `/tnsadmin/` wallet directory inside the container includes the correct wallet files:
   ```
   kubectl exec -it <oracle-db-monitoring-pod> -n <namespace> -- ls -l /tnsadmin/
   ```
2. If the wallet directory is empty, check the Kubernetes secret for wallet data. Confirm that the Kubernetes secret (for example, `db-wallet-secret`) includes the wallet data:
   ```
   kubectl get secret db-wallet-secret -n <namespace> -o yaml
   ```

   Note: Make sure that the `.data` section in the Kubernetes secret includes non-empty wallet files.
3. If the Kubernetes secret in the `.data` section is empty, disable Oracle DB monitoring, and then re-enable it to regenerate and populate the secret. For more information, see *Disabling Oracle Database Monitoring in Siebel CRM* and *Enabling Oracle Database Monitoring for Siebel CRM*.

8 Log Analytics in Siebel CRM Deployments

Log Analytics in Siebel CRM Deployments

This chapter describes how you can monitor the backend of a Siebel CRM deployment created using Siebel Cloud Manager (SCM) using the **Siebel CRM Observability - Monitoring and Log Analytics** module. This module helps you align your Siebel CRM deployment with cloud-native best practices.

The observability stack uses best of breed tools, such as Prometheus, Grafana, Oracle OpenSearch, Fluentd, OCI services, and other supporting components.

The **Siebel CRM Observability - Monitoring and Log Analytics** module includes the following components:

- Siebel CRM Observability – Monitoring
- SiebelCRM Observability – Log Analytics

The **Siebel CRM Observability - Log Analytics** feature helps you ingest, search, analyze, visualize, and generate actionable insights from logs for important components in your Siebel CRM deployment created using SCM.

This feature is optional. You can enable and manage it by using SCM.

This feature offers integration with two major log analytics solutions with distinctive benefits:

- Oracle OpenSearch – which is the Open Source option
- OCI Logging Analytics – which is a powerful OCI Native solution

This chapter contains the following topics:

- *Log Analytics in Siebel CRM Deployments*
- *Log Analytics Tooling Options*
- *Solution Architecture and Components*
- *Log Collection and Aggregation*
- *Sample Dashboards*
- *Pre-requisites for Enabling OCI Logging Analytics*
- *Enabling Log Analytics in Siebel CRM Observability*
- *Accessing Log Analytics URLs*
- *Oracle OpenSearch Usage in Siebel CRM Observability – Log Analytics*
- *OCI Logging Analytics Configurations of Importance*
- *OCI Logging Analytics Usage in Siebel CRM Observability – Log Analytics*
- *Extending Siebel CRM Observability – Log Analytics*

Features and Benefits

Here are some of the features of this modern solution offering log analytics integration for Siebel CRM deployments on OCI OKE done by SCM:

- Visualization and analysis of Siebel CRM logs directly from the browser.

- Powerful and effective search capabilities – see all occurrences at once across all files.
- Leading text analysis tools, both Open Source and OCI-native, at disposal.
- Ability to publish to dashboards easily – values and trends.
- Alerting capabilities based on logs.
- GitOps-based operation and management.
- Easy customization of dashboards.
- Extensible by integration of additional third-party tooling.

Some of the benefits that may accrue to the users by utilizing the above set of functionalities:

- Analyzing logs without needing access to server host machines:
  - Have at-a-glance, near real-time view of system-health of your Siebel CRM backend.
  - Search across all log files without guesswork.
  - Engage tech resources from any geography for debugging business-critical issues.
- Reducing system downtimes and performance degradations – improving user experience:
  - Use organization-specific criteria, set up alerts and perform preventive maintenance.
  - Track error occurrences based on parameter changes.
  - Store and analyze past occurrences of log patterns.
  - Predict future occurrences of errors and plan self-healing – get ready to move into AIOps.
- Generating business insights and measuring effectiveness of investments:
  - Analyze logs, generate reports on offering-effectiveness and on user experience.
  - Gain insight on user behavior.
  - Detect suspicious activities near real-time and prevent fraud.

Log Analytics Tooling Options

You can use the following log analytics options:

- *Oracle OpenSearch*
- *OCI Logging Analytics*

Oracle OpenSearch

The Oracle OpenSearch includes:

- OpenSearch engine (which is derived from Elasticsearch 7.10.2 and enhanced thereafter).
- OpenSearch Dashboards (which is derived from Kibana 7.10.2 and enhanced thereafter, Kibana being the visualization component of the Elastic Open Source Stack).

Oracle OpenSearch supports various search options and techniques including search by field, searching multiple indices, boosting, score based ranking, and various sorting and aggregation options. Also supports ANN based search engines like NMSLIB, FAISS, and LUCENE. Leveraging all capabilities can help users make best strides in logs and other data analysis for their Siebel CRM deployments.

OpenSearch Dashboards is visualization tool for data in OpenSearch. Oracle OpenSearch stack also allows using plugins to enhance search, analysis, perform anomaly detection, and so on.

Oracle OpenSearch is under active development by Oracle Corporation.

OCI Logging Analytics

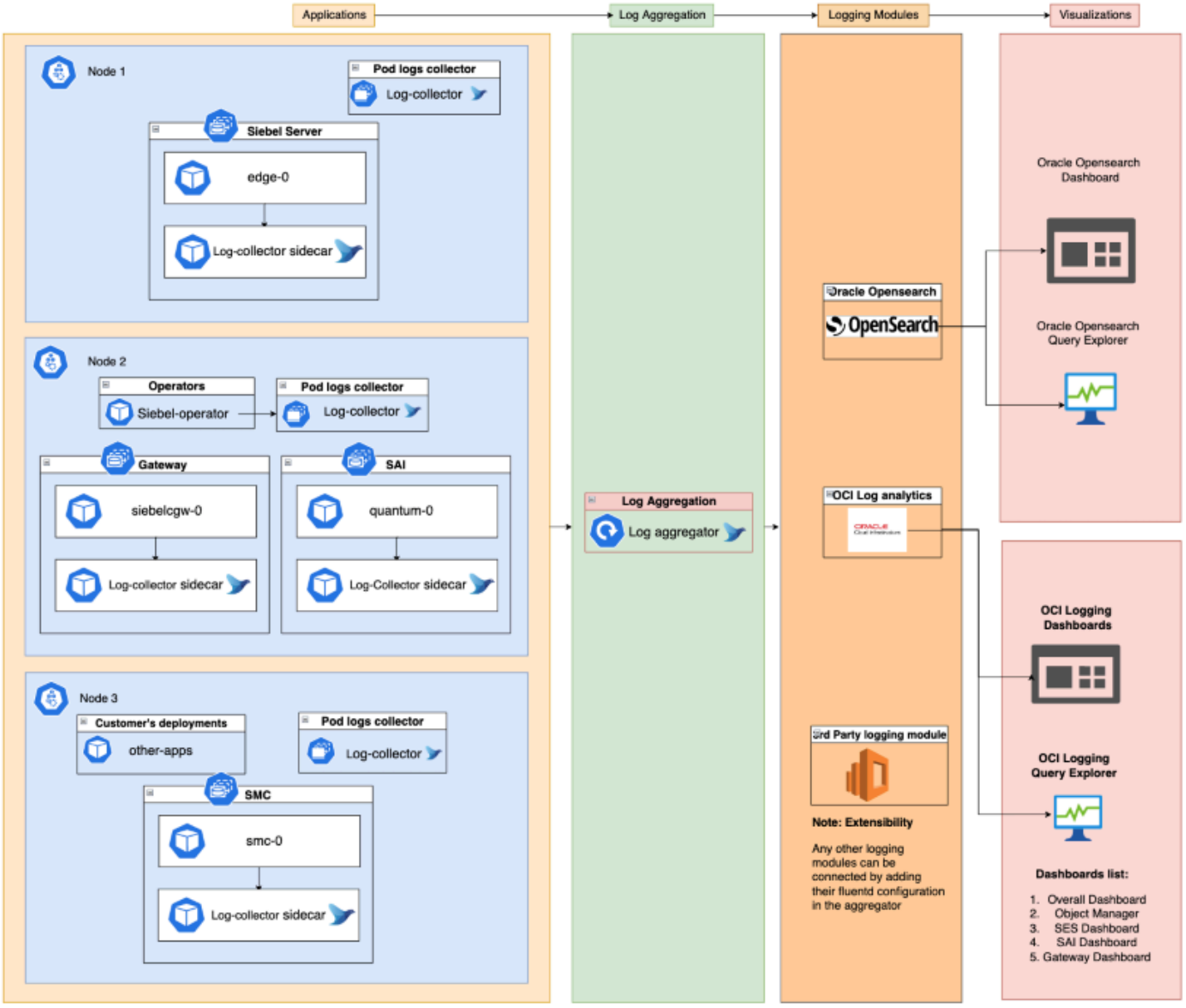
OCI Logging Analytics is an OCI-native cloud solution.

OCI Logging Analytics is a powerful SaaS analytics offering that enables users index, enrich, aggregate, explore, search, analyze, correlate, visualize, and monitor all log data. These analysis capabilities are further augmented by the included wide array of powerful management dashboards.

It also offers and supports wide array of developer tools and offers most detailed logging analytics for various OCI services that might be in use in your enterprise including, but not limited to, the ones for your Siebel CRM deployment done using SCM.

For details about the OCI Logging Analytics solution, refer *https://docs.oracle.com/en-us/iaas/logging-analytics/home.htm*.

Solution Architecture and Components



In the "Siebel CRM Observability – Log Analytics" solution, the primary log collection and aggregation tasks are done by the agents of application Fluentd.

Fluentd is a Cloud Native Computing Foundation (CNCF) graduated open source log data handler. Fluentd helps unify all facets of processing log data: collecting, filtering, buffering, and streaming across multiple sources and destinations. It has a flexible plugin system that allows the users to extend its functionality with minimal development efforts. Fluentd does not need significant physical resources to function and can be easily setup for high availability. It is one of the leading log collection and aggregation engines in the cloud native ecosystem.

For collecting logs from pods running Siebel Server deployment components like AI, Server, Gateway and so on, Fluentd log-collector agents are deployed as sidecars.

In addition, all nodes contain a Fluentd log collector running as a Kubernetes DaemonSet to collect logs that are not specifically covered by the sidecar Fluentd agents.

Logs collected by collector agents are passed onto the log aggregation agents of Fluentd. These, in turn, pass on the necessary data to the Logging Analytics components like Oracle OpenSearch and OCI Log Analytics – which have Dashboards and other UI components available to view, query, and analyze logging and other data available in the logging analytics layer.

Log Collection and Aggregation

The primary log collection from Siebel Servers, SMC, AI and Gateway are done by Fluentd log collector agents. They run as sidecars to all the pods of these applications, collect necessary logs per configurations and forward them to the Fluentd Log Aggregator agents.

Log aggregators are another Fluentd agent type running as a standalone (not sidecar, that is) deployment which receives the logs which are forwarded by the log collectors previously mentioned, and transform them to be pushed to the enabled modules such as OCI Logging Analytics and/or Oracle OpenSearch.

DaemonSet pod log collectors: The logs which are generated by pods are present in the `/var/log/containers/` location. These logs are collected by a daemon running in all the nodes and streamed to OCI Logging Analytics and/or Oracle OpenSearch.

Sample Dashboards

The "Siebel Observability – Log Analytics" solution provides several out-of-the-box sample dashboards.

- **Overall Dashboard**: Provides details about the number of logs streamed, pod wise logs streamed, and so on.
- **SES Dashboard**: Provides details about the number of logs streamed in the edge servers hosting Siebel servers, pod wise logs streamed etc.
- **SAI Dashboard**: Provides details about the number of logs streamed in the quantum servers hosting Siebel AIs, pod wise logs streamed, among others.
- **OM Dashboard**: Provides details about the event occurrences, Object manager occurrences, log count, pod wise log count, word cloud of the error codes, SBL error occurrences and such logs.
- **Gateway Dashboard**: Provides details about the number of logs streamed in the gateway servers, pod wise logs streamed, and so on.

Pre-requisites for Enabling OCI Logging Analytics

The pre-requisites for enabling OCI Logging Analytics are two folds from a functional standpoint:

- *Enable Log Analytics at Region Level*

- *Add Policies for Creating Logging Resources*

Enable Log Analytics at Region Level

To enable log analytics at region level:

1. Log in into your OCI console.
2. Choose the appropriate region.
3. Navigate inside the hamburger menu to **Observability & Management** > **Logging Analytics** > **Home**.
4. Click **Start Using Logging Analytics** to enable log analytics for the current region.

Add Policies for Creating Logging Resources

You need to add policies for creating logging resources in OCI.

Different logging analytics resources such as parsers, log groups, fields, and so on need to be created.

The policy required for creating all the resources and streaming, one may need to create the necessary instance principals or user principals. The command below will allow the `dynamic-group` named `{cm_namespace}-instance-principal-group` to create and use OCI Logging Analytics.

```
Allow dynamic-group
 {cm_namespace}-instance-principal-group to MANAGE loganalytics-features-family in
 tenancy
```

For complete details on such setups, refer OCI documentation: *https://docs.oracle.com/en-us/iaas/logging-analytics/doc/iam-policies-catalog-logging-analytics.html*.

Enabling Log Analytics in Siebel CRM Observability

To enable only Log Analytics module, Siebel CRM deployment payload for SCM to contain:

```
{
   ...
   "observability": {
      "siebel_logging": true,
      "enable_oracle_opensearch": true,
      "enable_oci_log_analytics": true,
      "oci_config": {
         "oci_config_path": "/home/opc/siebel/oci-config/config1",
         "oci_private_api_key_path": "/home/opc/siebel/oci-config/mykey.pem",
         "oci_config_profile_name": "DEFAULT"
      }
   }
}
```

For more information about the parameters, see *Payload Parameters for Siebel CRM Deployment*.

- To enable log analytics, the parameter "siebel_logging" is to be set to true. Only if siebel_logging is enabled, OCI Log Analytics or Oracle OpenSearch can be enabled.
- To enable both the logging modules, both the `enable_oci_log_analytics` and `enable_oracle_opensearch` parameters have to be set to true, and the `oci_config` parameter has to be passed with relevant parameters.
- To enable only Oracle OpenSearch, the `enable_oracle_opensearch` parameter has to be set to `true`. The `oci_config` parameters are not needed.

If OCI Log Analytics has to be enabled in a BYOR Deployment, Siebel CRM deployment payload for SCM should contain:

```
{
   ...
   "observability": {
      "siebel_logging": true,
      "enable_oracle_opensearch": true,
      "enable_oci_log_analytics": true,
      "oci_config": {
         "oci_config_path": "/home/opc/siebel/oci-config/config1",
         "oci_private_api_key_path": "/home/opc/siebel/oci-config/mykey.pem",
         "oci_config_profile_name": "DEFAULT"
      },
      "oci_log_analytics": {
         "smc_log_group_id": "ocid1.loganalyticsloggroup.oc1.uk-.......",
         "sai_log_group_id": "ocid1.loganalyticsloggroup.oc1.uk-......",
         "ses_log_group_id": "ocid1.loganalyticsloggroup.oc1.uk-......",
         "gateway_log_group_id": "ocid1.loganalyticsloggroup.oc1.uk-....",
         "node_logs_log_group_id": "ocid1.loganalyticsloggroup.oc1.uk-london-.....",
         "log_source_name": "scm-log-source"
      }
   }
}
```

For more information about the parameters, see *Payload Parameters for Siebel CRM Deployment*.

To enable OCI Log Analytics when BYOR (Use existing resources) was chosen during SCM installation, `enable_oci_log_analytics` has to be set to true and all values under `oci_log_analytics` have to be provided.

Disabling Log Analytics in Siebel CRM

You can disable OCI logging that was enabled earlier through the `enable_oci_log_analytics` parameter. Setting the value of `enable_oci_log_analytics` to `false` stops the streaming of logs to OCI Logging Analytics.

To disable OCI logging, include the `enable_oci_log_analytics` parameter in the Siebel CRM deployment payload as follows:

```
{
     "observability": {
          "enable_oci_log_analytics": false
     }
}
```

For more details on the payload elements, see the "observability" parameters in *Payload Parameters for Siebel CRM Deployment*.

Accessing Log Analytics URLs

- **Oracle OpenSearch**: Oracle OpenSearch and Oracle OpenSearch Dashboards helm charts are installed in the OKE cluster where the siebel applications are installed. The dashboard URL's are exposed via a load balancer URL. The URL is available as a part of the deployment API response as well as in the response to GET of environment API. The URL is contained in the section "urls" in the deployment API response. An URL ending in `/opensearch/app/home` is published where the dashboards can be accessed.

- **OCI Logging Analytics**: Logging Analytics is a managed service provided by OCI. To access the oracle dashboard, navigate to hamburger menu in the OCI console, click on Observability & Management, then choose Log Analytics followed by Dashboards. Choose the compartment where the siebel environment is deployed, which will be named like `{namespace}_compartment`.

Oracle OpenSearch Usage in Siebel CRM Observability – Log Analytics

Oracle OpenSearch offers an intuitive interface for querying which can reached via clicking hamburger menu - **Oracle OpenSeach** > **Discover**. Then choose the desired index pattern, which is logical representation of a set of log files, on the left and enter search string.

For example, to search a string across all files, choose **all_logs** index pattern.

Dashboards Query Language (DQL) provides additional controls for more complex query and analysis. Fields under index patterns can be incorporated for DQL based search and timeframes can be chosen to narrow down search results.

OCI Logging Analytics Configurations of Importance

These configuration elements form the building blocks of Siebel CRM Observability - OCI Logging Analytics configurations:

- *Log Sources*
- *Parsers and Fields*
- *Log Groups*

Log Sources

Log Sources indicate the location of logs. In OCI Logging Analytics configuration, each Siebel Deployment is defined as {namespace}_source. Configuration for Log sources are located at **Hamburger menu** > **Observability & Management** > **Logging Analytics** > **Administration**.

Parsers and Fields are required for ingestion of logs from every log sources.

Parsers and Fields

Ingesting logs from log sources may use parsers to parse incoming data streams into defined fields and these fields can be used in query and in analytics functions like aggregation, grouping, statistical functions, among others. Oracle solution provides multiple parsers for Siebel logs. It is also easy to create one's own parsers – remember to use Creation Type as "User - created".

Log Groups

Log groups are effectively logical containers for logs to help in, for example, data security. This observability solution groups the log streams based on the application type – the following Log Groups are provided as samples:

- SAI log group
- SES log group
- SMC log group
- Gateway log group
- Node log group

OCI Logging Analytics Usage in Siebel CRM Observability – Log Analytics

Log Explorer is the dashboard for querying and visualizing log streams in OCI Logging Analytics. It is available under option Log explorer of Logging Analytics. Choosing the right compartment for corresponding Siebel CRM deployment is required.

Log Explorer utilizes a powerful query language that helps in forming advanced queries according to business requirements. Results of Query can be visualized in catchy graphics, saved, and used in dashboards.

In order to query on fields which are already present, a query can be written on top of that field name. For example - In order to fetch all the different types of error codes which got generated in the Object Manager, a query like this can be executed in the log explorer:

```
* | distinct SIEBEL_om_error_code
```

In order to look for a keyword "GenericError" in all the SAI logs, a query like this may be used:

```
GenericError and 'Log Group' in ('tsts48t683b-SAI-logGroup') | fields 'Original Log Content' | timestats
 count as logrecords by 'Log Group'
```

Extending Siebel CRM Observability – Log Analytics

To extend the Siebel CRM Observability – Log Analytics solution to integrate with other log analytics software than OCI Logging Analytics and Oracle OpenSearch, the configurations can be updated to stream the logs to those target services.

In the SCM deployed Siebel CRM environment, the component which pushes the log streams to any of these modules is the log aggregator. Log aggregator is a Fluentd agent which takes a Fluentd config and defines where the next stage of streaming is. The following describe the changes required to stream to new Log Analytics targets prevalent in your organization:

- *Updating the Fluend Aggregator Image*
- *Updating the Fluend Aggregator Configurations*

- *Rolling Out the Changes*
- *Verifying in the Target Logging Module*

Updating the Fluend Aggregator Image

Based on the type of logging module (for example, Splunk), do one of the following:

- Update the Fluentd aggregator container image.
- Build a new Fluentd aggregator container image to include the plugin gems for the corresponding module.

This will allow fluentd configuration to forward requests to the newly added logging software. All the gems needed for the module have to be available in the image.

FluentD aggregator image supplied with SCM already contains gems for modules such as Oracle OpenSearch, OCI Logging Analytics, and so on. When a new Logging module has to be added (for example, for Splunk), then the corresponding gem has to be installed into the container image using commands like the following (always check detailed documentation for the plugin for detailed instructions):

```
fluent-gem install fluent-plugin-splunk-enterprise
```

For more information, refer *https://github.com/fluent/fluent-plugin-splunk*.

Updating the Fluend Aggregator Configurations

The Fluentd log aggregator configuration can be found in the Helm charts Git repository in the file `siebel-logging/templates/log-aggregator-cm.yaml`.

The match block configurations for the log aggregator are contained in the files:

- `opensearch.conf` (if only Oracle OpenSearch is enabled)
- `logan.conf` (if only OCI Logging Analytics is enabled)
- `all.conf` (if both Oracle OpenSearch and OCI Logging Analytics are enabled)

A new match block must be added for forwarding to the required logging module. Note that in the match block, if the log data has to be pushed to more than one output/logging module, then Fluentd "store" tag has to be used within the same match block.

Example:

Fluentd supports Splunk as an output module. For more information, refer *https://docs.fluentd.org/0.12/output/splunk*.

A sample configuration of Fluentd aggregator to push logs to a Splunk endpoint can be the following:

```
<match splunk.**>
   @type splunk_tcp
   host example.com
   port 8089

   # format parameter
   format raw
   event_key log

   # ssl parameter
   use_ssl true
   ca_file /path/to/ca.pem

   # buffered output parameter
   flush_interval 10s
```

```
</match>
```

For more information on Splunk configuration docs for Fluentd, refer *https://github.com/fluent/fluent-plugin-splunk/blob/master/README.tcp.md*.

Note that network access between the Fluentd log aggregator and the logging module's host must be available on the specified port. In the example above, the log aggregator must be able to connect to example.com on port 8089.

Rolling Out the Changes

After all the changes outlined above are done, commands like the following can be executed to roll out the changes.

- Delete the existing log aggregator-cm:

  ```
  kubectl delete cm/log-aggregator-cm -n <namespace>
  ```

- Reconcile the changes:

  ```
  flux reconcile source git siebel-repo -n <namespace> && flux reconcile kustomization apps -n <namespace>
  ```

- Review that the match block is available now:

  ```
  kubectl get -o yaml cm/log-aggregator-cm -n <namespace>
  ```

- Rollout/restart the log aggregator deployment:

  ```
  kubectl rollout restart deploy/log-aggregator -n <namespace>
  ```

- The logs of the log collector should show log ingestion:

  ```
  kubectl logs deploy/log-aggregator -n <namespace>
  ```

Verifying in the Target Logging Module

Verify that the logs are appearing in the respective logging module, for example Splunk.

9 Siebel CRM Deployment Parameters

Overview

The chapter provides detailed information about the Siebel CRM deployment payload parameters. It includes the following topics:

- *Overview*
- *Parameter Usage Considerations*
- *Payload Parameters for Siebel CRM Deployment*

For examples of payloads and usage guidelines, see *Example Payload to Deploy Siebel CRM*.

Parameter Usage Considerations

You must consider the following points when using the parameters listed below:

- The `config_id` parameter is applicable only when provisioning a greenfield environment with a previously customized configuration. For more information, see *Customizing Configurations Prior to Greenfield Deployment*.
- The `database_type` and `industry` parameters are applicable only to greenfield deployments.
- The `db_type` parameter under database is not the same as `database_type`. Valid values for the `db_type` parameter are:
  - ATP (for Oracle Autonomous Database)
  - DBCS_VM(for Oracle Database Cloud Service)
  - BYOD
- The `bucket_url` parameter is used only for the deployment scenario that uses the Siebel Lift utility. This parameter is not used for greenfield deployments.

The users are advised to get familiarized with various notes before proceeding to the section on payload parameters.

Payload Parameters for Siebel CRM Deployment

The following table lists and provides details of the Siebel CRM deployment payload parameters:

Payload Parameter	Section	Description
name	(top level)	(Required) A short name for identification of the environment. This name is used as a prefix in all the resources. The namespace in the Kubernetes cluster is created with this name. Choose something meaningful and short (no more than 10 to 15 alphanumeric characters),

Payload Parameter	Section	Description
		such as DevExample (perhaps using the name of your company or organization).
config_id	(top level)	(Required for customization workflow) The configuration ID that is obtained as described in *Customizing Configurations Prior to Greenfield Deployment*. You specify this configuration ID in the payload only when you provision a greenfield environment with a configuration that you previously customized.
network_policy	(top level)	(Optional) Specifies the Kubernetes network policy configuration for SCM-managed Siebel CRM workloads. By default, SCM enables Kubernetes **NetworkPolicy** generation unless you explicitly disable it in the Siebel CRM deployment payload.
enabled	network_policy	(Optional) Specifies whether SCM generates Kubernetes **`NetworkPolicy`** resources for SCM-managed Siebel CRM workloads. Set this value to **`true`** to enable **`NetworkPolicy`** generation. When you omit **`network_policy.enabled`** or set it to **`true`**, SCM generates Kubernetes **`NetworkPolicy`** resources for SCM-managed CRM workloads. The generated policies restrict ingress to expected SCM and Siebel service paths, including ingress controller traffic, Siebel controller webhook traffic, observability metrics traffic, logging traffic, and OpenSearch traffic. You must set **`network_policy.enabled`** to **`false`** only for temporary compatibility or troubleshooting. When you set this parameter to **`false`**: • SCM disables generated CRM **`NetworkPolicy`** resources and chart-level **`NetworkPolicy`** values for that environment. • SCM does not apply the new **`NetworkPolicy`** restrictions for that Siebel environment. For more information, see *Using Network Policies*.
database_type	siebel	(Required for greenfield deployments) Specifies the database type to use for a greenfield deployment. The available options are: • Sample • Vanilla **Note:** This parameter is used only for greenfield deployments and is not used for the deployment scenario that uses the Siebel Lift utility.
industry	siebel	(Required for greenfield deployments) Specifies the industry-specific functionality to enable in a greenfield deployment. The available options are: • Automotive • Financial Services • Life Sciences • Sales • Service • Partner Relationship Management

Payload Parameter	Section	Description
		• Public Sector • Telecommunications • Loyalty • Consumer Goods • Hospitality Note: This parameter is used only for greenfield deployments and is not used for the deployment scenario that uses the Siebel Lift utility.
registry_url	siebel	(Required) Specifies the URL of the Open Container Initiative (OCI) compliant container registry. For example, for the Oracle Cloud Infrastructure container registry in the Ashburn region, you might use **`iad.ocir.io`**. For more information, see the registry concepts information in the Oracle Cloud Infrastructure documentation (*https://docs.oracle.com/en-us/iaas/Content/Registry/Concepts/registryprerequisites.htm*).
registry_user	siebel	(Required) Specifies the user ID to connect to the container registry. This user must have container registry access to push and pull images.
registry_password	siebel	(Required) Specifies the password or authentication token for this user.
registry_prefix	siebel	(Optional) Specifies a prefix that's appended after the **`registry_url`**. For OCI container registry, registry_prefix must be the tenancy namespace, if needed, you can add a suffix to it. As it's an optional field, it can be left blank.
architecture	siebel	(Required for Siebel Component Services deployments) Specifies the architecture for your Siebel CRM deployment. Specify one of the following values: • **SCS.** For deployments of Siebel Component Services. • **CRM.** For deployments of Siebel CRM. (This value is the default, if the parameter is omitted.)
bucket_url	siebel	Specifies the bucket that the Siebel Lift utility creates when it is run and uploads deployment artifacts. Create a pre-authenticated request URL for the bucket. The access type must permit object reads and the bucket must enable object listing. **Note:** This parameter is used only for the deployment scenario that uses the Siebel Lift utility and is not used for greenfield deployments. You must not configure this parameter if the **`nfs`** section is configured under the **`siebel`** section in the Siebel CRM deployment payload. To create a pre-authenticated request URL: 1. Launch the OCI console and navigate to Storage/Buckets. Open the bucket that is created during the lift process. 2. Click the Pre-Authenticated Requests link in the Resources section.

Payload Parameter	Section	Description
		3. Click Create Pre-Authenticated Request. 4. Provide a name, select the target as Bucket, and specify the access type as reads. 5. Select Enable Object Listing (check box).
keystore	siebel	(Optional) This parameter allows for Custom Keystore Management.
gateway_deployment_type	siebel	(Optional) Use this parameter to specify the Siebel Gateway deployment mode. Allowed values are **`standalone`** or **`cluster`**. Set the value of this parameter to: • **`standalone`** for creating a single Siebel Gateway node without any cluster features. • **`cluster`** for high availability with multiple Siebel Gateway nodes (this is the default). Note: If you configure the **`gateway_cluster_replica_count`** parameter, the gateway deployment type is automatically set to **`cluster`**. So, do not set **`gateway_deployment_type`** to **`standalone`** when **`gateway_cluster_replica_count`** is configured. Use only one of these options at a time to avoid conflicts.
gateway_cluster_replica_count	siebel	(Optional) Use this parameter to install and configure a gateway cluster, based on the specified number. Applies to both Greenfield and Lift-and-Shift deployments. The Siebel Gateway cluster requires a minimum of three replicas. We recommend using an odd number of replicas for optimal operation. By default, a three-node gateway cluster is created if this parameter is not overridden. If you override the parameter in the payload, the gateway cluster is created with the specified value instead. Note: If you configure the **`gateway_cluster_replica_count`** parameter, the gateway deployment type is automatically set to **`cluster`**. So, do not set **`gateway_deployment_type`** to **`standalone`** when **`gateway_cluster_replica_count`** is configured. Use only one of these options at a time to avoid conflicts.
security_adapter_type	siebel	(Optional) Specify the security adapter type. Supported values are 'DB' and 'LDAP'. Default value: DB.
path	siebel > nfs	This parameter is used to specify the NFS server directory path that holds the lifted Siebel CRM artifacts. Note: You must not configure this parameter if you have configured the **`bucket_url`** parameter in the Siebel CRM deployment payload.
server	siebel > nfs	This parameter is used to specify the NFS server endpoint. Note: You must not configure this parameter if you have configured the **`bucket_url`** parameter in the Siebel CRM deployment payload.
use_self_signed_certificate	siebel	(Required only if you want SCM to generate self-signed certificates.)

Payload Parameter	Section	Description
		This Boolean parameter controls whether SCM generates self-signed certificates, or you provide your own custom keystore. Set the value of this parameter to: • **`true`** to have SCM generate self-signed certificates. • **`false`**, or omit the parameter, to use your own custom certificate (keystore).
siebel_server_keystore_path	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the path to a custom keystore (.jks) file that contains the Siebel server and Siebel Controller certificates.
siebel_client_keystore_path	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the path to a custom keystore (.jks) file that contains the Siebel client certificate. For more information, see *Using Custom Keystore*.
siebel_truststore_path	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the path to a custom keystore (.jks) file that contains the root certificate. For more information, see *Using Custom Keystore*.
siebel_server_keystore_password	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the password for the keystore file specified in the **`siebel_server_keystore_path`** parameter. For more information, see *Using Custom Keystore*.
siebel_client_keystore_password	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the password for the keystore file specified in the **`siebel_client_keystore_path`** parameter. For more information, see *Using Custom Keystore*.
siebel_truststore_password	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the password for the truststore. For more information, see *Using Custom Keystore*.
siebel_server_certificate_alias	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the alias used when importing the Siebel server certificate into the keystore specified in the **`siebel_server_keystore_path`** parameter. For more information, see *Using Custom Keystore*.
siebel_client_certificate_alias	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the alias used when importing the Siebel client certificate into the keystore specified in the **`siebel_client_keystore_path`** parameter. For more information, see *Using Custom Keystore*.

Payload Parameter	Section	Description
siebel_controller_certificate_alias	siebel > keystore	(Required only when you want to use your own certificate.) This parameter specifies the alias used when importing the Siebel Controller certificate into the keystore specified in the **`siebel_server_keystore_path`** parameter. For more information, see *Using Custom Keystore*.
ldap_host_name	siebel > ldap	(Required) Host name of the ldap server for ldap authentication. Note that you may have to include the IP address if the server is configured to listen only with the IP address: You must specify the FQDN (fully qualified domain name) of the LDAP server, not just the domain name. For example, specify ldapserver.example.com, not example.com.
ldap_port	siebel > ldap	(Required) Specify the port number for the ldap for ldap authentication. For example, 389.
application_user_dn	siebel > ldap	(Required) Specify the user name of a record in the directory with sufficient permissions to read any user's information and do any necessary administration. This user provides the initial binding of the LDAP directory with the Application Object Manager when a user requests the login page, or else anonymous browsing of the directory is required. You enter this parameter as a full distinguished name (DN), for example "uid=appuser, ou=people, o=example.com" (including quotes) for LDAP. The security adapter uses this name to bind. You must implement an application user.
application_password	siebel > ldap	(Required) OCID of the secret containing the password for the user defined by the Application User Distinguished Name parameter. The secret must be stored encrypted in the vault. In an LDAP directory, the password is stored in an attribute and clear text passwords are not supported for the LDAPSecAdpt named subsystem.
base_dn	siebel > ldap	(Required) Specify the base distinguished name, which is the root of the tree under which users of this Siebel application are stored in the directory. Users can be added directly or indirectly after this directory. For example, a typical entry for an LDAP server might be: BaseDN = "ou=people, o=domain_name" where: • o denotes organization • ou denotes organization unit and is the subdirectory in which users are stored
credentials_attribute_type	siebel > ldap	(Required) Specify the attribute type that stores a database account. For example, if Credentials Attribute is set to dbaccount, then when a user

Payload Parameter	Section	Description
		with user name HKIM is authenticated, the security adapter retrieves the database account from the dbaccount attribute for HKIM. This attribute value must be of the form username=U password=P, where U and P are credentials for a database account. There can be any amount of space between the two key-value pairs but no space within each pair. The keywords username and password must be lowercase. In LDAP security adapter authentication to manage the users in the directory through the Siebel client, the value of the database account attribute for a new user is inherited from the user who creates the new user. The inheritance is independent of whether you implement a shared database account, but does not override the use of the shared database account.
password_attribute_type	siebel > ldap	(Required) Specify the attribute type under which the user's login password is stored in the directory.
roles_attribute_type	siebel > ldap	(Optional) Specify the attribute type for roles stored in the directory. For example, if Roles Attribute is set to roles, then when a user with user name HKIM is authenticated, the security adapter retrieves the user's Siebel responsibilities from the roles attribute for HKIM.
shared_db_credentials_dn	siebel > ldap	(Optional) Specify the absolute path (not relative to the Base Distinguished Name) of an object in the directory that has the shared database account for the application. If not set, then the database account is looked up in the user's DN as usual. If set, then the database account for all users is looked up in the shared credentials DN instead. The attribute type is determined by the value of the Credentials Attribute parameter. For example, if the Shared Database Account Distinguished Name parameter is set to "uid=HKIM, ou=people, o=example.com" when a user is authenticated, the security adapter retrieves the database account from the appropriate attribute in the HKIM record. This parameter's default value is an empty string.
shared_db_username	siebel > ldap	(Optional) Specify the user name to connect to the Siebel database. You must specify a valid Siebel user name and password for the Shared DB User Name and Shared DB Password parameters. Specify a value for this parameter if you store the shared database account user name as a parameter rather than as an attribute of the directory entry for the shared database account. To use this parameter, you can use an LDAP directory.
shared_db_password	siebel > ldap	(Optional) OCID of the secret containing the password associated with the Shared DB User Name parameter.
username_attribute_type	siebel > ldap	(Required) Specifies the attribute type under which the user's login name is stored in the directory.

Payload Parameter	Section	Description
		For example, if User Name Attribute Type is set to uid, then when a user attempts to log in with user name HKIM, the security adapter searches for a record in which the uid attribute has the value HKIM. This attribute is the Siebel user ID, unless the Security Adapter Mapped User Name check box is selected.
use_adapter_username	siebel > ldap	(Optional) If this boolean parameter is set to true, then when the user key name passed to the security adapter is not the Siebel User ID, then the security adapter retrieves the Siebel User ID for authenticated users from an attribute defined by the Siebel Username Attribute parameter.
siebel_username_attribute_type	siebel > ldap	This is mandatory parameter when 'use_adapter_username' is set to 'true' If set, then this parameter is the attribute from which the security adapter retrieves an authenticated user's Siebel User ID. If not set, then the user name passed in is assumed to be the Siebel User ID.
siebel_admin_username	siebel > ldap	(Required) The username of the Siebel CRM administrative user.
siebel_admin_password	siebel > ldap	(Required) OCID of the secret containing the Siebel CRM Administration User password.
anonymous_username	siebel > ldap	(Required) The username of the web anonymous user.
anonymous_user_password	siebel > ldap	(Required) OCID of the secret containing the anonymous user password which will be updated.
propagate_change	siebel > ldap	(Optional) This is a boolean flag. Set this parameter to True to allow administration of the directory through Siebel Business Applications UI. When an administrator then adds a user or changes a password from within the Siebel application, or a user changes a password or self-registers, the change is propagated to the directory. A non-Siebel security adapter must support the SetUserInfo and ChangePassword methods to allow dynamic directory administration.
hash_db_password	siebel > ldap	(Optional) This is a boolean flag. Set this parameter to True to specify password hashing for database credentials passwords. Hash Algorithm will be set to "SHA1", which is the default value, is read-only for the Siebel Gateway (SGW) security profile.
hash_user_password	siebel > ldap	(Optional) This is a boolean flag.

Payload Parameter	Section	Description
		Set this parameter to True to specify password hashing for user passwords. Hash Algorithm will be set to "SHA1", which is the default value, is read-only for the SGW security profile
salt_attribute_type	siebel > ldap	(Optional) This is a boolean flag. Specifies the attribute that stores the salt value if you have chosen to add salt values to user passwords. The default attribute is title.
salt_user_password	siebel > ldap	(Optional) This is a boolean flag. Set this parameter to True to specify that salt values are to be added to user passwords before they are hashed. This parameter is ignored if the Hash User Password parameter is set to False.
enable_ssl	siebel > ldap	(Optional) Specifies whether to enable SSL for connections to the LDAP server (that is, LDAP over SSL or, in short, LDAPs).
ldap_wallet_path	siebel > ldap	(Required only when **`enable_ssl`** is set to 'True') This parameter specifies the path to the wallet file required for LDAP over SSL connection. The wallet file (Example: ewallet.p12) wont be lifted during lift process and one needs to manually copy it to OCI SCM container location and pass the path in this payload parameter. You can also copy the wallet file to the SCM container using File Sync Utility, for more information see *Uploading Files to the SCM Container Using File Sync Utility*. Here, the wallet should be created from Oracle Wallet Manager and the Oracle wallet must contain CA server certificate that has been issued by Certificate Authorities to LDAP directory server.
ldap_wallet_password	siebel > ldap	(Required when **`enable_ssl`** is set to 'True') OCID of the secret containing the password to open the LDAP wallet that contains a certificate for the certificate authority used by the LDAP directory server.
resource_tags	siebel	(Optional) Specifies custom OCI tags to apply to supported OCI resources created by SCM. Use this section to define **`defined_tags`** and **`freeform_tags`**. If omitted or empty, SCM uses the default behavior and does not apply custom tag overrides.
defined_tags	siebel > resource_tags	(Optional) Specifies OCI defined tags to apply to supported resources. The tag namespaces and tag keys must already exist in the target OCI tenancy.

Payload Parameter	Section	Description
freeform_tags	siebel > resource_tags	(Optional) Specifies OCI free-form tags to apply to supported resources as key-value pairs. Free-form tags do not require preconfigured OCI tag namespaces or tag definitions.
security_context	siebel	(Optional) Defines the custom security context for Siebel CRM containers and pods. If you configure this section, you must configure the **run_as_user**, **run_as_group**, and **fs_group** parameters. If you omit this section, SCM uses **1000** for all three values.
run_as_user	siebel > security_context	(Required when **security_context** is specified) Specifies the user ID under which Siebel CRM container processes run. Note: On OpenShift, this value must be within the dynamic range specified by the **openshift.io/sa.scc.uid-range** namespace annotation.
run_as_group	siebel > security_context	(Required when **security_context** is specified) Specifies the group ID under which Siebel CRM container processes run.
fs_group	siebel > security_context	(Required when **security_context** is specified) Specifies the file system group ID for shared volume access. Note: On OpenShift, this value must be the first ID in the dynamic range specified by the **openshift.io/sa.scc.supplemental-groups** namespace annotation.
git_type	infrastructure > git	(Required) Used to specify the SCM Git provisioning type. Allowed values are 'gitlab' or 'byo_git'. If the value is set to 'gitlab'', SCM will create and manage the Git repositories. If the value is set to 'byo_git', SCM will read the SCM repository and Helm repository details from the payload and use the same during Siebel CRM provisioning.
commit_signing	infrastructure > git	(Optional) Specifies the Git commit signing configuration for the SCM Git repositories. Configure this section when you want Flux to apply repository changes only when the latest commit is signed by a trusted GPG key. For more information, see *Using Git Commit Signing*.
enabled	infrastructure > git > commit_ signing	(Optional) Specifies whether Git commit signing is enabled or not: • Set the value to **true** to enable Git commit signing. • Set the value to **false**, or omit the **commit_signing** section, to keep Git commit signing disabled. When signing is disabled, you do not need to provide a customer GPG public key file.
customer_public_key_file	infrastructure > git > commit_ signing	(Required when **commit_signing**.enabled is set to **true**) Specifies the path to the customer GPG public key file on the SCM host or container. The customer public key file must already exist on the SCM host or container. Note: Do not provide the customer private key to SCM.

Payload Parameter	Section	Description
git_protocol_type	infrastructure > git > byo_git	(Required when git_type is set to 'byo_git') Used to specify the protocol type to transfer data. Allowed values are 'ssh' and 'http'.
git_accesstoken	infrastructure > git > byo_git	(Required when git_protocol_type is set to 'http') Used to specify the access token with API scope for the Git user. You can create the access token in user settings.
git_ssh_private_key	infrastructure > git > byo_git	(Required when git_protocol_type is set to 'ssh') Used to specify the path of the ssh private key file required to access the Git repositories. Note: An encrypted private key protected with a passphrase isn't supported. You must ensure that the key is decrypted before you use it in the deployment payload. The current user must have read and write access to the private key, but other users should not have access to it. Hence, you must assign the private key permissions as follows: `chmod 600 <git_ssh_private_key>`
git_ssh_host_key_sha256	infrastructure> git > byo_git	(Required when git_protocol_type is set to 'ssh'.) Used to specify the verified SHA-256 host-key fingerprint for each Git host. Use the host name as the key for SSH port 22. For a non-default port, use `host:port` as the key. For more information, see the "Configuring SSH Host-Key Verification" section in *Using Existing Git Repositories*.
git_scm_repo_url	infrastructure > git > byo_git	(Required when git_type is set to 'byo_git') Used to specify the Git repository URL to use as the SCM repository. You can provide both http and https URLs in this parameter.
git_scm_repo_branch	infrastructure > git > byo_git	(Required when git_type is set to 'byo_git') Used to specify the branch of the Git repository to use for the SCM repository.
git_scm_flux_folder	infrastructure > git > byo_git	(Required when git_type is set to 'byo_git') Used to specify the folder to use for Flux bootstrap setup in the SCM repository. SCM will have full control over this folder. Hence, it's recommended to use a dedicated folder for the Flux folder in the SCM repository. Note: • If folder exists, SCM will clean up the existing content and continue with deployment. • If this folder doesn't exist, SCM will create the folder and continue with deployment.

Payload Parameter	Section	Description
git_helm_repo_url	infrastructure > git > byo_git	(Required Used to specify the Git repository URL to use as the Helm repository. Note: SCM will have full control over the Helm repository. So use a dedicated repository for the Helm repository.
git_helm_repo_branch	infrastructure > git > byo_git	(Required) Used to specify the Git repository branch to use for the Helm repository.
git_selfsigned_cacert	infrastructure > git > byo_git	(Optional) Used to specify the path to a self-signed certificate. For example, if you copy the Git certificate from the Git instance to the SCM instance in the **`/home/opc/cmapp/<CM_RESOURCE_PREFIX>/certs`** directory, which is volume mounted to the **`/home/opc/certs`** directory in the SCM container, you can assign **`/home/opc/certs/rootCA.crt`** as the value for this parameter. You can also copy the certificates to SCM using File Sync Utility, for more information see *Uploading Files to the SCM Container Using File Sync Utility*.
git_url	Infrastructure > git > gitlab	(Required) Used to specify the URL for the GitLab instance.
git_user	infrastructure > git > gitlab	(Required) Used to specify the user with access to create GitLab projects in the specified GitLab instance.
git_accesstoken	infrastructure > git > gitlab	(Required) Used to specify the access token, with the API scope, for the GitLab user. You can create the access token in user settings.
git_selfsigned_cacert	infrastructure > git > gitlab	(Required) Used to specify the path to a self-signed certificate. For example, if you copy the Git certificate from the Git instance to the SCM instance in the "**`/home/opc/cmapp/<CM_RESOURCE_PREFIX>/certs`**" directory, which is volume mounted to the "**`/home/opc/certs`**" directory in the SCM container, you can assign "**`/home/opc/certs/rootCA.crt`**" as the value for this parameter. You can also copy the certificates to SCM using File Sync Utility, for more information see *Uploading Files to the SCM Container Using File Sync Utility*.
siebel_lb_subnet_cidr	infrastructure	(Required for advanced network configuration) CIDR range for Load Balancer subnet. For more information about CIDR ranges for subnets, see *Using Advanced Network Configuration*.
siebel_private_subnet_cidr	infrastructure	(Required for advanced network configuration) CIDR range for Kubernetes worker nodes private subnet.
siebel_db_subnet_cidr	infrastructure	(Required for advanced network configuration) CIDR range for the database private subnet.
siebel_cluster_subnet_cidr	infrastructure	(Required for advanced network configuration) CIDR range for OKE cluster subnet (Kubernetes API server).

Payload Parameter	Section	Description
siebel_lb_subnet_ocid	infrastructure	(Required for using existing VCN resource) OCID of the regional subnet where the Load Balancer will be attached. Allow TCP port 443 from your client network where the users will access Siebel application.
siebel_private_subnet_ocid	infrastructure	(Required for using existing VCN resource) OCID of the subnet where the OKE worker nodes will be attached. The following needs to be ensured: • Allow all TCP port traffic in the same subnet, enables pods on one worker node to communicate with pods on other worker nodes. • Allow all TCP port traffic from cluster subnet, enables Kubernetes control plane to communicate with worker nodes. • Allow TCP port 22 ingress from SCM instance subnet/VCN, for SCM to SSH to worker nodes. • Allow egress rule for "All <region> Services in Oracle Services Network", enables accessing OCI container registry and other OCI services from worker nodes. • Allow egress for TCP port 6443, 12250 to cluster subnet, enables Kubernetes worker to Kubernetes API endpoint communication and Kubernetes worker to control plane communication. • Allow ICMP Port 3 and 4 for instances to receive Path MTU Discovery fragmentation messages.
siebel_db_subnet_ocid	infrastructure	(Required for using existing VCN resource) OCID of the subnet where the Database will be created. The following needs to be ensured: • Allow TCP port 22 ingress from SCM instance subnet/VCN, for SCM to SSH to DBCS node. • Allow TCP port 1521, 1522 from SCM instance subnet for database import and worker nodes for Siebel CRM to connect to the database.
siebel_cluster_subnet_ocid	infrastructure	(Required for using existing VCN resource) OCID of the subnet where the Kuberenetes API end point will be made available. The following needs to be ensured: • Allow all TCP port traffic from worker nodes private subnet, enables Kubernetes control plane to communicate with worker nodes. • Allow ingress for TCP port 6443 to the cloud manager instance subnet for accessing the Kubernetes cluster. • Allow ingress for TCP port 6443 and 12250 to the worker nodes for connecting with the API server. • Allow ICMP port 3 and 4 for instances to receive Path MTU Discovery fragmentation messages. • Allow egress rule for "All <region> Services in Oracle Services Network", enables accessing OCI container registry and other OCI services from worker nodes.
vcn_ocid_of_db_subnet	infrastructure	(Required for using existing VCN resource) OCID of the VCN which will be attached to the access control list of autonomous database (ATP). This is needed for establishing connection when the database is launched in a different VCN than the worker node subnet.

Payload Parameter	Section	Description
load_balancer_type	infrastructure	(Optional) Option to make load balancer as private/public Customer can restrict visibility of the Siebel application using this payload parameter. Supported values are one of: Private, Public. Choosing the "Public" option will assign a loadbalancer with public IP for public access. Choosing the "Private" option will create a loadbalancer with only private IP which can be accessed within the network only. If it is not specified, a public IP will be assigned.
load_balancer_ssl_cert_path	infrastructure	(Optional) Specifies the path of the ssl certificate file which contains public certificate or collection of public certificates that you can provide as an aggregated group for load balancer. The ssl certificate should be in PEM format only. If your ssl certificate submission returns an error, the most common reasons are: • Your ssl public certificate is malformed. • The system does not recognize the encryption method used for your certificate.
load_balancer_private_key_path	infrastructure	(Optional) Speficies the path of the private key file for the Load Balancer TLS/SSL certificate. The private key should be in PEM format only. If your private key submission returns an error, the most common reasons are: • You provided an incorrect password. • Your private key is malformed. • The system does not recognize the encryption method used for your key.
load_balancer_private_key_password	infrastructure	(Optional) The OCID of the secret containing the password of the Load Balancer private key. This will be used to decrpyt the private key provided in the 'load_balancer_private_key_path' parameter.
load_balancer_tls_secret_name	infrastructure	Specifies the name of the Load Balancer tls secret name to be given during environment provisioning. Note: If you provide ingress annotations, the value of tls-secret annotation should be same as the value of this parameter. The default value for load_balancer_tls_secret_name is "lb-tls-certificate". You can provide "lb-tls-certificate" for the value of tls-secret annotation under the ingress controller annotation section if this parameter is not configured in the payload.

Payload Parameter	Section	Description
shift_siebel_fs	infrastructure	(Optional) This parameter specifies whether shifting of the file system is to be executed or skipped while BYO-FS(infrastructure > mounttarget_exports) is used. Default value is set to True.
mounttarget_exports	infrastructure	(Required if the "Use existing resources" option is chosen during SCM stack creation) The mount_target_private_ip and export_path information to be used for Siebel file system.
siebel_storage_kms_key_ocid	infrastructure	Optional. Specifies the OCID of the OCI Vault KMS key used to encrypt Siebel environment file systems created by SCM. If you omit this value, Siebel file systems use Oracle-managed encryption. If you configure this parameter, you must also configure the **`siebel_storage_kms_key_dg_defined_tag`** parameter. Note: This parameter is not applicable for BYO Siebel FSS. For more information, see *Using Vault for Managing Secrets*.
siebel_storage_kms_key_dg_defined_tag	infrastructure	(Required only when the **`siebel_storage_kms_key_ocid`** parameter is configured.) Specifies the defined tag that SCM applies to Siebel file systems created by SCM so that the customer-created dynamic group can match the file system resources.
namespace	infrastructure > siebel_storage_kms_key_dg_defined_tag	(Required only when the **`siebel_storage_kms_key_ocid`** parameter is configured.) Specifies the defined tag namespace used by the dynamic group rule.
key	infrastructure > siebel_storage_kms_key_dg_defined_tag	(Required only when the **`siebel_storage_kms_key_ocid`** parameter is configured.) Specifies the defined tag key used by the dynamic group rule.
value	infrastructure > siebel_storage_kms_key_dg_defined_tag	(Required only when the **`siebel_storage_kms_key_ocid`** parameter is configured.) Specifies the defined tag value used by the dynamic group rule.
memory_in_gbs	infrastructure > kubernetes > oke > oke_node_shape_config	(Optional for Flex shape type) Specifies the amount of memory available to each node in the node pool, in gigabytes. This setting is editable only for flexible node shape options.
ocpus	infrastructure > kubernetes > oke > oke_node_shape_config	(Optional for Flex shape type) Specifies the number of Oracle CPUs (OCPUs) available to each node in the node pool. This setting is editable only for flexible node shape options.
oke_cluster_id Note: You can either pass oke_cluster_id and oke_endpoint or you can pass only oke_kubeconfig_path in payload	infrastructure > kubernetes > byo_oke	(Required when 'kubernetes_type' is BYO_OKE) The OCID of the OCI Kubernetes Cluster. Note: • The SCM instance should have access to the OKE cluster to perform any operation on cluster-related resources. The following policy statement enables <subject> to access and perform operations

Payload Parameter	Section	Description
		on cluster-family in compartment id <oke_compartment_ocid> - Allow <subject> to manage cluster-family in compartment id <oke_compartment_ocid> • VCN peering is required if OKE and SCM instance reside in different VCNs. For more information, refer to *Access to Other VCNs: Peering*. For more information, see *Using Vault for Managing Secrets*.
oke_endpoint Note: You can either pass oke_cluster_id and oke_endpoint or you can pass only oke_kubeconfig_path in payload	infrastructure > kubernetes > byo_oke	(Required when 'kubernetes_type' is BYO_OKE) Specifies the endpoint used to generate kubeconfig and access cluster. The available options are • PRIVATE • PUBLIC Depending on the input, either private or public endpoint will be used to access cluster.
oke_kubeconfig_path	infrastructure > kubernetes > byo_oke	(Required when 'kubernetes_type' is BYO_OKE) Specifies the path of kubeconfig file of an existing OKE to access and configure cluster. Copy the kubeconfig file to the SCM container at this location: '/home/opc/siebel' and provide the path for the file, such as '/home/opc/siebel/kubeconfig'. Note: • You can provide OKE information either by passing parameters (oke_cluster_id and oke_endpoint) or directly passing kubeconfig path using parameter oke_kubeconfig_path • SCM instance should be having access to OKE to perform any operation on cluster-related resources. The following policy statement enables <subject> to access and perform operation on cluster-family in comartment id <oke_compartment_ocid> - Allow <subject> to manage clusterfamily in compartment id <oke_compartment_ocid> • VCN peering is required if OKE and SCM instance reside in different VCNs. For more information, refer to *Access to Other VCNs: Peering*. For more information, see *Using Vault for Managing Secrets*.
kubeconfig_path	infrastructure > kubernetes > byo_ocne infrastructure > kubernetes > byo_other infrastructure > kubernetes > byo_openshift	Required when **`kubernetes_type`** is **`BYO_OCNE`**, **`BYO_OTHER`**, or **`BYO_OPENSHIFT`**. Specifies the path to the **`kubeconfig`** file for an existing Kubernetes cluster. Copy the **`kubeconfig`** file to **`/home/opc/siebel`** in the SCM container and provide the file path, such as **`/home/opc/siebel/kubeconfig`**. Note: SCM must have access to the Kubernetes cluster to perform operations on cluster-related resources.
byo_ns	infrastructure > kubernetes > byo_ocne infrastructure > kubernetes > byo_other	(Optional for non-OpenShift clusters) Specifies whether the namespace is user managed.

Payload Parameter	Section	Description
	infrastructure > kubernetes > byo_oke infrastructure > kubernetes > byo_openshift	When set to **true**, SCM does not create or delete the Kubernetes namespace during Siebel CRM provisioning and cleanup operations. You must ensure that the existing namespace: • Matches the **name** field in the Siebel CRM deployment payload. • Is empty, with no existing Flux configurations or deployments.
kms_key_id	infrastructure > kubernetes > oke	(Optional) Specifies the OCID of the OCI Vault KMS key used to encrypt Kubernetes secrets for a new SCM-created OKE cluster. If you omit this value, SCM preserves the default OKE encryption behavior. If you provide this value, you must also configure **kms_key_dg_defined_tag** for SCM-Created Infrastructure Resources. For more information, see *Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM*.
kms_key_dg_defined_tag	infrastructure > kubernetes > oke	(Required only when the **kms_key_id** parameter is configured.) Specifies the defined tag that SCM applies to the OKE cluster so that the customer-created dynamic group can match the cluster resource principal. For more information, see *Configuring Customer-Managed KMS Keys for Infrastructure Resources Created by SCM*.
namespace	infrastructure > kubernetes > oke > kms_key_dg_defined_tag	(Required only when the **kms_key_id** parameter is configured.) Specifies the defined tag namespace used by the dynamic group rule.
key	infrastructure > kubernetes > oke > kms_key_dg_defined_tag	(Required only when the **kms_key_id** parameter is configured.) Specifies the defined tag key used by the dynamic group rule.
value	infrastructure > kubernetes > oke > kms_key_dg_defined_tag	(Required only when the **kms_key_id** parameter is configured.) Specifies the defined tag value used by the dynamic group rule.
ingress_service_type	infrastructure > ingress_controller	Specifies the ingress service type to be provisioned during Siebel CRM deployment. Allowed values are LoadBalancer or NodePort.
ingress_controller_service_annotations	infrastructure > ingress_controller	(Optional) Specifies annotations that needs to be added to ingress service Note: When **ingress_service_type** is LoadBalancer and for **BYO OKE** or **BYO OCNE** use case '**service.beta.kubernetes.io/oci-load-balancer-subnet1**' annotation is required under sub-section '**ingress_controller_service_annotations**'
node_ports	infrastructure > ingress_controller	Optional. Specifies fixed Kubernetes NodePorts for externally exposed Traefik entry points. Configure this parameter only when **ingress_service_type** is set to **NodePort**. If you omit this parameter, Kubernetes assigns NodePorts automatically. For more information, see *Configuring Fixed NodePorts for Traefik*.
web	infrastructure > ingress_controller > node_ports	Optional. Specifies the fixed NodePort for the Traefik HTTP entry point. Enter an unused integer within the NodePort range configured for the Kubernetes cluster. Use a different value from **websecure**.

Payload Parameter	Section	Description
websecure	infrastructure > ingress_controller > node_ports	Optional. Specifies the fixed NodePort for the Traefik HTTPS entry point. Enter an unused integer within the NodePort range configured for the Kubernetes cluster. Use a different value from **web**.
siebfs_mt_export_paths	infrastructure > mounttarget_ exports	(Required if the "Use existing resources" option is chosen during SCM stack creation) The list of **mount_target_private_ip** and **export_path** information to be used for Siebel file system matching the number of **siebel_file_system_count** in source environment. The payload structure would be: **"infrastructure": { "mounttarget_exports": { "siebfs_mt_export_paths": [{"mount_target_private_ip" : ****,"export_path": "/exttest2-siebfs0"}, {"mount_target_private_ip" : **** ,"export_path": "/exttest2-siebfs1"}, {"mount_target_private_ip" : ****, "export_path": "/exttest2-siebfs1"}] }, (other infrastructure payload parameters) }**
migration_package_mt_export_ path	infrastructure > mounttarget_ exports	(Required if the "Use existing resources" option is chosen during SCM stack creation) The **mount_target_private_ip** and **export_path** information to be used for Migration storage. The payload structure would be: **{ "mounttarget_exports": { "migration_package_mt_export_path": {"mount_target_private_ip" : "****", "export_path": "/test-migration"} } }** **Note**: If this parameter is not provided for SCM created Siebel Deployment, SCM will create a dedicated export path for migration storage with path **/<env_namespace-migration**. This can be mounted in target environments.
db_type	database	Specifies one of the following: • ATP (for Oracle Autonomous Database) • DBCS_VM (for Oracle Database Cloud Service) • BYOD (stands for Bring Your Own Database – for the case when the "Use existing resources" option is chosen during Cloud Manager stack creation) For ATP, also include options under database > atp. For DBCS_VM, also include options under database > dbcs_vm. For BYOD, also include options under database > byod. For more information, see *Using a BYO Database*.
siebel_admin_username	database > auth_info	(Mandatory) The username of the Siebel administrative user.

Payload Parameter	Section	Description
siebel_admin_password	database > auth_info	(Mandatory) OCID of the secret containing the Siebel Administration User password. Password should not contain the username as a part of it. For more information on the password format criteria, refer to the "Characters Supported in Siebel Passwords" section in the Siebel Security Guide. For more information, see *Using Vault for Managing Secrets*.
table_owner_user	database > auth_info	(Mandatory) The Table owner in which the Siebel schema will be imported.
table_owner_password	database > auth_info	(Mandatory) OCID of the secret containing he login password used for the Siebel table owner. Password should have at least 2 Upper characters, 2 Lower characters, 2 Digits and 2 special characters from _,#,- of length 9 to 30 characters. Password should not contain the username as a part of it. For more information, see *Using Vault for Managing Secrets*.
default_user_password	database > auth_info	(Mandatory) OCID of the secret containing the default user password updated for all the users. For more information on the password format criteria, refer to the "Characters Supported in Siebel Passwords" section in the Siebel Security Guide. For more information, see *Using Vault for Managing Secrets*.
anonymous_user_password	database > auth_info	(Mandatory) OCID of the secret containing the anonymous user password which will be updated. For more information on the password format criteria, refer to the "Characters Supported in Siebel Passwords" section in the Siebel Security Guide. For more information, see *Using Vault for Managing Secrets*.
admin_password	database > atp	OCID of the secret for the password of the ATP database administrator user. Password should be have at least 12 to 30 characters, 1 upper character, 1 lower character and one number. Password cannot contain "" or the word "admin" in it. Review the password policy for shared ATP infrastructure in OCI and provide a valid password. For more information about the Oracle Autonomous Database, see *https://docs.oracle.com/en/cloud/paas/atp-cloud/index.html* on Oracle Help Center. For more information, see *Using Vault for Managing Secrets*.
wallet_password	database > atp	(OCID)(Required) OCID of the secret containing the password for ATP wallet download. Password can contain alphanumeric characters and of length 8 to 60. For more information, see *Using Vault for Managing Secrets*.

Payload Parameter	Section	Description
cpu_cores	database > atp	(Required) Specifies the ATP database's allocated Elastic CPUs (ECPUs). The minimum value is 2.
whitelist_cidrs	database > atp	Specifies the cidrs to be added to the ATP DB ACL list when cloudmanager creates database Cloudmanager creates Autonomous Database with the Secure access from allowed IPs and VCNs only option, you can restrict network access by defining Access Control Lists (ACLs). When using bring your own flow like BYO OCNE and if you want to include cidrs of bring your own components in ACL list of ATP DB to establish connection between them, you can utilize this parameter. Example: "whitelist_cidrs": "[129.0.0.0/8]"
storage_in_tbs	database > atp	(Required) Specifies the ATP database's disk storage, in terabytes. The minimum value is 1.
db_version	database > atp	(Optional) Specifies the ATP database version. Supported values are 23ai or 19c. The default is 19c.
wallet_path	database > byod	(Required for user provided database if the "Use existing resources" option is chosen during SCM stack creation) The absolute path of the Oracle net services configuration files or Oracle client credentials (wallet) is required for connecting to the database. The wallet files have to be copied inside the SCM container. The wallet should contain atleast the tnsnames.ora for a valid folder. During environment provisioning the wallet will be validated if it contains the tnsnames.ora. TLS enabled wallets are also supported. The provided wallet path will be copied inside the environment directory for usage. For more information, see *Using a BYO Database*.
tns_connection_name	database > byod	(Required for user provided database if the "Use existing resources" option is chosen during SCM stack creation) This is the connection identifier which will be used by the Siebel CRM application to establish connection to the database. The provided connection identifier will be validated if it's present in the tnsnames.ora. For more information, see *Using a BYO Database*.
drg_ocid	database > byod	(Optional) OCID of the DRG to be attached with the OKE nodes subnet to allow traffic from the VCN (where Database resides) provided that the both the DB VCN and CM VCN is peered. For more information, see *Using Vault for Managing Secrets*.
destination_db_cidr_block	database > byod	(Optional) Destination CIDR block where traffic has to be routed from OKE nodes subnet to the VCN (where Database resides) provided that the both the DB VCN and CM VCN is peered.
availability_domain	database > dbcs_vm	(Optional) The availability domain in which the database is to be used. Possible availability domains are 1, 2, and 3, depending on the region. Defaults to 1.

Payload Parameter	Section	Description
cpu_count	database > dbcs_vm	(Optional) The OCPU count for the DBCS database node. Possible values are from 4 to 64. Required memory is calculated on the formula of 16 GB times the number of OCPU cores. The current supported flex type relevant to this setting is VM.Standard.E4.Flex.
data_storage_size_in_gbs	database > dbcs_vm	(Required) The storage size of the database instance, in gigabytes. The different storage sizes are: 256, 512, 1024, 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672, 30720, 32768, 34816, 36864, 38912, or 40960.
database_edition	database > dbcs_vm	(Optional) The edition of Oracle Database to be used. Currently supported versions are: • DATABASE_EDITION_ENTERPRISE_EDITION (default) • DATABASE_EDITION_ENTERPRISE_EDITION_HIGH_PERFORMANCE
db_admin_username	database > dbcs_vm	(Required) Username for the Oracle schema user to be created with DBA privileges for administration activities. Username should have atleast 6 to 15 characters and only alphabets.
db_admin_password	database > dbcs_vm	(OCID)(Required) OCID of the secret for the password of the Oracle schema user. Password should have atleast 2 Upper characters, 2 Lower characters, 2 Digits and 2 special characters from _,#,- of length 9 to 30 characters. Password should not contain the username as a part of it.Password should not contain the username as a part of it. For more information, see *Using Vault for Managing Secrets*.
db_home_admin_password	database > dbcs_vm	(Required) OCID of the Vault secret that contains the password for the Oracle Database administrator account created for the DBCS DB home. SCM uses this password while provisioning the DBCS database system. The password must contain at least 2 uppercase letters, 2 lowercase letters, 2 digits, and 2 special characters from _, #, or -. Its length must be 9 to 30 characters. This is separate from db_admin_password, which is used for the SCM-created Oracle schema user.
mount_target_ip	database>dbcs_vm	(Required when infrastructure > mounttarget_exports is provided) IP address of the mount target used for creating the database directory in the DB node.
export_path	database>dbcs_vm	(Required when infrastructure > mounttarget_exports is provided) Export path in the mount target used for creating the database directory in the DB node. Note: This export path will be used for copying the database dumps and database directory for the import in database shifting stage.
db_version	database > dbcs_vm	(Mandatory) Specifies the DBCS database version. Supports versions from 19c to 23ai.
shape	database > dbcs_vm	(Required) The shape of the node for the Oracle Database instance. The different shapes in which the database can be provisioned can be found in the Limits, Quotas, and Usage section in the OCI console.

Payload Parameter	Section	Description
cpu	size > ses_resource_limits	(Optional) Specifies CPU resource limits of SES containers. This parameter specifies the max number of CPU units that can be allocated to the container. It can be given as a whole number like "1" or as a decimal number like "0.5" or in milliCPU units like "500m". The default is "2". Precision finer than "1m" is not allowed. For more information, refer to Kubernetes documentation. If not specified in payload, default value is used. ses_resource_limits must be greater than or equal to the value of ses_resource_requests parameter.
memory	size > ses_resource_limits	(Optional) Specifies memory resource limits of SES containers. This parameter specifies the max amount of memory that can be allocated to the container. It can be given in Ki,Mi,Gi and Ti units. The default is "4Gi". Specify in multiples of 2, such as 4, 8, 16, and so on. For more information, refer to Kubernetes documentation. If not specified in payload, default value is used. ses_resource_limits must be greater than or equal to the value of ses_resource_requests parameter.
cpu	size > ses_resource_requests	(Optional) Specifies the minimum guaranteed amount of CPU resources that is to be reserved for SES containers. It can be given as a whole number or with a decimal point like "0.5" or in milliCPU units like "500m". The default is "1". A request with a decimal point, such as "0.1", is converted to "100m" (100 milliCPU) by the API. Precision finer than "1m" is not allowed. For more information, refer to Kubernetes documentation. If not specified in payload, default value is used. ses_resource_limits must be greater than or equal to the value of ses_resource_requests parameter.
memory	size > ses_resource_requests	(Optional) Specifies the minimum guaranteed amount of memory resources that is to be reserved for SES containers. It can be given in Ki,Mi,Gi and Ti units. The default is "4Gi". Specify in multiples of 2, such as 4, 8, 16, and so on. For more information, refer to Kubernetes documentation. If not specified in payload, default value is used. ses_resource_limits must be greater than or equal to the value of ses_resource_requests parameter.
cpu	size > cgw_resource_limits	(Optional) Specifies CPU resource limits of Siebel Cloud Gateway containers. Default value is "2". If not specified in payload, default value is used. cgw_resource_limits must be greater than or equal to the value of cgw_resource_requests parameter.

Payload Parameter	Section	Description
memory	size > cgw_resource_limits	(Optional) Specifies memory resource limits of Siebel Cloud Gateway containers. Default value is "4Gi". If not specified in payload, default value is used. cgw_resource_limits must be greater than or equal to the value of cgw_resource_requests parameter.
cpu	size > cgw_resource_requests	(Optional) Specifies the minimum guaranteed amount of CPU resources that is to be reserved for Siebel Cloud Gateway containers. Default value is "1". If not specified in payload, default value is used. cgw_resource_limits must be greater than or equal to the value of cgw_resource_requests parameter.
memory	size > cgw_resource_requests	(Optional) Specifies the minimum guaranteed amount of memory resources that is to be reserved for Siebel Cloud Gateway containers Default value is "4Gi". If not specified in payload, default value is used. cgw_resource_limits must be greater than or equal to the value of cgw_resource_requests parameter.
cpu	size > sai_resource_limits	(Optional) Specifies CPU resource limits reserved for Siebel Application Interface containers (SAI). Default value is "2". If not specified in payload, default value is used. sai_resource_limits must be greater than or equal to the value of sai_resource_requests parameter.
memory	size > sai_resource_limits	(Optional) Specifies memory resource limits of Siebel Application Interface containers (SAI). Default value is "4Gi". If not specified in payload, default value is used. sai_resource_limits must be greater than or equal to the value of sai_resource_requests parameter.
cpu	size > sai_resource_requests	(Optional) Specifies the minimum guaranteed amount of CPU resources that is to be reserved for Siebel Application Interface containers (SAI). Default value is "1". If not specified in payload, default value is used. sai_resource_limits must be greater than or equal to the value of sai_resource_requests parameter.
memory	size > sai_resource_requests	(Optional) Specifies the minimum guaranteed amount of memory resources that is to be reserved for Siebel Application Interface containers (SAI). Default value is "4Gi". If not specified in payload, default value is used.

Payload Parameter	Section	Description
		sai_resource_limits must be greater than or equal to the value of sai_ resource_requests parameter.
siebel_monitoring	observability	(Optional) Set this value to true if you want to enable Siebel CRM Observability – Monitoring feature. Set this value to false to disable all of monitoring feature.
enable_oci_monitoring	observability	(Optional) Set this value to true to send metrics from Prometheus to the OCI monitoring service and create an OCI Application Performance Monitoring (APM) dashboard in OCI. Set this value to false to restrict sending the metrics from Prometheus to the OCI monitoring service and to restrict creating the OCI APM dashboard. Notes: The OCI infrastructure metrics for OCI resources will be available in OCI irrespective of the value of this parameter. siebel_monitoring should be 'true' and the oci_config parameter must be configured when enable_oci_monitoring is set to 'true'.
send_alerts	observability	(Optional) Set this value to true if you want to enable alerting feature in Siebel CRM Observability – Monitoring Set this value to false to disable alerting feature in Siebel CRM Observability – Monitoring. Note: siebel_monioring should be 'true' when send_alerts is set to 'true' in payload.
siebel_logging	observability	(Optional) Set this value to true if you want to enable Siebel CRM Observability – Log Analytics feature. Set this value to false to disable Siebel CRM Observability – Log Analytics feature.
enable_oci_log_analytics	observability	Set this value to true if you want to enable log streaming to OCI Logging Analytics. Set this value to false to disable log streaming to OCI Logging Analytics. Note: siebel_logging should be 'true' when enable_oci_log_analytics is set to 'true' in payload.
enable_oracle_opensearch	observability	Set this value to true if you want to create Oracle OpenSearch infrastructure and enable log streaming to Oracle OpenSearch. Set this value to false to disable log streaming to Oracle OpenSearch. Note: siebel_logging should be 'true' when enable_oracle_opensearch is set to 'true' in payload.

Payload Parameter	Section	Description
oci_log_analytics	observability	Required only for enabling OCI Logging Analytics for BYOR scenario, else optional. This section provides identifiers for various input parameters needed for enabling OCI Logging Analytics when BYOR ("Use existing resource") option is chosen during SCM installation.
smc_log_group_id	observability > oci_log_analytics	OCID of the log group in OCI Logging Analytics to send all SMC logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
sai_log_group_id	observability > oci_log_analytics	OCID of the log group in OCI Log Analytics to push all SAI related logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
ses_log_group_id	observability > oci_log_analytics	OCID of the log group in OCI Log Analytics to push all SES related logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
gateway_log_group_id	observability > oci_log_analytics	OCID of the log group in OCI Log Analytics to push all Gateway related logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
node_logs_log_group_id	observability > oci_log_analytics	OCID of the log group in OCI Log Analytics to push all Pod logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
log_source_name	observability > oci_log_analytics	Name of the log source in OCI Log Analytics for identifying the origin of logs. This is required only when enable_oci_log_analytics is set to 'true' in "Siebel CRM Observability – Monitoring and Log Analytics" solution and "Use existing resources" option is selected.
mount_target_private_ip	observability->monitoring_mt_export_path	Mount target private IP details required for monitoring component.
export_path	observability->monitoring_mt_export_path	Mount target export path details required for monitoring component.
storage_class_name	observability > prometheus observability > oracle_opensearch	(Optional In SCM Observability feature, Prometheus and Oracle OpenSearch use block volume. Block Volumes can be provisioned in one of the two following ways.

Payload Parameter	Section	Description
		• Dynamic provisioning involves automatic creation of storage volumes as needed by applications running in Kubernetes Cluster. Example: oci-bv • Static provisioning involves manual creation of storage volumes and making them available to applications by predefining them in Kubernetes cluster. For example: local-storage If your Kubernetes cluster doesn't have support for dynamic provisioning of block volumes, and you want to use local storage of a node for Prometheus or Oracle OpenSearch., you can provide local-storage as the storage_class_name. You can also provide your own custom integration storage type by passing the name of the storage class in this parameter. Default value for this field is 'oci-bv'.
local_storage	observability > prometheus > local_storage_info observability > oracle_opensearch > local_storage_info	If storage_class_name is local-storage, then this parameter specifies the local storage path.
kubernetes_node_hostname	observability > prometheus > local_storage_info observability > oracle_opensearch > local_storage_info	If storage_class_name is local-storage, then this parameter specifies the hostname in which the local storage path is present.
oci_config_path	observability->oci_config	Specifies the path to the oci config file. This is required only when either siebel_monitoring or enable_oci_log_analytics is enabled. Note: The region defined in the oci configuration file provided as oci_config_path parameter should be same as region where SCM is deployed.
oci_private_api_key_path	observability->oci_config	Specifies the path to the oci private key file. This is required only when either siebel_monitoring or enable_oci_log_analytics is enabled for Siebel CRM Observability – Monitoring and Log Analytics solution.
oci_config_profile_name	observability->oci_config	Specifies the profile name to be used in the oci config file. This is required only when either siebel_monitoring or enable_oci_log_analytics is enabled for Siebel CRM Observability – Monitoring and Log Analytics solution.
smtp_host	observability->alertmanager_email_config	Specifies the SMTP host name required for SMTP configuration. This is required only when send_alerts is set to 'true' in Siebel CRM Observability – Monitoring and Log Analytics solution.

Payload Parameter	Section	Description
smtp_from_email	observability->alertmanager_email_config	Specifies the SMTP from email address using which emaill will be sent required for SMTP configuration. This is required only when send_alerts is set to 'true' in Siebel CRM Observability – Monitoring and Log Analytics solution.
smtp_auth_username	observability->alertmanager_email_config	Specifies the SMTP auth username required for SMTP configuration. This is required only when send_alerts is set to 'true' in Siebel CRM Observability – Monitoring and Log Analytics solution.
smtp_auth_password	observability->alertmanager_email_config	(Required when **`send_alerts`** is set to **`true`**) Specifies the password used for SMTP authentication. If Vault is enabled, enter the OCID of the Vault secret that contains the password. Otherwise, enter the password in plain text.
to_email	observability->alertmanager_email_config	Specifies one or more email addresses to which alerts are sent. To send alerts to multiple recipients, provide a comma-separated list of email addresses. For example: "to_email": test1@oracle.com,test2@oracle.com Spaces before or after each email address are ignored. Do not include empty entries in the comma-separated list. For example: test1@oracle.com,,test2@oracle.com This is required only when send_alerts is set to 'true' in Siebel CRM Observability – Monitoring and Log Analytics solution.
enable_oracle_db_monitoring	observability	(Optional) This parameter is used to enable Oracle DB monitoring. Set the value of this parameter to: • 'true' to enable Oracle DB monitoring. • 'false' to disable Oracle DB monitoring. By default, the value is set to 'false'. Note: If Siebel monitoring is not enabled, make sure that the **`siebel_monitoring`** parameter is set to 'true' when **`enable_oracle_db_monitoring`** is set to 'true'.
db_metrics_exporter_username	observability > auth_info	(Required only when **`enable_oracle_db_monitoring`** is set to 'true' and **`db_type`** is set to 'BYOD') The username that Oracle DB Exporter uses to authenticate with the Oracle Database and collect database metrics. Note: • This user must have the privileges described in the *Prerequisites for Enabling Oracle Database Monitoring for BYOD* section. • Do not use any existing Siebel User.
db_metrics_exporter_password	observability > auth_info	(Required only when **`enable_oracle_db_monitoring`** is set to 'true')

Payload Parameter	Section	Description
		The OCID of the secret that contains the DB Exporter user password. If you are not using a vault, enter the password as plain text. For more information on the password format criteria, refer to the "Characters Supported in Siebel Passwords" section in the Siebel Security Guide. For more information, see *Using Vault for Managing Secrets*.
wallet_path	observability > auth_info	(Required only when **enable_oracle_db_monitoring** is set to 'true' and **db_type** is set to 'BYOD') The absolute path of the Oracle net services configuration files or Oracle client credentials (wallet) that is required for connecting to the database. You must copy the wallet files into the SCM container. The specified wallet path is then copied into the environment directory for use.
tns_connection_name	observability > auth_info	(Required only when **enable_oracle_db_monitoring** is set to 'true' and **db_type** is set to 'BYOD') The connection identifier that Oracle DB Exporter uses to connect to the database. The connection identifier is validated to ensure it is present in the **tnsnames.ora** file.
data_as_metrics_exporter_username	observability > auth_info	(Required only when **enable_oracle_db_monitoring** is set to 'true' and **db_type** is set to 'BYOD') The username which Oracle DB Exporter will use to authenticate with Oracle database and execute custom SQL queries and provide it as database metrics. Note: • This user must have the privileges described in the *Prerequisites for Enabling Oracle Database Monitoring for BYOD* section. • Do not use any existing Siebel User.
data_as_metrics_exporter_password	observability > auth_info	(Required only when **enable_oracle_db_monitoring** is set to 'true') The OCID of the secret that contains the password of the user specified in the **data_as_metrics_exporter_username** parameter. If you are not using a vault, enter the password as plain text. For more information on the password format criteria, refer to the "Characters Supported in Siebel Passwords" section in the Siebel Security Guide. For more information, see *Using Vault for Managing Secrets*.
security_context	observability	(Optional) Defines the custom security context for observability containers and pods. If you configure this section, you must configure the **run_as_user**, **run_as_group**, and **fs_group** parameters.
run_as_user	observability > security_context	(Required when **security_context** is specified) Specifies the user ID under which observability container processes run.

Payload Parameter	Section	Description
		Note: On OpenShift, this value must be within the dynamic range specified by the **`openshift.io/sa.scc.uid-range`** namespace annotation.
run_as_group	observability > security_context	(Required when **`security_context`** is specified.) Specifies the group ID under which observability container processes run.
fs_group	observability > security_context	(Required when **`security_context`** is specified.) Specifies the file system group ID for observability shared volume access. Note: On OpenShift, this value must be the first ID in the dynamic range specified by the **`openshift.io/sa.scc.supplemental-groups`** namespace annotation.
resource_tags	observability	(Optional) Specifies custom OCI tags to apply to supported observability-related OCI resources created by SCM, such as dashboards, log groups, and Prometheus block volumes. Use this section to define **`defined_tags`** and **`freeform_tags`**. If omitted or empty, SCM preserves existing tags or applies no custom tag overrides.
defined_tags	observability > resource_tags	(Optional) Specifies OCI defined tags to apply to supported observability resources. The specified tag namespaces and tag keys must already exist in the target OCI tenancy. For dashboard imports, these values override any tags defined in the dashboard template.
freeform_tags	observability > resource_tags	(Optional) Specifies OCI free-form tags to apply to supported observability resources as key-value pairs. Free-form tags do not require predefined OCI tag namespaces. For dashboard imports, these values override any tags defined in the dashboard template.
prometheus_auth	observability	(Required when the **`siebel_monitoring`** parameter is set to **`true`**.) Specifies the Basic Authentication configuration for accessing the Prometheus dashboard.
basic_auth	observability > prometheus_auth	(Required) Specifies the Basic Authentication configuration for the Prometheus dashboard.
users	observability > prometheus_auth > basic_auth	(Required) Specifies the list of users that can access the Prometheus dashboard. Specify at least one user. Each user name must be unique.
username	observability > prometheus_auth > basic_auth > users	(Required) Specifies the user name for accessing the Prometheus dashboard. The user name must contain 6 to 15 alphabetic characters. Each user name must be unique in the users list.
password	observability > prometheus_auth > basic_auth > users	(Required) Specifies the OCID of the secret that contains the password for the Prometheus dashboard user. If you are not using a vault, enter the password as plain text. The password must contain 9 to 30 characters, including at least two uppercase letters, two lowercase letters, two numbers, and two special characters: underscore (_), number sign (#), or hyphen (-). The password must not contain the user name.
opensearch_dashboard_auth	observability	(Required when the **`siebel_logging`** and **`enable_oracle_opensearch`** parameters are set to **`true`**.) Specifies the Basic Authentication configuration for accessing OpenSearch Dashboards.

Payload Parameter	Section	Description
basic_auth	observability > opensearch_dashboard_auth	(Required) Specifies the Basic Authentication configuration for OpenSearch Dashboards.
users	observability > opensearch_dashboard_auth > basic_auth	(Required) Specifies the list of users that can access OpenSearch Dashboards. Specify at least one user. Each user name must be unique.
username	observability > opensearch_dashboard_auth > basic_auth > users	(Required) Specifies the user name for accessing OpenSearch Dashboards. The user name must contain 6 to 15 alphabetic characters. Each user name must be unique in the users list.
password	observability > opensearch_dashboard_auth > basic_auth > users	(Required) Specifies the OCID of the secret that contains the password for the OpenSearch Dashboards user. If you are not using a vault, enter the password as plain text. The password must contain 9 to 30 characters, including at least two uppercase letters, two lowercase letters, two numbers, and two special characters: underscore (_), number sign (#), or hyphen (-). The password must not contain the user name.

10 Appendix

Applying Siebel POC Patches in Siebel CRM Environment Deployed using SCM

This topic describes the process to patch your Siebel CRM containers with the POC that is supplied to you by Oracle Support, to fix an issue specific to your use of a specific Siebel CRM application version. Each POC patch contains one or more files which will add to, or overwrite, files in your Siebel CRM installation.

> **Note:** For a specific version of Siebel CRM, check whether a POC is available and apply it if required. Review the release notes for that version to determine whether a KM article with version-specific instructions for applying the POC is available. If a KM article is available, follow the instructions in the KM article to apply the POC, if required, in addition to the instructions mentioned in this section.

This topic includes the following sections:

- *Prerequisites for Applying Siebel POC Patches*
- *Applying Siebel POC Patches*

Prerequisites for Applying Siebel POC Patches

You must have the following prerequisites before applying Siebel POC patches from an Oracle Linux VM:

- Oracle Linux VM 8 or later: You must have an Oracle Linux VM with sufficient free disk space to pull the Siebel CRM base image, create the patched base image, and build the custom Siebel image. The VM must have access to the Git repository that stores the SCM Helm charts and to the user's container registry used for the deployment.
- Podman: Podman is an open-source container management tool for Linux, Windows, and other platforms. In this procedure, Podman is used to pull the Siebel CRM base image, extract the `POCApply` scripts, create the patched base image, build the custom Siebel image, and push images to the container registry. For information about installing Podman, see the Podman installation documentation.
- Git: Git must be installed on the VM. Git is used to clone the SCM Helm chart repository, pull the latest changes, commit updates, and push the updated `siebel-artifacts.yaml` file.

You must perform the following tasks before applying Siebel POC patches from an Oracle Linux VM:

- Create a symbolic link for Docker: Run the following command to create a symbolic link for Docker:
  ```
  sudo ln -sf /usr/bin/podman /usr/local/bin/docker
  ```
- Clone the Helm chart repository on the VM:
  a. Retrieve the Git CA certificate path from the `siebel-common-config` ConfigMap in the Siebel environment as follows:
  ```
  kubectl -n <env_namespace> get cm siebel-common-config -o jsonpath='{.data.GIT_CA_CERT_PATH}'
  ```
  b. Copy the CA certificate to the host VM.
  c. Configure Git to use the copied CA certificate:
  ```
  git config --global http.sslCAInfo <path_to_ca_certificate_on_host>
  ```

d. Clone the Helm chart repository:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

```
git clone -b <git_scm_repo_branch> https://<git_user>:<git_accesstoken>@<git_helm_repo_url>
 <helmchart_repo>
```

Applying Siebel POC Patches

To apply Siebel POC patches in a SCM provisioned Siebel CRM environment, do the following:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Perform the following steps in the SCM container or pod:
   a. Connect to the SCM instance:
      - For SCM deployed on OCI, connect to the host, and access the container as follows:
        ```
        ssh opc@<SCM instance id>
        docker exec -it cloudmanager bash
        ```
        or
        ```
        sudo podman exec -it cloudmanager bash
        ```
      - For SCM deployed in a CNCF Kubernetes cluster, access the SCM pod as follows:
        ```
        kubectl -n <env_namespace> exec -it pod/scm-<replicaset-id>-<pod-id> -- bash
        ```
        In the above command, `<env_namespace>` is the Siebel CRM environment name where you want to apply the POC patches.
   b. Synchronize the local Helm chart repository:
      ```
      cd /home/opc/siebel/<env_id>/<helmchart_repo>
      git pull
      ```
   c. Make a note of the Siebel CRM base image in the `/home/opc/siebel/<ENV_ID>/<cloudmanager_flux_repo>/flux-crm/apps/base/siebel/siebel-artifacts.yaml` file. Use the value of `spec.values.image.siebel.base_image`.

      > **Note:** The image tag for the Siebel CRM base image uses the format `<YY>.<MM>-full`. For example, `26.6-full`.

   d. Retrieve the latest custom image tag:
      ```
      source /home/opc/siebel/<env_id>/k8sprofile
      ```

```
flux get image policy cm-siebel-image-policy -n metatest1 | tail -n 1 | awk '{print $NF}'
```

e. Increment the custom image tag retrieved in the previous step:

```
<YY>.<MM>.CUSTOM.X to <YY>.<MM>.CUSTOM.X+1
```

For example: 25.2.CUSTOM.1 to 25.2.CUSTOM.2

Use the incremented value as `<new_custom_image_tag>` in subsequent steps.

2. Perform the following steps on the Oracle Linux VM:

a. Go to the cloned Helm chart repository:

```
cd <helmchart_repo>
```

b. Download the POC application scripts from the Siebel CRM container image in your registry:

```
podman login <user_registry_url> -u <registry_user> -p "<registry_password>"
id=$(podman create <user_registry_url>/<env_namespace>/siebel:YY.MM-full)
echo $id
podman cp $id:/config/POCApply <helmchart_repo>
podman rm $id
```

In the above commands:

- `$id` contains the container ID returned by the `podman create` command.
- `<user_registry_url>` is the user's container registry URL.
- `<env_namespace>` is the Siebel CRM environment name where you want to apply the POC patches.

c. Verify that the `<helmchart_repo>/POCApply` directory contains the following files:

- `applyPOC`
- `Dockerfile`
- `processPatches`

d. Copy the POC patches to the `<helmchart_repo>/POCApply` directory. POC patches are provided through My Oracle Support. For example:

```
P12345678_1200_Linux-x86-64_1of2.zip
```

e. Apply the POC patches as follows:

```
cd <helmchart_repo>/POCApply
bash applyPOC -s <source_base_image> -t <target_base_image>
```

In this command:

- `<source_base_image>` is the source base image - `<user_registry_url>/<env_namespace>/siebel:YY.MM-full`.
- `<target_base_image>` is the target base image - `<user_registry_url>/<env_namespace>/siebel:YY.MM-full-patchesTo<month><yy>`.

f. Push the target base image to the registry:

```
podman push <user_registry_url>/<envname>/siebel:YY.MM-full-patchesTo<month><yy>
```

g. Build a new custom image by using the patched base image:

```
cd <helmchart_repo>

git pull

podman build \
 --build-arg BASE_IMAGE=<target_base_image> \
 -t <user_registry_url>/<envname>/siebel:<new_custom_image_tag> \
 -f dockerfile \
 siebel-artifacts/build \
```

```
--storage-opt ignore_chown_errors=true
```

h. Push the custom image to the registry configured in the global configuration:

```
podman push <user_registry_url>/<envname>/siebel:<new_custom_image_tag>
```

3. Perform the following steps in the SCM container or pod:

a. Synchronize the local Helm chart repository:

```
ssh opc@<SCM_instance_id>
docker exec -it cloudmanager bash
```

or

```
sudo podman exec -it cloudmanager bash
```

b. Go to the Helm chart repository directory and pull the latest changes from the remote Git repository:

```
cd /home/opc/siebel/<env_id>/<helmchart_repo>
git pull
```

c. Update the `base_image` parameter in the following file with the target base image that you pushed in Step 2.f:

```
/home/opc/siebel/<ENV_ID>/<cloudmanager_flux_repo>/flux-crm/apps/base/siebel/siebel-artifacts.yaml
```

d. Commit and push the changes to the Git repository:

```
git add .
git commit -m "Update base_image with POC patched Siebel image"
git pull
git push
```

e. Monitor the deployment through Flux CD and verify that the new image containing the POC patches and custom files is deployed successfully. Wait for the Siebel CRM environment to initialize and become operational:

- Verify that the environment is functioning correctly by accessing the appropriate application URLs.
- Verify that the POC patches have been deployed successfully by checking the timestamps of the associated libraries or executables:

```
source /home/opc/siebel/<env_id>/k8sprofile
kubectl -n <env_namespace> exec -it edge-0 -- ls -ltr /siebel/mde/siebsrvr/lib
```

Installing Siebel Monthly Updates in a Siebel CRM Environment Deployed by SCM

You can use these steps to install latest monthly updates in a Siebel CRM environment deployed by SCM. These steps do not include repository upgrade steps, which are optional and identical to those relevant for on-premises Siebel CRM deployments.

This topic includes the following sections:

- *Prerequisites*
- *Before You Begin*
- *Installing Monthly Updates in a Siebel CRM Environment*
- *Troubleshooting HelmRelease Failure*

Starting with version 26.6, SCM and Siebel CRM support custom non-root user ID and group ID for pods. This helps harden the security posture of deployed environments. Pods can run with user-selected non-root user ID, group ID, and file system group ID based on the security requirements of the target cluster.

Note: Using custom non-root user ID and group ID is optional. To use this feature, you must first update your environment from version 26.5 or earlier to version 26.6 or later. Perform the custom non-root user ID and group ID update steps only when you need to change the user ID and group ID or when the target cluster security policy requires specific non-root user, group, or file system group values.

To move to the hardened security model, you must:

1. Install the monthly update by following the procedure in this section.
2. Perform the post-update security hardening steps for SCM and Siebel pods. For more information, see the *Post-Update Security Hardening for SCM and Siebel Pods* section in Appendix.

Note: For OpenShift environments, you must:

1. Complete the prerequisites for updating to version 26.6 or later. For more information, see *Prerequisites for Updating to Version 26.6* .
2. Apply the monthly update.
3. (Optional) Perform the post-update security hardening steps. For more information, see the *Post-Update Security Hardening for SCM and Siebel Pods* section in Appendix.

Note: When moving Siebel environments from versions older than CM_23.8.1 to the latest, by following steps below, the sourcing of python virtual environment is required in addition to sourcing of the k8sprofile to access the OKE Cluster. Otherwise, for example, kubectl commands such as `kubectl get pods` may throw error. Sourcing of virtual environment and k8sprofile can be done by running the following commands:

```
sudo podman exec -it cloudmanager bash
bash-4.4$ source /home/opc/venv/bin/activate
source /home/opc/siebel/<env_id>/k8sprofile
```

If you're migrating from a Siebel CRM release earlier than 25.10, you'll need to apply OpenSSL-based encryption later in the monthly updates' installation process, after you build and push the new Siebel CRM custom image for the target version.

Prerequisites

Before you install Siebel monthly updates, ensure that the following prerequisites are met:

- **Oracle Linux 8 VM or later:** You must provision an Oracle Linux 8 (or later) VM with sufficient free disk space to mount the Siebel and SCM file system, pull the Siebel CRM base image, and build the custom Siebel CRM image. You must ensure that the VM has access to the Siebel CRM and SCM file system and to the container registry used for deployment.
- **Podman:** Install Podman, an open-source container management tool. This procedure uses Podman to pull the Siebel CRM base image, build the custom Siebel CRM image, and push images to the container registry.

  For installation instructions, see the Podman installation documentation.
- **NFS client package:** Install the NFS client package on the VM so that you can mount the SCM and Siebel CRM file systems.

Before You Begin

Before installing Siebel monthly updates, you must mount the SCM and Siebel CRM file systems on the Oracle Linux VM as follows:

```
mkdir -p /home/opc/mounts/scm
sudo mount -t nfs <SCM_FILESYSTEM_HOST>:<SCM_NFS_PATH> /home/opc/mounts/scm

mkdir -p /home/opc/mounts/<ENV_NAMESPACE>-siebfs0
sudo mount -t nfs <SIEBEL_FILESYSTEM_HOST>:/<SIEBEL_FILESYSTEM_PATH> /home/opc/mounts/<ENV_NAMESPACE>-
siebfs0
```

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

Installing Monthly Updates in a Siebel CRM Environment

To install a Siebel monthly update in a Siebel CRM environment deployed using SCM:

1. Back up the database.

   You must perform a backup of the database; preferably, a full backup.
2. Enable separate keystores for server and client authentication.

   Starting with Siebel release 26.3, Siebel mandates the use of separate certificates for server and client authentication to improve security and ensure compliance with modern Public Key Infrastructure (PKI) standards.

   For the steps to configure separate keystores for client and server authentication, refer to MOS article KB888322 - Configuring Separate Keystores for Server and Client Authentication.

   Note: Effective May 1, 2026, many Trusted Root Certification Authorities will no longer issue SSL certificates that include both the server authentication and client authentication Extended Key Usage (EKU).
3. Back up the SCM provided files for the Siebel CRM environment.

   You must back up the files in the current environment to ensure that you have a working version of the required set of files in case of any issues with the new update or if you want to roll back to the previous version.

   To create a backup, do the following:

   a. Create a backup directory on the Oracle Linux VM prepared as part of the prerequisites. Create the directory under the mounted SCM file system as follows:

      ```
      ssh -i <private_key> opc@<cm_instance>
      ```

```
mkdir /home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<backup_dir_name>
```

In the above example:

- For VM-based SCM deployments, <SCM_BASE_DIR> is the resource prefix in uppercase.
- For SCM deployments on a Kubernetes cluster, <SCM_BASE_DIR> is the SCM namespace in uppercase.

b. Copy the SCM Helm charts, SCM Git repositories, environment configurations (for example, CGW and SES), Siebel server (for example, quantum) and AI configurations (for example, quantum) to the backup directory (<backup_dir_name>):

```
cd /home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<backup_dir_name>

cp -R /home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<env_namespace>-helmcharts/
/home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<ENV_NAMESPACE>-cloud-manager/
/home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/environments/<ENV_ID>_environment.yaml

/home/opc/mounts/<ENV_NAMESPACE>-siebfs0/<ENV_NAMESPACE>/CGW/
/home/opc/mounts/<ENV_NAMESPACE>-siebfs0/<ENV_NAMESPACE>/SES/
/home/opc/mounts/<ENV_NAMESPACE>-siebfs0/<ENV_NAMESPACE>/SAI/

/home/opc/mounts/<ENV_NAMESPACE>-siebfs0/<ENV_NAMESPACE>/edge/
/home/opc/mounts/<ENV_NAMESPACE>-siebfs0/<ENV_NAMESPACE>/quantum/
/home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<backup_dir_name>/
```

Note: The variables in the example have the following values:

- <SCM_BASE_DIR>: The resource prefix in uppercase or the SCM namespace in uppercase based on your environment.
- <ENV_ID>: The six characters long environment ID.
- <env_namespace>: The name of the environment given in the payload.
- <ENV_NAMESPACE>: The name of the environment given in the payload, in uppercase.

4. Tag git repositories.
   a. Create a tag in the SCM Git repository as follows:

```
sudo podman exec -it cloudmanager bash
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/
git pull
git tag <tag_name>
git push origin --tags
```

In the example, <tag_name> is the source Siebel version. For example, 25.7.

   b. Create a tag in the Helm charts Git repository as follows:

```
sudo podman exec -it cloudmanager bash
cd /home/opc/siebel/<ENV_ID>/ <env_namespace>-helmcharts/
git pull
git tag <Tag_Name>
git push origin --tags
exit
```

In the example, <Tag_Name> is the source Siebel version. For example, 25.7.

5. Update SCM.

   You must update your SCM instance to match the target Siebel CRM version to which you wish to upgrade. For more information, see *Updating Siebel Cloud Manager with a New Container Image*.

   To update SCM using Helm in a Kubernetes cluster, see *Updating SCM using Helm*.

6. Update SCM container images to the user's container registry.
   a. Copy the SCM container images to the user's container registry using the SCM mirror API. For more information, see *Mirroring Siebel Base Container Images*.
   b. Verify the mirror API response by comparing it with the image details specified in the `/home/opc/siebel-cloud-manager/scripts/cmapp/yaml/siebel_images.yaml` file.

   Note:
   - You must execute this step only when installing monthly updates in a Siebel CRM environment deployed on OCI using SCM. For more information, see *About this Chapter*.
   - You don't need to run this step when installing monthly updates in a Siebel CRM environment deployed on a Kubernetes cluster using Siebel Installer, because Siebel Installer updates the container images in the user's container registry during the installation process. For more information, see *Deploying Siebel CRM on a Kubernetes Cluster using Siebel Installer*.

7. Update secrets with the user's container registry details.

   You must update the secret definition used to pull images from the user's container registry with the user's container registry credentials. To recreate the secret definition with the user's container registry credentials and update the secret named `ocirsecret` used in flux project, do the following:

   a. Delete existing secret definitions as follows:

   ```
   sudo podman exec -it cloudmanager bash
   source /home/opc/siebel/<ENV_ID>/k8sprofile
   kubectl delete secret -n <env_namespace> ocirsecret
   kubectl delete secret -n <env_namespace> customsecret
   ```

   b. Go to the `secrets` directory, create new `ocirsecret` and `customsecret` definitions with the user's container registry details and write the secrets to YAML files as follows:

   ```
   cd /home/opc/siebel/<env_id>/<Cloud manager repository name>/flux-crm/infrastructure/secrets

   kubectl --dry-run=client -n <env_namespace> create secret docker-registry ocirsecret \
   --docker-server=<registry_url> \
   --docker-username=<registry_username>\
   --docker-password=<registry_password> \
   --docker-email=siebel@oracle.com \
   -o yaml > ocir-siebeldev.yaml

   kubectl --dry-run=client -n <env_namespace> create secret docker-registry customsecret \
   --docker-server=<registry_url> \
   --docker-username=<registry_username>\
   --docker-password=<registry_password> \
   --docker-email=siebel@oracle.com \
   -o yaml > customsecret.yaml
   ```

   c. Commit the changes to the remote Git repository:

   ```
   git add .
   git commit -m "updated secrets with user's container registry details"
   git pull
   git push
   ```

d. Reconcile flux to rollout the new secret created to the SCM repository:

```
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace>-repo -n <env_namespace>
flux reconcile kustomization infrastructure -n <env_namespace>
```

In the example, `<env_namespace>` is the name of the flux repository.

e. Verify the `ocirsecret` and `customsecret` definitions. Get the secret and observe the age of the secret to confirm that the secret is updated:

```
kubectl -n <env_namepace> get secret ocirsecret
kubectl -n <env_namepace> get secret customsecret
```

f. Verify that the secret is working by pulling a basic image from the user's container registry using the new `ocirsecret`, as follows:

> **Note:** Since `ocirsecret` is updated in `<env_namespace>`, you must create the test image pod with the name `test_imagePull.yaml` only in `<env_namespace>`.

i. Go to the `siebel` directory:

```
cd /home/opc/siebel/
```

ii. Create a kubernetes YAML file with the name `test_imagePull.yaml` that uses the updated secret to pull the image from the user's container registry. Create the `test_imagePull.yaml` file:

```
vi test_ImagePull.yaml
```

Add the following details to the `test_ImagePull.yaml` file:

```
apiVersion: v1 kind: Pod metadata:
name: test-image-pod namespace: <env_namespace>
spec: containers:
- name: my-test-container

image: <user_registry_url>/<registry_prefix>/cm/base-builder:<tag id from mirror response>
command: ["sh", "-c", "echo 'Image pulled successfully!' && sleep 180"] imagePullSecrets:
- name: ocirsecret
```

In the example above, `<env_namespace>` is the secret's namespace.

iii. Apply `test_imagePull.yaml` manifest to verify the secret:

```
kubectl apply -f test_imagePull.yaml -n <env_namespace>
```

iv. Query the status of pod continuously until the status changes from `ContainerCreating` to `Running`:

```
kubectl get pod test-image-pod -n <env_namespace> -w
```

v. Verify if the image has been pulled successfully:

```
kubectl -n <env_namespace> logs po/test-image-pod
```

vi. Clean up the test pod:

```
kubectl delete -f test_imagePull.yaml -n <env_namespace>
```

8. Update the passphrases for the required PKI files.

> **Note:** This secret is a pre-upgrade compatibility requirement for environments created with SCM 26.7 or earlier that use SCM-generated PKI material. It does not rotate PKI certificates, keystores, truststores, or their passphrases.

You must update the PKI material passphrases before the Siebel environment update, as follows:

a. Create the `siebel-pki-passphrase` secret as follows:

i. Set the environment namespace and passphrase values:

```
set -euo pipefail
NAMESPACE="<ENV_NAMESPACE>"
PASSWORD="siebel"
WORK_DIR="$(mktemp -d)"
```

ii. Remove the temporary working directory when the script exits:

```
trap 'rm -rf "$WORK_DIR"' EXIT
```

iii. Source the profile for the Siebel CRM environment:

```
source /home/opc/siebel/<ENV_ID>/k8sprofile
```

For SCM versions earlier than 23.8.1, activate the virtual environment before you source the Kubernetes profile:

```
source /home/opc/venv/bin/activate
```

iv. Create temporary passphrase files:

```
printf %s "$PASSWORD" > "$WORK_DIR/keystore-password"
printf %s "$PASSWORD" > "$WORK_DIR/client-keystore-password"
printf %s "$PASSWORD" > "$WORK_DIR/truststore-password"
```

v. Create the `siebel-pki-passphrase` Kubernetes Secret:

```
kubectl -n "$NAMESPACE" create secret generic siebel-pki-passphrase \
   --from-file=keystore-password="$WORK_DIR/keystore-password" \
   --from-file=client-keystore-password="$WORK_DIR/client-keystore-password" \
   --from-file=truststore-password="$WORK_DIR/truststore-password" \
   --dry-run=client -o yaml | kubectl apply -f -
```

vi. Clear the passphrase variable:

```
unset PASSWORD
```

b. Verify that the secret and its associated keys were created as follows:

```
kubectl -n "$NAMESPACE" get secret siebel-pki-passphrase \
   -o jsonpath='{.data.keystore-password}{"\n"}{.data.client-keystore-password}{"\n"}
{.data.truststore-password}{"\n"}'
```

The command returns three non-empty values.

9. Update the Traefik Helm chart:

   You must update the Traefik Helm chart to the latest version. To update the Traefik Helm chart, do the following:

   a. Suspend Flux kustomization for Traefik as follows:
      i. Verify the deployment state of Traefik:
      ```
      helm list -n <env_namespace>
      ```
      The name of the Traefik in the command response is `<env_namespace>-traefik`.
      ii. Suspend the Flux Kustomizations and Helm release for Traefik before applying the Traefik Helm chart changes from the Helm chart repository:
      ```
      flux suspend kustomization traefik-controller traefik-resources -n <env_namepsace>
      flux suspend helmrelease traefik -n <env_namepsace>
      ```

   b. Update the Helm chart repository:
      i. Copy the `infrastructure` Helm chart from the SCM `charts` directory:
      ```
      cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
      rm -Rf traefik
      cp -R /home/opc/siebel-cloud-manager/charts/traefik /home/opc/siebel/<ENV_ID>/
      <env_namespace>-helmcharts/
      ```
      ii. Go to the `traefik` directory:
      ```
      cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/traefik
      ```
      iii. Open `Chart.yaml` and increment the chart version to roll out the Traefik Helm chart changes.
      iv. Commit the changes to the Git repository:
      ```
      git add .
      git commit -m "traefik latest helmchart changes"
      git push
      ```
      v. Verify that the working tree is clean:
      ```
      git status
      ```

   > **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

   c. Resume the Flux Kustomizations and Helm release for Traefik:
   ```
   cd /home/opc/siebel/<ENV_ID>/ source k8sprofile
   flux resume kustomization traefik-controller traefik-resources -n <env_namepsace>
   flux resume helmrelease traefik -n <env_namepsace>
   flux reconcile source git siebel-repo -n <env_namespace>
   flux reconcile source git <env_namespace>-repo -n <env_namespace>
   ```

   In the example above, `<env_namespace>-repo` is the name of the Flux repository and `<env_namespace>` is the repository name.

   > **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

   d. Verify that the Traefik updates have been applied successfully. Confirm that the `SUSPENDED` field is `False` and the `READY` field is `True` in the output of the following commands:
   ```
   flux get kustomization traefik-controller -n <env_namespace>
   flux get kustomization traefik-resources -n <env_namespace>
   ```

```
flux get helmrelease traefik -n <env_namespace>
```

Verify that the `APP VERSION` field shows the latest version and that the version number has increased from the previous version:

```
helm list -n <env_namespace>
```

10. Update Metacontroller Helm chart.

You must update the Metacontroller Helm chart to the latest version. To update the Metacontroller Helm chart, do the following:

a. Suspend flux kustomization for Metacontroller as follows:
   i. Verify the deployment state of Metacontroller:
   ```
   helm list -n <env_namespace>
   ```
   The name of the Metacontroller in the command response is <env_namespace>-metacontroller.
   ii. Suspend flux kustomization for Metacontroller to rollout the latest Metacontroller changes in the Helm repository:
   ```
   flux suspend kustomization metacontroller -n <env_namepsace>
   ```

b. Copy the `tls` directory to a backup directory:
```
cd metacontraoller
cp -r tls /home/opc/siebel/<env_id>/<backup_directory>/tls
```

c. Update the Helm chart repository:
   i. Copy the Metacontroller Helm chart from the SCM `charts` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
   rm -Rf metacontroller
   cp -R /home/opc/siebel-cloud-manager/charts/metacontroller /home/opc/siebel/<ENV_ID>/
   <env_namespace>-helmcharts/
   ```
   ii. Go to the `metacontroller` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/metacontroller
   ```
   iii. Open `Chart.yaml` and increment the chart version to roll out the Metacontroller changes.
   iv. Copy the `tls` directory from the backup location to the `<helm_repo>/metacontroller` directory:
   ```
   cd /home/opc/siebel/<env_id>/<backup_folder>/metacontroller
   cp -r tls /home/opc/siebel/<env_id>/<helm_repo>/metacontroller/tls
   ```
   v. Commit the changes to the Git repository:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/metacontroller
   git add .
   git commit -m "metacontroller latest helmchart changes"
   git push
   ```
   vi. Verify the status of the working tree to confirm that the status is "nothing to commit, working tree clean":
   ```
   git status
   ```

> **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

d. Resume flux kustomization for Metacontroller:
```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume kustomization metacontroller -n <env_namepsace>
flux reconcile source git siebel-repo -n <env_namespace>
```

```
flux reconcile source git <env_namespace> -n <env_namespace>
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

e. Verify and confirm the Metacontroller updates, ensure that the value of the field SUSPENDED is False and READY is True for the following commands:

```
flux get kustomization metacontroller -n <env_namespace>
flux get helmrelease metacontroller -n <env_namespace>
```

f. Ensure that APP VERSION points to the latest version and is incremented from the previous version:

```
helm list -n <env_namespace>
```

11. Update Siebel operator Helm chart.

You must update the Siebel operator Helm chart to the latest version. To update the Siebel operator Helm chart, do the following:

a. Suspend flux kustomization for Siebel operator as follows:

i. Verify the deployment state of the Siebel operator:

```
kubectl get pods -n <env_namespace>
```

The name of the Siebel operator in the command response is siebel-controller-xxx; here, xxx is the random number generated and assigned to the Siebel-operator.

ii. Suspend flux kustomization for siebel-operator to rollout latest Siebel-operator changes in SCM and Helm repositories:

```
flux suspend kustomization siebel-operator -n <env_namepsace>
```

b. Update the Helm charts repository:

i. Copy the Siebel operator Helm charts from the SCM `siebel-operator` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
rm -Rf siebel-operator
cp -R /home/opc/siebel-cloud-manager/operators/siebel-operator /home/opc/siebel/<ENV_ID>/
<env_namespace>-helmcharts/
```

ii. Update the following in the `/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/manifest/siebel-operator.yaml` file:

- Replace `{{ infrastructure.namespace }}` with the actual namespace value. For example:

```
kind: CompositeController
 metadata:
 name: siebel-controller
 spec:
 generateSelector: true
 parentResource:
 apiVersion: siebel.metacontroller.io/v1alpha1
 resource: monitors
 childResources:
 - apiVersion: v1
 resource: configmaps
 updateStrategy:
 method: InPlace
 hooks:
 sync:
 webhook:
 url: https://siebel-controller.test.svc.cluster.local/sync
 timeout: 1800s
```

```
customize:
webhook:
url: https://siebel-controller.test.svc.cluster.local/customize
timeout: 1800s
```

Note: In this example, `test` is the namespace in which Siebel CRM is deployed.

- Update the `securityContext` section as follows:

```
securityContext:
 runAsUser: 1000
 runAsGroup: 0
 fsGroup: 1000
```

iii. Confirm that the `oracle.siebel.namespace-envid` parameter is set to `<env_namespace>-<env-id>` in the following files:

- `/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/kustomization.yaml`
- `/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/manifest/kustomization.yaml`

iv. Go to the `siebel-operator` Helm chart directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/manifest
```

v. Open `kustomization.yaml` and add the image details under the values section as follows.

```
images:
- name: siebel-operator-base
 newName: <user_registry_url>/<registry_prefix>/cm/siebel-operator-base
```

vi. Commit the changes to the Git repository:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/
git add .
git commit -m "Siebel operator latest helmchart changes"
git push
```

vii. Verify that the working tree is clean:

```
git status
```

Note: Flux reconciliation does not happen at this time as flux is in suspended mode.

c. Resume flux kustomization for Siebel operator:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume kustomization siebel-operator -n <env_namepsace>
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
```

In the example, `<env_namespace>` is the name of the flux repository.

Note: If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

d. Verify and confirm the Siebel operator updates, ensure that the value of the field SUSPENDED is False and READY is True:

```
flux get kustomization siebel-operator -n <env_namespace>
```

12. Suspend image update automation in Flux:

```
flux suspend image update cm-siebel-image-update1 -n <env_namespace>
flux suspend image update cm-siebel-image-update2 -n <env_namespace>
```

13. Build and push the new Siebel CRM custom image for the target version.

You must re-tag the copied Siebel CRM base image to the user's environment registry and push it to the registry specific to the user's environment. Perform the following steps on the Oracle Linux VM that you prepared as part of the prerequisites:

a. Re-tag the copied Siebel CRM base image to the user's environment registry:
   i. Set the target Siebel CRM version:
   ```
   export target_version=<target_siebel_version>
   ```
   ii. Set the `source_base_image` variable to the destination registry repository obtained from the mirror:
   ```
   export source_base_image="<user_registry_url>/<user_registry_prefix>/cm/siebel:
   $target_version-full"
   ```
   iii. Verify that the `source_base_image` variable is set properly:
   ```
   echo $source_base_image
   export target_base_image="<user_registry_url>/<user_registry_namespace>/<env_namespace>/
   siebel:$target_version-full"
   ```
   iv. Verify that the `target_base_image` variable is set properly:
   ```
   echo $target_base_image
   ```
   v. Re-tag the target Siebel CRM base image to the user's environment registry:
   ```
   podman login <user_region>.ocir.io
   podman tag $source_base_image $target_base_image
   ```
b. Log in to the docker registry to push the target Siebel CRM base image to the user's registry:
   ```
   podman login <user_region>.ocir.io
   podman push $target_base_image
   ```
c. Synchronize the Helm charts Git repository in the local environment with the remote repository to retrieve the latest custom artifact changes:
   ```
   sudo podman exec -it cloudmanager bash
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
   git clean -d -x -f
   git pull
   ```
d. Copy the `siebel-artifacts` Helm charts from SCM charts directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
   rm -Rf siebel-artifacts/
   cp -R /home/opc/siebel-cloud-manager/charts/crm/siebel-artifacts /home/opc/siebel/<ENV_ID>/
   <env_namespace>-helmcharts/
   ```
e. Restore the `siebel-artifacts/build` folder from the backup:
   ```
   cp -R /home/opc/siebel/<ENV_ID>/<backup_dir_name>/<env_namespace>-helmcharts/siebel-artifacts/
   build/ /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts/
   ```
f. Go to the `siebel-artifacts` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts
   ```
g. Open `Charts.yaml` and increment the chart version to roll out the Siebel artifact changes.
h. Commit the changes to the Git repository:
   ```
   git add .
   git commit -m "siebel-artifact latest helmchart changes"
   git push
   ```
i. Verify that the working tree is clean:
   ```
   git status
   ```
j. Replace `chown` with `chmod` in `dockerfile`:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts/build/
   ```

```
sed -i 's|COPY --chown=1000:1000 mde/ /siebel/mde|COPY --chmod=775 mde/ /siebel/mde|' dockerfile
```

k. Build a new custom image for the Siebel CRM web artifacts on the Oracle Linux VM that you prepared as part of the prerequisites, and then push the image to the customer registry:

```
cd /home/opc/mounts/scm/<SCM_BASE_DIR>/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts/build/
export target_image=<registry_url>/<registry_namespace>/<env_namespace>/siebel:$target_version.CUSTOM.1
podman build --build-arg BASE_IMAGE=${target_base_image} -t ${target_image} ./ -f dockerfile
podman push $target_image
exit
```

14. (Execute this step only if you are migrating from a Siebel CRM release earlier than 25.10). Apply OpenSSL-based encryption. For more information, see *Migrating Siebel CRM Containers to OpenSSL Encryption*.
15. Update Siebel Gateway Helm chart and Helm release.

    You must update the Siebel Gateway Helm chart to the latest version and Helm release to point to the user's container registry. To update the Siebel Gateway Helm chart and Helm release:

    a. Suspend flux Helm release for Siebel Gateway as follows:
        i. Verify the deployment state of Siebel Gateway:

```
sudo podman exec -it cloudmanager bash
source /home/opc/siebel/<ENV_ID>/k8sprofile
helm list -n <env_namespace>
```

        ii. Suspend flux Helm release for Siebel Gateway to roll out the latest changes:

```
flux suspend helmrelease siebel-gateway -n <env_namespace>
```

    b. Update Helm charts repository:
        i. Copy the latest Siebel Gateway Helm chart from SCM the charts directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
rm -Rf siebel-gateway
cp -R /home/opc/siebel-cloud-manager/charts/crm/siebel-gateway /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
```

        ii. Go to the `siebel-gateway` Helm chart directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-gateway
```

        iii. Open `Charts.yaml` and increment the chart version to roll out the Siebel Gateway changes.
        iv. Commit the changes to the Git repository:

```
git add .
git commit -m "siebel-gateway latest helmchart changes"
git push
```

v. Verify that the working tree is clean:

```
git status
```

> **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

c. Update the SCM repository with the Siebel Gateway changes:

i. Go to the `siebel` directory in the SCM repository:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/
```

> **Note:** You must verify that the image tags defined in each Helm release YAML file under the `<env_namespace>-cloud-manager` repository match the image tag IDs listed in the `/home/opc/siebel-cloud-manager/scripts/cmapp/yaml/siebel_images.yaml` file. Apply this verification to all Helm release YAML updates in each of the subsequent steps.

ii. Add the custom image details under the values section in the `siebel-gateway.yaml` file as follows:

```
values:
 image:
 siebel:
 imagePullPolicy: IfNotPresent
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/siebel
 tag: <target_image built in step 8.d, for example 25.5.CUSTOM.1>
busybox:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
logging:
 image:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/fluentd-collector
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
```

iii. Commit the changes to the repository:

```
git add .
git commit -m "Updating siebel-gateway helmrelease container images repository to user's
 container registry"
git push
```

d. Resume flux Helm release for Siebel Gateway:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume helmrelease siebel-gateway -n <env_namepsace>
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

e. Verify and confirm the Siebel Gateway updates, ensure that the value of the field SUSPENDED is False and READY is True:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease -n <env_namespace>
```

f. Ensure that APP VERSION points to the latest version and is incremented from previous version:
```
helm list -n <env_namespace>
```

16. Update Siebel Config Helm chart and Helm release.

You must update the `siebel-config` Helm chart to the latest version. To update the `siebel-config` Helm chart:

a. Suspend flux Helm release for `siebel-config` as follows:
   i. Verify the deployment state of `siebel-config`:
   ```
   helm list -n <env_namespace>
   ```
   ii. Suspend Helm release to rollout `siebel-config` changes in SCM and Helm repositories:
   ```
   flux suspend helmrelease siebel-config -n <env_namespace>
   ```

b. Update the Helm chart repository:
   i. Copy the `siebel-config` Helm chart from the SCM `charts` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-config
   rm -Rf templates/ values.yaml Chart.yaml
   cp -R /home/opc/siebel-cloud-manager/charts/crm/siebel-config/templates/ /home/opc/siebel-cloud-manager/charts/crm/siebel-config/values.yaml /home/opc/siebel-cloud-manager/charts/crm/siebel-config/Chart.yaml /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-config/
   ```
   ii. Go to the Helm chart `siebel-config` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-config
   ```
   iii. Open `Chart.yaml` and increment the chart version to roll out the infrastructure changes.
   iv. Commit the changes to the Git repository:
   ```
   git add .
   git commit -m "siebel-configlatest helmchart changes"
   git push
   ```
   v. Verify that the working tree is clean:
   ```
   git status
   ```

   > **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

c. Update the Helm release:
   i. Go to the `siebel` directory:
   ```
   cd /home/opc/siebel/<ENV_ID>/<Cloud manager repository name>/flux-crm/apps/base/siebel/
   ```
   ii. Update the value of the `busybox` parameter in `siebel-config.yaml` in SCM Git repository as follows:
   ```
   values:
   image:
   busybox:
    registry: <user_registry_url>
    repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
    tag: <tag_id from siebel_images.yaml>
    imagePullPolicy: Always
   dbutils:
    registry: <user_registry_url>
    repository: <registry_prefix/object_namespace>/<env_namespace>/cm/dbutils
    tag: <tag_id from siebel_images.yaml>
    imagePullPolicy: Always
   ```
   iii. Commit the changes to the Git repository:
   ```
   git add .
   git commit -m "Updating the image tags"
   git push
   ```

d. Resume flux Helm release for `siebel-config`:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume helmrelease siebel-config -n <env_namepsace>
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
flux reconcile -n <env_namepsace> kustomization apps
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

e. Verify and confirm the siebel-config updates, ensure that the value of the field SUSPENDED is False and READY is True for the following commands:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease -n <env_namespace>
```

f. Ensure that APP VERSION points to the latest version and is incremented from previous version:

```
helm list -n <env_namespace>
```

17. Update Siebel Helm chart and Helm release.

You must update the `siebel` Helm chart to the latest version and Helm release to point to the user's container registry. To update the `siebel` Helm chart and Helm release, do the following:

a. Suspend flux Helm release for `siebel` as follows:

i. Verify the deployment state of the `siebel`:

```
helm list -n <env_namespace>
```

ii. Suspend Helm release for `siebel` to roll out the latest changes:

```
flux suspend helmrelease siebel -n <env_namespace>
```

b. Update the Helm chart repository:

i. Copy the siebel Helm charts from SCM `charts` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
rm -Rf siebel
cp -R /home/opc/siebel-cloud-manager/charts/crm/siebel /home/opc/siebel/<ENV_ID>/
<env_namespace>-helmcharts/
```

ii. Go to the siebel Helm `chart` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel
```

iii. Open `Chart.yaml` and increment the chart version to roll out the Siebel changes.

iv. Commit the changes to the Git repository:

```
git add .
git commit -m "siebel latest helmchart changes"
git push
```

v. Verify that the working tree is clean:

```
git status
```

> **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

c. Update the SCM repository with the `siebel` changes

i. Go to the `siebel` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/
```

ii. Update the `siebel.yaml` file as follows:

```
values:
 image:
 siebel:
 imagePullPolicy: IfNotPresent
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/siebel
 tag: <target_image built in step 8.d, for example 25.5.CUSTOM.1>
 busybox:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
 dbutils:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/dbutils
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
logging:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/fluentd-collector
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
```

iii. Commit the changes to the Git repository:

```
git add .
git commit -m "Updating siebel helmrelease container images repo to my registry"
git push
```

d. Resume flux Helm release for `siebel`:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume helmrelease siebel -n <env_namepsace>
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
flux reconcile -n <env_namepsace> kustomization apps
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

e. Verify and confirm the Siebel updates, ensure that the value of the field SUSPENDED is False and READY is True for the following commands:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease siebel -n <env_namespace>
```

f. Ensure that APP VERSION points to the latest version and is incremented from previous version:

```
helm list -n <env_namespace>
```

18. Update Siebel artifacts Helm chart.

    You must update the Siebel artifacts Helm chart to the latest version and Helm release to point to the user's container registry. To update the Siebel artifacts Helm chart and Helm release:

    a. Suspend flux Helm release for `siebel-artifacts` as follows:
        i. Verify the deployment state of the `siebel-artifacts`:
        ```
        helm list -n <env_namespace>
        ```
        ii. Suspend Helm release for `siebel-artifacts` to roll out the latest changes:
        ```
        flux suspend helmrelease siebel-artifacts -n <env_namespace>
        ```
    b. Update the Helm chart repository:
        i. Copy the `siebel-artifacts` Helm charts from SCM charts directory:
        ```
        cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
        rm -Rf siebel-artifacts/
        cp -R /home/opc/siebel-cloud-manager/charts/crm/siebel-artifacts /home/opc/siebel/<ENV_ID>/
        <env_namespace>-helmcharts/
        ```
        ii. Replace `chown` with `chmod` in `dockerfile`:
        ```
        cd /home/opc/siebel/<ENV_ID>/<env_namespace>- helmcharts/siebel-artifacts/build/
        sed -i 's|COPY --chown=1000:1000 mde/ /siebel/mde|COPY --chmod=775 mde/ /siebel/mde|'
         dockerfile
        ```
        iii. Restore the `siebel-artifacts/build` folder from the backup:
        ```
        cp -R /home/opc/siebel/<ENV_ID>/backup/<env_namespace>-helmcharts/siebel-artifacts/build/ /
        home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts/
        ```
        iv. Go to the `siebel-artifacts` directory:
        ```
        cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts
        ```
        v. Open `chart.yaml` and increment the chart version to roll out the `siebel-artifacts` changes.
        vi. Commit the changes to the Git repository:
        ```
        git add .
        git commit -m "Siebel-artifacts latest helmchart changes"
        git push
        ```
        vii. Verify that the working tree is clean:
        ```
        git status
        ```

        > **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

    c. Update the SCM repository with the `siebel-artifacts` changes:
        i. Go to the `siebel` directory:
        ```
        cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel
        ```
        ii. Update the image details in the `siebel-artifacts.yaml` file as follows:
        ```
        values:
           image:
              siebel:
                    base_image: <user_registry_url>/<registry_prefix/object_namespace>/cm/siebel:
        $target_version-full
                    imagePullPolicy: IfNotPresent
                    registry: <user_registry_url>
                    repository: <registry_prefix/object_namespace>/<env_namespace>/siebel
                    tag: <target_image built in step 8.d, for example 25.5.CUSTOM.1>
              busybox:
                    registry: <user_registry_url>
                    repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
        ```

```
        tag: <tag_id from siebel_images.yaml>
        imagePullPolicy: Always
    dbutils:
        registry: <user_registry_url>
        repository: <registry_prefix/object_namespace>/<env_namespace>/cm/dbutils
        tag: <tag_id from siebel_images.yaml>
        imagePullPolicy: Always
```

iii. Set `imageBuilder.privileged` based on the target cluster type:

```
spec:
values:
imageBuilder:
privileged: <false_for_OCNE_and_OpenShift_true_for_other_cluster_types>
```

In the above payload section, set the value of `imageBuilder.privileged` to:

- `false` for OCNE and OpenShift clusters.
- `true` for other supported Kubernetes cluster types that require privileged image-builder execution.

iv. Commit the changes to the Git repository:

```
git add .
git commit -m "Updating siebel-artifacts helmrelease container images repo to user's
 container registry"
git push
```

d. Update version in `Chart.yaml` as follows:

i. Go to the `siebel-artifacts` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-artifacts
```

ii. Increment the version in the `Chart.yaml` file.

iii. Commit the changes to the Git repository:

```
git add .
git commit -m "Updating new siebel version"
git push
```

e. Resume flux Helm release for `siebel-artifacts`:

i. Resume `siebel-artifacts` helmrelease:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux resume helmrelease siebel-artifacts -n <env_namepsace>
```

ii. Trigger manual reconciliation:

```
flux resume image update cm-siebel-image-update1 -n <env_namespace>
flux resume image update cm-siebel-image-update2 -n <env_namespace>
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
flux reconcile kustomization apps -n <env_namepsace>
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

f. Verify and confirm the siebel-artifacts updates, ensure that the value of the field SUSPENDED is False and READY is True for the following commands:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease siebel-artifacts -n <env_namespace>
```

g. Ensure that APP VERSION points to the latest version:

```
helm list -n <env_namespace>
```

19. Update Siebel observability Helm chart and Helm release.

You must update the Siebel observability Helm chart to the latest version. To update the Siebel-observability Helm chart, do the following:

a. Verify the deployment state of monitoring Helm charts (`kube-state-metrics`, `node-exporter`, `prometheus`, `prometheus-adapter`, `prometheus-alertmanager`, and `oracle-db-monitoring`):

```
helm list -n <env_namespace>
```

> **Note:**
> - `prometheus-alertmanager` is deployed only if `send_alerts` is set to `true`.
> - `oracle-db-monitoring` is deployed only if `enable_oracle_db_monitoring` is set to `true`.

b. Suspend flux Helm release for Siebel observability to roll out the latest changes as follows:

```
flux suspend helmrelease kube-state-metrics -n <env_namespace>
flux suspend helmrelease node-exporter -n <env_namespace>
flux suspend helmrelease prometheus -n <env_namespace>
flux suspend helmrelease prometheus-adapter -n <env_namespace>
flux suspend helmrelease prometheus-alertmanager -n <env_namespace>
flux suspend helmrelease oracle-db-monitoring -n <env_namespace>
```

c. Update the Helm chart repository:

> **Note:** If you have made custom ConfigMap changes for any observability-related chart, back up the changes before replacing the Helm chart content. After you copy the updated charts, restore the custom changes as required.

i. Copy the Siebel observability Helm charts from the SCM `charts` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
rm -Rf node-exporter/ kube-state-metrics/prometheus prometheus-adapter/prometheus-alert-manager/oracle-db-monitoring/
cp -R /home/opc/siebel-cloud-manager/charts/siebel_observability/* /home/opc/siebel/<ENV_ID>/<env_namespace>
```

ii. Increment the chart version in the `Chart.yaml` file to roll out the changes, in the following directories:

```
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/kube-state-metrics/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/node-exporter/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/prometheus/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/prometheus-adapter/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/prometheus-alert-manager/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/oracle-db-monitoring/
```

iii. Commit the changes to the Git repository:

```
git add .
git commit -m " infrastructure latest helmchart changes"
git push
```

iv. Verify that the working tree is clean:

```
git status
```

> **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

d. Update the Helm release:

i. Go to the `siebel_observability` directory:

```
cd <git_url>/root/<Cloud manager repository name>/-/blob/master/ flux-crm/apps/base/
siebel_observability/
```

ii. Update the following in `prometheus.yaml` in SCM Git repository

- The `busybox` section as follows:

```
values:
 image:
 busybox:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
```

- The `prometheus_oci_adapter` section as follows:

```
values:
 prometheus_oci_adapter:
 image:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/prometheus-oci-
adapter
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
```

- The `siebel_metric_exporter` section as follows:

```
values:
 siebel_metric_exporter:
 image:
 imagePullPolicy: Always
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/siebel
 tag: <target_version>-full # Example: tag: 26.6-full
```

iii. Commit the changes to the Git repository:

```
git add .
git commit -m "Updating the image tags"
git push
```

e. Update the ownership of the Prometheus local storage directory on the worker node as follows:

```
chown -R 1000:1000 <prometheus_local_storage_path>
```

In the above command, `<prometheus_local_storage_path>` is the local storage path used by the Prometheus persistent volume in the target environment.

f. Resume flux Helm release for Siebel observability:

```
flux resume helmrelease kube-state-metrics -n <env_namespace>
flux resume helmrelease node-exporter -n <env_namespace>
flux resume helmrelease prometheus -n <env_namespace>
flux resume helmrelease prometheus-adapter -n <env_namespace>
flux resume helmrelease prometheus-alertmanager -n <env_namespace> #this can be given only if
 send_alerts = true in payload.
flux resume helmrelease oracle-db-monitoring -n <env_namespace> # Run this command only if
 enable_oracle_db_monitoring is set to true in the payload.
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace> -n <env_namespace>
```

```
flux reconcile -n <env_namepsace> kustomization apps
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

g. Verify and confirm the Siebel observability updates, ensure that the value of the field SUSPENDED is False and READY is True for `kube-state-metrics`, `node-exporter`, `prometheus`, `prometheus-adapter`, `prometheus-alertmanager`, and `oracle-db-monitoring`:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease -n <env_namespace>
```

h. Ensure that APP VERSION points to the latest version and is incremented from previous version for `kube-state-metrics`, `node-exporter`, `prometheus`, `prometheus-adapter`, `prometheus-alertmanager`, and `oracle-db-monitoring`:

```
helm list -n <env_namespace>
```

20. Update Siebel logging Helm chart and Helm release.

You must update Siebel logging Helm chart to the latest version. To update the Siebel logging Helm chart:

a. Suspend flux Helm release for Siebel logging as follows:

i. Verify the deployment state of Siebel logging Helm charts (`oracle-opensearch`, `oracle-opensearch-dashboards`, and `siebel-logging`):

```
helm list -n <env_namespace>
```

ii. Suspend flux Helm release to rollout Siebel logging changes in Helm repositories:

```
flux suspend helmrelease oracle-opensearch -n <env_namespace>
flux suspend helmrelease oracle-opensearch-dashboards -n <env_namespace>
flux suspend helmrelease siebel-logging -n <env_namespace>
```

b. Update the Helm chart repository:

> **Note:** If you have made custom ConfigMap changes for any logging-related chart, back up the changes before replacing the Helm chart content. After you copy the updated charts, restore the custom changes as required.

i. Copy the Siebel logging Helm charts from the SCM `charts` directory:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
rm -Rf siebel-logging/ oracle-opensearch/ oracle-opensearch-dashboards/
cp -R /home/opc/siebel-cloud-manager/charts/siebel-logging/ /home/opc/siebel-cloud-manager/charts/oracle-opensearch/ /home/opc/siebel-cloud-manager/charts/oracle-opensearch-dashboards/ /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
```

ii. Increment the chart version, to roll out the changes, in the `Chart.yaml` file in the following directories:

```
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-logging/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/oracle-opensearch/
/home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/oracle-opensearch-dashboards/
```

iii. Commit the changes to the Git repository:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/
git add .
git commit -m " siebel-logging latest helmchart changes"
git push
```

iv. Verify that the working tree is clean:

```
git status
```

> **Note:** Flux reconciliation does not happen at this time as flux is in suspended mode.

c. Update the Helm release:

i. Go to the `siebel-logging` directory:

```
cd <git_url>/root/<Cloud manager repository name>/-/blob/master/ flux-crm/apps/base/siebel-logging/
```

ii. Update the value of the `busybox` parameter in `oracle-opensearch.yaml` in SCM Git repository as follows:

```
values:
 image:
 busybox:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/base-builder
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
 opensearch:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/oracle-opensearch
 tag: <tag_id from siebel_images.yaml>
 imagePullPolicy: Always
```

iii. Update the `logging` and `logCollectorImage` sections in `siebel-logging.yaml` in SCM Git repository as follows:

```
values:
 logging:
 image:
 imagePullPolicy: Always
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/log-aggregator
 tag: <tag_id from siebel_images.yaml>
 logCollectorImage:
 registry: <user_registry_url>
 repository: <registry_prefix/object_namespace>/<env_namespace>/cm/fluentd-collector
 tag: <tag_id from siebel_images.yaml>
```

iv. Commit the changes to the Git repository:

```
git add .
git commit -m "Updating the image tags parameters"
git push
```

d. Resume flux Helm release for Siebel logging as follows:

i. Go to the environment directory and source the Kubernetes profile:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
```

ii. Confirm that the OpenSearch pods and PVCs are available before updating the OpenSearch controller:

```
oc -n <env_namespace> get pods | grep opensearch
```

```
oc -n <env_namespace> get pvc | grep opensearch
```

iii. Suspend the `oracle-opensearch` Helm release to prevent reconciliation retries during the StatefulSet update:

```
oc -n <env_namespace> patch helmrelease oracle-opensearch --type=merge -p '{"spec":
{"suspend":true}}'
```

iv. Save the current OpenSearch StatefulSet manifest and Helm release history for rollback purposes:

```
oc -n <env_namespace> get sts opensearch-cluster-master -o yaml > /tmp/opensearch-sts-
before.yaml
oc -n <env_namespace> get pvc | grep opensearch
helm history oracle-opensearch -n <env_namespace> --max 20
```

v. Delete the OpenSearch StatefulSet controller and orphan the existing pods:

```
oc -n <env_namespace> delete sts opensearch-cluster-master --cascade=orphan
```

> **Note:** You must not delete the OpenSearch pods, PVCs, or PVs.

vi. Resume and force reconcile `oracle-opensearch` in Flux to apply the updated chart:

```
oc -n <env_namespace> patch helmrelease oracle-opensearch --type=merge -p '{"spec":
{"suspend":false}}'
flux reconcile helmrelease oracle-opensearch -n <env_namespace> --with-source --reset –force
```

vii. Verify that the OpenSearch StatefulSet is recreated or adopted successfully:

```
oc -n <env_namespace> get sts opensearch-cluster-master
oc -n <env_namespace> get pods | grep opensearch
oc -n <env_namespace> get helmrelease oracle-opensearch
helm history oracle-opensearch -n <env_namespace>
```

viii. After `oracle-opensearch` shows `Ready=True`, reconcile OpenSearch Dashboards as follows

```
flux reconcile helmrelease oracle-opensearch-dashboards -n <env_namespace> --with-source --
reset --force
```

ix. Reconcile the Git source and application kustomization:

```
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile -n <env_namespace> kustomization apps
```

In the example, `<env_namespace>` is the name of the flux repository.

> **Note:** If the Helm release is in a failed state, roll back to a previous stable revision. For more information, see *Troubleshooting HelmRelease Failure*.

e. Verify and confirm the Siebel logging updates, ensure that the value of the field SUSPENDED is False and READY is True for `oracle-opensearch`, `oracle-opensearch-dashboard`, and `siebel-logging`:

```
flux get kustomization apps -n <env_namespace>
flux get helmrelease -n <env_namespace>
```

f. Ensure that APP VERSION points to the latest version and is incremented from previous version for `oracle-opensearch`, `oracle-opensearch-dashboard`, and `siebel-logging`:

```
helm list -n <env_namespace>
```

21. Check the status of the flux components after update:

```
flux get all -n <env_namespace>
```

22. Watch out for the successful completion of postinstalldb Kubernetes job.

For more information, see *Reviewing the PostInstallDBSetup Execution Status*.

- The new image updates will trigger `postinstalldb` update through flux-crm sync up.

- Wait for the Kubernetes job completion.
- Manually verify the `postinstalldb` job reports and exit code from the logs.
- In case of errors, take corrective actions and rerun `postinstalldb` Kubernetes job by updating the version in `chart.yaml` file as required for an incremental run.

For more information, see *Making Incremental Changes*.

```
sudo podman exec -it cloudmanager bash source /home/opc/siebel/<env_id>/k8sprofile kubectl -n
 <env_namespace> get pods
```

23. Configuration instructions specific to a release.
    - For any configuration instructions specific to a release, refer to Siebel Upgrade Guide and Siebel Release Notes.
    - Migrate the persistent volume content. Refer to the "Migrating Persistent Volume Content" section in the Deploying Siebel CRM Containers Guide.
24. Upgrading the repository.

    During the update process if any new features require repository upgrade, then upgrade the repository. Refer to Using Siebel Tools Guide.
25. Verify the application URLs once the environment comes up.

Troubleshooting HelmRelease Failure

If you encounter issues with the Helm release, during the Siebel image rollout and Flux reconciliation:

1. Check the Helm release status:
   a. Check the HelmRelease status:

      ```
      kubectl get helmrelease -n <env_namespace>
      ```

      Verify that the `READY` column shows `True` for all HelmRelease resources.
   b. Check the Helm charts deployment status:

      ```
      helm ls -n <env_namespace>
      ```

      Verify that the STATUS column shows `deployed` for all releases.
2. If the Helm release is in failed state, you must roll back to a stable revision and increment the version in `Chart.yaml` for the next Helm upgrade:
   a. Identify the previously deployed stable revision:

      ```
      helm history <Helm Release Name> -n <env_namespace>
      ```

   b. Roll back to the stable revision identified in the helm history output:

      ```
      helm rollback <Helm Release Name> -n <env_namespace> 1
      ```

      For example:

      ```
      W0505 10:56:23.450209 3296 warnings.go:70] would violate PodSecurity "restricted:latest":
       allowPrivilegeEscalation != false (containers "persist-folders", "sai" must set
       securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (containers "persist-
      folders", "sai" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or
       containers "persist-folders", "sai" must set securityContext.runAsNonRoot=true), seccompProfile
       (pod or containers "persist-folders", "sai" must set securityContext.seccompProfile.type to
       "RuntimeDefault" or "Localhost")
      W0505 10:56:23.511704 3296 warnings.go:70] would violate PodSecurity "restricted:latest":
       allowPrivilegeEscalation != false (containers "persist-fix", "ses" must set
       securityContext.allowPrivilegeEscalation=false), unrestricted capabilities
      ```

```
(containers "persist-fix", "ses" must set securityContext.capabilities.drop=["ALL"]),
 runAsNonRoot != true (pod or containers "persist-fix", "ses" must set
 securityContext.runAsNonRoot=true), seccompProfile (pod or containers "persist-fix", "ses" must
 set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
Rollback was a success! Happy Helming!
```

Post-Update Security Hardening for SCM and Siebel Pods

This topic lists the post-update procedure to move an environment to the hardened-security model after the monthly update is complete. It includes the following topics:

- *Prerequisites for Post-Update Security Hardening*
- *Updating SCM with Custom Non-Root User ID and Group ID*
- *Updating Siebel with Custom Non-Root User ID and Group ID*

Starting with version 26.6, SCM and Siebel support running pods with custom non-root user ID and group ID. This help SCM and Siebel workloads run with custom non-root user ID and group ID that align with the target Kubernetes cluster security policy. When you adopt the enhanced security model, you can configure pods to run with specific `runAsUser`, `runAsGroup`, and `fsGroup` values. This configuration also enables persistent file system ownership to align with the pod security context.

Note: This feature is optional. Configure it only if you want to move to the hardened security model or if the target cluster security policy requires specific non-root user, group, or file system group IDs.

Prerequisites for Post-Update Security Hardening

- Ensure that your environment is running version 26.6 or later. If you are running a version earlier than 26.6, update the environment to version 26.6 or later.
- For OpenShift environments, complete the OpenShift-specific post-update SCC configuration steps. For more information, see *Prerequisites for Updating to Version 26.6*.
- For the SCM pod:
  - Select the `runAsUser` and `fsGroup` values.
  - Set `runAsGroup` to `0`.
- For the Siebel pods, select the `runAsUser`, `runAsGroup`, and `fsGroup` values.
- For non-OpenShift Kubernetes clusters, choose any non-root values.
- For OpenShift, describe the SCM and Siebel namespaces to identify the allowed security context ranges:
  - Select `runAsUser` from the `openshift.io/sa.scc.uid-range` value.
  - Select `fsGroup` by using the first ID from `openshift.io/sa.scc.supplemental-groups`.

```
kubectl get namespace <SCM namespace name> -o jsonpath='openshift.io/sa.scc.uid-
range={.metadata.annotations.openshift\.io/sa\.scc\.uid-range}{"\n"}openshift.io/sa.scc.supplemental-
groups={.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}{"\n"}'
kubectl get namespace <Siebel namespace name> -o jsonpath='openshift.io/sa.scc.uid-
range={.metadata.annotations.openshift\.io/sa\.scc\.uid-range}{"\n"}openshift.io/sa.scc.supplemental-
groups={.metadata.annotations.openshift\.io/sa\.scc\.supplemental-groups}{"\n"}'
```

Updating SCM with Custom Non-Root User ID and Group ID

To update SCM with custom non-root user ID and group ID:

1. Go to the new SCM Helm chart directory, typically the `cloudmanager` directory created by the installer.
2. Update the `securityContext` section in `values.yaml`, as follows:

```
securityContext:
 runAsGroup: 0
 runAsUser: <scm_run_as_user>
 fsGroup: <scm_fs_group>
 privileged: false
```

3. Mount the SCM NFS server to an instance where the file system is accessible:

```
sudo mount -t nfs <FILESYSTEM_HOST>:/<export_path> <mount_directory> -o nolock
```

4. Change ownership of the SCM base directory.

```
cd <mount_directory>
sudo chown -R <scm_run_as_user>:0 <SCM_NAMESPACE_IN_UPPERCASE>
sudo chmod -R g+rwx <SCM_NAMESPACE_IN_UPPERCASE>
```

   In the above command `<scm_run_as_user>` is the user ID specified in `securityContext.runAsUser` and the group owner must be `0`.
5. Run the Helm upgrade from the SCM Helm chart directory:

```
cd <cloudmanager helmchart directory>
helm upgrade <releaseName> . -n <namespace>
```

   In the above command `<releaseName>` is the SCM Helm chart instance identifier and `<namespace>` is the SCM namespace.
6. Verify that the SCM pod is running.

```
kubectl get pods -n <namespace>
```

Updating Siebel with Custom Non-Root User ID and Group ID

To update Siebel with custom non-root user ID and group ID:

- Prepare the Siebel File System. For more information, see *Prerequisites to Update Siebel with Custom Non-Root User ID and Group ID*.
- Update the Siebel Helm releases with custom non-root user ID and group ID. For more information, see *Updating Siebel Helm Releases with Custom Non-Root User ID and Group ID*.
- Update Observability with custom non-root user ID and group ID. For more information, see *Updating Observability with Custom Non-Root User ID and Group ID*.
- Update logging with custom non-root user ID and group ID. For more information, see *Updating Logging with Custom Non-Root User ID and Group ID*.
- Commit and push Helm release updates. For more information, see *Committing and Pushing Helm Release Updates*.
- Update Siebel Operator manifest with custom non-root user ID and group ID. For more information, see *Updating Siebel Operator Manifest with Custom Non-Root User ID and Group ID*.

Prerequisites to Update Siebel with Custom Non-Root User ID and Group ID

Before you update Siebel Helm releases, you must prepare Siebel File System as follows:

Note: If you have multiple Siebel file systems, perform these steps for all of them.

1. Mount every Siebel File System and migration file system to an instance where the file systems are accessible:
   ```
   sudo mount -t nfs <FILESYSTEM_HOST>:/<export_path> <mount_directory_for_sfs_or_migration> -o nolock
   ```
   Note: You must mount each file system to a different directory.
2. For each mounted file system, change ownership of the Siebel base directory.
   ```
   cd <mount_directory_for_sfs_or_migration>
   sudo chown -R <siebel_run_as_user>:0 <SIEBEL_NAMESPACE_IN_UPPERCASE>
   sudo chmod -R g+rwx <SIEBEL_NAMESPACE_IN_UPPERCASE>
   ```
   In the above commands:
   - The Siebel base directory name is the Siebel namespace in uppercase.
   - `<siebel_run_as_user>` is the user ID specified in `securityContext.runAsUser`.
   - The group owner must be `0`.
3. If the Siebel deployment uses a migration file system with a migration directory instead of the namespace-uppercase base directory, update the `migration` directory as follows:
   ```
   sudo chown -R <siebel_run_as_user>:0 migration
   sudo chmod -R g+rwx migration
   ```

Updating Siebel Helm Releases with Custom Non-Root User ID and Group ID

To update Helm release files with custom non-root user ID and group ID:

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Verify that the Flux components, Helm releases, deployments, and pods are healthy and running without errors:
   ```
   cd /home/opc/siebel/<ENV_ID>/
   source k8sprofile

   flux get all -n <env_namespace>
   kubectl get helmrelease -n <env_namespace>
   helm ls -n <env_namespace>
   kubectl get pods -n <env_namespace>
   ```
   You must proceed only when the environment is stable, that is:
   - `flux get all` does not show failing reconciliations.
   - `kubectl get helmrelease` shows `READY=True` for all relevant Helm releases.
   - `helm ls` shows `STATUS=deployed` for all relevant Helm charts.
   - `kubectl get pods` shows workload pods in `Running` state and completed job pods in `Completed` state.
2. Pull the latest Git changes:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager
   ```

```
git pull
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts
git pull
```

3. Update the Siebel Gateway Helm release as follows:
   a. Open the Siebel Gateway Helm release file:

      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/siebel-gateway.yaml
      ```

   b. Under `spec.values`, add the `securityContext` section. If logging is enabled, also add `logging.securityContext.runAsUser`:

      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: siebel-gateway
         namespace: <env_namespace>
      spec:
         releaseName: siebel-gateway
         chart:
            spec:
               chart: ./siebel-gateway
               sourceRef:
                  kind: GitRepository
                  name: <flux_helm_repo_name>
                  namespace: <flux_helm_repo_namespace>
         interval: 30m
         install:
            timeout: 60m
         upgrade:
               timeout: 60m
               remediation:
                  retries: 3
                  remediateLastFailure: true
                  strategy: rollback
         uninstall:
            timeout: 30m
         values:
            image:
               siebel:
                     imagePullPolicy: Always
                     registry: <user_registry_url>
                     repository: <registry_prefix>/<env_namespace>/siebel
                     tag: <target_siebel_image_tag>
               busybox:
                     imagePullPolicy: Always
                     registry: <user_registry_url>
                     repository: <registry_prefix>/<env_namespace>/cm/base-builder
                     tag: <base_builder_tag_from_siebel_images_yaml>
            logging:
                     enableLogging: <true_or_false>
                     image:
                        imagePullPolicy: Always
                        registry: <user_registry_url>
                        repository: <registry_prefix>/<env_namespace>/cm/fluentd-collector
                        tag: <fluentd_collector_tag>
                     securityContext:
                        runAsUser: <siebel_run_as_user>
            securityContext:
                  runAsGroup: <siebel_run_as_group>
                  runAsUser: <siebel_run_as_user>
                  fsGroup: <siebel_fs_group>
      ```

4. Update the Siebel Config Helm release:
   a. Open the Siebel Config Helm release file:

      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/siebel-config.yaml
      ```

   b. Under `spec.values`, add the `securityContext` section. Retain the existing environment-specific Git, registry, database, and image configuration values:

      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: siebel-config
         namespace: <env_namespace>
      spec:
         releaseName: siebel-config
         chart:
            spec:
               chart: ./siebel-config
               sourceRef:
                  kind: GitRepository
                  name: <flux_helm_repo_name>
                  namespace: <flux_helm_repo_namespace>
         interval: 10m
         install:
            timeout: 120m
         upgrade:
            timeout: 120m
            remediation:
               retries: 3
               remediateLastFailure: true
               strategy: rollback
         uninstall:
            timeout: 30m
         values:
            image:
               busybox:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/base-builder
                  tag: <base_builder_tag_from_siebel_images_yaml>
               configure:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/sieb-config
                  tag: <sieb_config_tag_from_siebel_images_yaml>
               dbutils:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/dbutils
                  tag: <dbutils_tag_from_siebel_images_yaml>
               utils:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/utils
                  tag: <utils_tag_from_siebel_images_yaml>
            securityContext:
               runAsGroup: <siebel_run_as_group>
               runAsUser: <siebel_run_as_user>
               fsGroup: <siebel_fs_group>
      ```

5. Update the Siebel Runtime Helm release as follows:
   a. Open the Siebel Runtime Helm release file:
      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/siebel.yaml
      ```
   b. Under `spec.values`, add the `securityContext` section. If logging is enabled, also add `logging.securityContext.runAsUser`:
      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: siebel
         namespace: <env_namespace>
      spec:
         releaseName: siebel
         chart:
            spec:
               chart: ./siebel
               sourceRef:
                  kind: GitRepository
                  name: <flux_helm_repo_name>
                  namespace: <flux_helm_repo_namespace>
         interval: 30m
         install:
            timeout: 120m
         upgrade:
            timeout: 120m
            remediation:
               retries: 3
               remediateLastFailure: true
               strategy: rollback
         uninstall:
            timeout: 30m
         values:
            image:
               siebel:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/siebel
                  tag: <target_siebel_image_tag>
               busybox:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/base-builder
                  tag: <base_builder_tag_from_siebel_images_yaml>
               dbutils:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/dbutils
                  tag: <dbutils_tag_from_siebel_images_yaml>
               utils:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/utils
                  tag: <utils_tag_from_siebel_images_yaml>
            logging:
               enableLogging: <true_or_false>
               image:
                  imagePullPolicy: Always
                  registry: <user_registry_url>
                  repository: <registry_prefix>/<env_namespace>/cm/fluentd-collector
                  tag: <fluentd_collector_tag>
               securityContext:
                  runAsUser: <siebel_run_as_user>
            securityContext:
               runAsGroup: <siebel_run_as_group>
      ```

```
            runAsUser: <siebel_run_as_user>
            fsGroup: <siebel_fs_group>
```

6. Update the Siebel Artifacts Helm release as follows:
   a. Open the Siebel Artifacts Helm release file:
   ```
   vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel/siebel-artifacts.yaml
   ```
   b. Under `spec.values`, add the `securityContext` section:
   ```
   apiVersion: helm.toolkit.fluxcd.io/v2
   kind: HelmRelease
   metadata:
      name: siebel-artifacts
      namespace: <env_namespace>
   spec:
      releaseName: siebel-artifacts
      chart:
         spec:
               chart: ./siebel-artifacts
               sourceRef:
    kind: GitRepository
    name: <flux_helm_repo_name>
    namespace: <flux_helm_repo_namespace>
      interval: 5m
      install:
         timeout: 60m
      upgrade:
         timeout: 60m
         remediation:
            retries: 3
            remediateLastFailure: true
            strategy: rollback
      uninstall:
         timeout: 30m
      values:
         image:
            siebel:
               base_image: <user_registry_url>/<registry_prefix>/<env_namespace>/
   siebel:<target_siebel_version>-full
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/siebel
               tag: <target_custom_siebel_image_tag>
            customImageBuilder:
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/cm/custom_image_builder
               tag: <custom_image_builder_tag_from_siebel_images_yaml>
            busybox:
               imagePullPolicy: Always
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/cm/base-builder
               tag: <base_builder_tag_from_siebel_images_yaml>
         imageBuilder:
            privileged: <false_for_OCNE_and_OpenShift_true_for_other_cluster_types>
         securityContext:
            runAsGroup: <siebel_run_as_group>
            runAsUser: <siebel_run_as_user>
            fsGroup: <siebel_fs_group>
   ```

7. Update the Metacontroller Helm release as follows:
   a. Open the Metacontroller Helm release file:
   ```
   vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/metacontroller/release.yaml
   ```
   b. Under `spec.values`, add the `podSecurityContext` section:
   ```
   values:
   ```

```
image:
   registry: <user_registry_url>
   repository: <registry_prefix>/<env_namespace>/cm/metacontroller
   pullPolicy: Always
   tag: <metacontroller_tag>
podSecurityContext:
   runAsNonRoot: true
   runAsUser: <siebel_run_as_user>
   runAsGroup: <siebel_run_as_group>
   fsGroup: <siebel_fs_group>
```

8. Update the Traefik Helm release as follows:
   a. Open the Traefik Helm release file:
      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/traefik/traefik-controller/
      release.yaml
      ```
   b. Under `spec.values`, add the `podSecurityContext` section:
      ```
      values:
         podSecurityContext:
            runAsGroup: <siebel_run_as_group>
            runAsNonRoot: true
            runAsUser: <siebel_run_as_user>
            seccompProfile:
               type: RuntimeDefault
      ```

In the above examples:

- `runAsNonRoot` ensures that the component runs as a non-root user.
- `<siebel_run_as_user>` specifies the user ID under which the component container processes run.
- `<siebel_run_as_group>` specifies the group ID under which the component container processes run.
- `<siebel_fs_group>` specifies the file system group ID used for access to shared volumes.

For more details about the `securityContext` and `podSecurityContext` parameters, see *Payload Parameters for Siebel CRM Deployment*.

Updating Observability with Custom Non-Root User ID and Group ID

If observability is enabled, update the observability Helm release files with the selected `runAsUser`, `runAsGroup`, and `fsGroup` values. You can use non-root user ID and group ID that are different from the Siebel non-root user ID and group ID or the same values as the Siebel non-root user ID and group ID.

If Prometheus uses a local-storage persistent volume, ensure that the local storage directory is owned by `<observability_run_as_user>:<observability_fs_group>` before you reconcile the Helm release changes:

```
sudo chown -R <observability_run_as_user>:<observability_fs_group> <local_storage_directory_path>
```

To update Observability with custom non-root user ID and group ID:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Update the Prometheus Helm release as follows:
   a. Open the Prometheus Helm release file:
      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/
      siebel_observability/prometheus.yaml
      ```
   b. Under `spec.values`, add the `securityContext` section:

```
apiVersion: helm.toolkit.fluxcd.io/v2
kind: HelmRelease
metadata:
   name: prometheus
   namespace: <env_namespace>
spec:
   releaseName: prometheus
   chart:
      spec:
         chart: ./prometheus
            sourceRef:
            kind: GitRepository
            name: <flux_helm_repo_name>
            namespace: <flux_helm_repo_namespace>
   interval: 10m
   install:
      timeout: 120m
      remediation:
         retries: 3
      remediateLastFailure: true
   upgrade:
      timeout: 120m
      remediation:
      retries: 3
      remediateLastFailure: true
      strategy: rollback
   uninstall:
      timeout: 30m
   values:
      alerting: <true_or_false>
      clusterDns: svc.cluster.local
      dbMonitoring: <true_or_false>
      configmap_reload:
         image:
         imagePullPolicy: Always
         registry: <user_registry_url>
         repository: <registry_prefix>/<env_namespace>/cm/configmap-reload
         tag: <configmap_reload_tag>
         securityContext:
            runAsGroup: <observability_run_as_group>
            runAsUser: <observability_run_as_user>
            fsGroup: <observability_fs_group>
```

2. Update the Prometheus Alertmanager Helm release as follows:
   a. Open the Prometheus Alertmanager Helm release file:

      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/
      siebel_observability/prometheus_alertmanager.yaml
      ```

   b. Under `spec.values`, add the `securityContext` section:

      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
        name: prometheus-alertmanager
        namespace: <env_namespace>
      spec:
        releaseName: prometheus-alertmanager
        chart:
          spec:
            chart: ./prometheus-alert-manager
            sourceRef:
              kind: GitRepository
              name: <flux_helm_repo_name>
              namespace: <flux_helm_repo_namespace>
        interval: 5m
        install:
      ```

```
    timeout: 120m
  upgrade:
    timeout: 120m
    remediation:
      retries: 3
      remediateLastFailure: true
      strategy: rollback
  uninstall:
    timeout: 30m
  values:
    alertmanager:
      image:
        registry: <user_registry_url>
        repository: <registry_prefix>/<env_namespace>/cm/alertmanager
        tag: <alertmanager_tag>
        pullPolicy: Always
    securityContext:
      runAsGroup: <observability_run_as_group>
      runAsUser: <observability_run_as_user>
      fsGroup: <observability_fs_group>
```

3. Update the Prometheus Adapter Helm release as follows:
   a. Open the Prometheus Adapter Helm release file:

```
vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/
siebel_observability/prometheus_adapter.yaml
```

   b. Under `spec.values`, add the `podSecurityContext` section:

```
apiVersion: helm.toolkit.fluxcd.io/v2
kind: HelmRelease
metadata:
   name: prometheus-adapter
   namespace: <env_namespace>
spec:
   releaseName: prometheus-adapter
   chart:
   spec:
      chart: ./prometheus-adapter
      sourceRef:
         kind: GitRepository
         name: <flux_helm_repo_name>
         namespace: <flux_helm_repo_namespace>
   interval: 5m
   dependsOn:
         - name: prometheus
   install:
      timeout: 120m
   upgrade:
      timeout: 120m
      remediation:
      retries: 3
      remediateLastFailure: true
      strategy: rollback
   uninstall:
      timeout: 30m
   values:
      prometheus_adapter:
      image:
   imagePullPolicy: Always
      registry: <user_registry_url>
      repository: <registry_prefix>/<env_namespace>/cm/prometheus-adapter
      tag: <prometheus_adapter_tag>
   podSecurityContext:
      runAsUser: <observability_run_as_user>
      runAsGroup: <observability_run_as_group>
```

```
        fsGroup: <observability_fs_group>
```

4. Update the Kube State Metrics Helm release as follows:
   a. Open the Kube State Metrics Helm release file:
   ```
   vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/
   siebel_observability/prometheus_kube_state_metrics.yaml
   ```
   b. Under `spec.values`, add the `securityContext` section:
   ```
   apiVersion: helm.toolkit.fluxcd.io/v2
   kind: HelmRelease
   metadata:
      name: kube-state-metrics
      namespace: <env_namespace>
   spec:
      releaseName: kube-state-metrics
      chart:
         spec:
      chart: ./kube-state-metrics
      sourceRef:
         kind: GitRepository
         name: <flux_helm_repo_name>
         namespace: <flux_helm_repo_namespace>
      interval: 5m
      install:
         timeout: 120m
      upgrade:
         timeout: 120m
         remediation:
         retries: 3
         remediateLastFailure: true
         strategy: rollback
      uninstall:
      timeout: 30m
      values:
         kube_state_metrics:
         image:
            registry: <user_registry_url>
            repository: <registry_prefix>/<env_namespace>/cm/kube-state-metrics
            tag: <kube_state_metrics_tag>
            pullPolicy: Always
         securityContext:
            runAsGroup: <observability_run_as_group>
            runAsUser: <observability_run_as_user>
            fsGroup: <observability_fs_group>
   ```

5. Update the Node Exporter Helm release as follows:
   a. Open the Node Exporter Helm release file:
   ```
   vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/
   siebel_observability/prometheus_node_exporter.yaml
   ```
   b. Under `spec.values`, add the `securityContext` section:
   ```
   apiVersion: helm.toolkit.fluxcd.io/v2
   kind: HelmRelease
   metadata:
      name: node-exporter
      namespace: <env_namespace>
   spec:
      releaseName: node-exporter
      chart:
         spec:
            chart: ./node-exporter
            sourceRef:
            kind: GitRepository
            name: <flux_helm_repo_name>
   ```

```
      namespace: <flux_helm_repo_namespace>
interval: 5m
install:
   timeout: 120m
upgrade:
   timeout: 120m
   remediation:
      retries: 3
      remediateLastFailure: true
      strategy: rollback
uninstall:
   timeout: 30m
values:
   node_exporter:
      image:
      registry: <user_registry_url>
      repository: <registry_prefix>/<env_namespace>/cm/node-exporter
      tag: <node_exporter_tag>
      pullPolicy: Always
   securityContext:
      runAsGroup: <observability_run_as_group>
      runAsUser: <observability_run_as_user>
      fsGroup: <observability_fs_group>
```

In the above examples:

- `<observability_run_as_user>` specifies the user ID under which the observability component container processes run.
- `<observability_run_as_group>` specifies the group ID under which the observability container processes run.
- `<observability_fs_group>` specifies the file system group ID used for access to shared volumes.

Updating Logging with Custom Non-Root User ID and Group ID

If logging is enabled, update the logging Helm release files with the selected user ID and group ID. These changes apply to the Log Aggregator, Node Log Collector, Oracle OpenSearch, and Oracle OpenSearch Dashboards components.

The logging sidecars inside Siebel pods are configured separately in the `siebel-gateway.yaml` and `siebel.yaml` Helm release files by using `logging.securityContext.runAsUser`.

To update logging with custom non-root user ID and group ID:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Update the Siebel Logging Helm release as follows:
   a. Open the Siebel Logging Helm release file:
      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel-logging/
      siebel-logging.yaml
      ```
   b. Under `spec.values`, add the `podSecurityContext` section:
      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: siebel-logging
         namespace: <env_namespace>
      spec:
         releaseName: siebel-logging
         chart:
            spec:
      ```

```
            chart: ./siebel-logging
            sourceRef:
               kind: GitRepository
               name: <flux_helm_repo_name>
               namespace: <flux_helm_repo_namespace>
      interval: 5m
      install:
         timeout: 120m
      upgrade:
         timeout: 120m
         remediation:
            retries: 3
            remediateLastFailure: true
            strategy: rollback
      uninstall:
         timeout: 30m
      values:
         logging:
            enableLogging: <true_or_false>
            enableOracleOpensearch: <true_or_false>
            enableOCILogan: <true_or_false>
            image:
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/cm/log-aggregator
               tag: <log_aggregator_tag>
               imagePullPolicy: Always
            logCollectorImage:
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/cm/fluentd-collector
               tag: <fluentd_collector_tag>
            podSecurityContext:
               runAsUser: <logging_run_as_user>
               runAsGroup: <logging_run_as_group>
               fsGroup: <logging_fs_group>
```

2. Update the Oracle OpenSearch Helm release as follows:
   a. Open the Oracle OpenSearch Helm release file:

      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel-logging/
      oracle-opensearch.yaml
      ```

   b. Under `spec.values`, add the `podSecurityContext` section:

      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: oracle-opensearch
         namespace: <env_namespace>
      spec:
         releaseName: oracle-opensearch
         chart:
            spec:
               chart: ./oracle-opensearch
               sourceRef:
       kind: GitRepository
       name: <flux_helm_repo_name>
       namespace: <flux_helm_repo_namespace>
         interval: 10m
         install:
            timeout: 120m
            strategy:
               name: RetryOnFailure
               retryInterval: 5m
         upgrade:
            timeout: 120m
            remediation:
               retries: 3
      ```

```
            remediateLastFailure: true
            strategy: rollback
      uninstall:
         timeout: 30m
      values:
         image:
            opensearch:
               registry: <user_registry_url>
               repository: <registry_prefix>/<env_namespace>/cm/oracle-opensearch
               tag: <oracle_opensearch_tag>
               imagePullPolicy: Always
            podSecurityContext:
               fsGroup: <logging_fs_group>
               runAsUser: <logging_run_as_user>
```

3. Update the Oracle OpenSearch Dashboards Helm release as follows:
   a. Open the Oracle OpenSearch Dashboards Helm release file:

      ```
      vi /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager/flux-crm/apps/base/siebel-logging/
      oracle-opensearch-dashboards.yaml
      ```

   b. Under `spec.values`, add the `podSecurityContext` section:

      ```
      apiVersion: helm.toolkit.fluxcd.io/v2
      kind: HelmRelease
      metadata:
         name: oracle-opensearch-dashboards
         namespace: <env_namespace>
      spec:
         releaseName: oracle-opensearch-dashboards
         chart:
               spec:
                  chart: ./oracle-opensearch-dashboards
                  sourceRef:
                     kind: GitRepository
                     name: <flux_helm_repo_name>
                     namespace: <flux_helm_repo_namespace>
         interval: 10m
         install:
               timeout: 120m
         upgrade:
               timeout: 120m
               remediation:
               retries: 3
                  remediateLastFailure: true
                  strategy: rollback
         uninstall:
               timeout: 30m
         values:
               image:
                  opensearch_dashboards:
                     registry: <user_registry_url>
                     repository: <registry_prefix>/<env_namespace>/cm/oracle-opensearch-dashboards
                     tag: <oracle_opensearch_dashboards_tag>
                     imagePullPolicy: Always
                  podSecurityContext:
                     runAsUser: <logging_run_as_user>
                     fsGroup: <logging_fs_group>
      ```

In the above examples:

- `<logging_run_as_user>` specifies the user ID under which the logging container processes run.
- `<logging_run_as_group>` specifies the group ID under which the logging container processes run.
- `<logging_fs_group>` specifies the file system group ID used for access to shared volumes and persistent storage.

Committing and Pushing Helm Release Updates

After you update the Helm release YAML files with the required user ID and group ID values, you must review the changes, commit them to the SCM GitOps repository, and then push the commit to the remote repository, as follows:

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Review the changes in the Helm release YAML files:
```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager
git status
git diff
```
2. Stage the updated files for commit:
```
git add flux-crm/apps/base/siebel/*.yaml \
   flux-crm/apps/base/siebel_observability/*.yaml \
   flux-crm/apps/base/siebel-logging/*.yaml \
   flux-crm/metacontroller/release.yaml \
   flux-crm/traefik/traefik-controller/release.yaml
```
3. Commit the changes to SCM GitOps repository:
```
git commit -m "Update runtime IDs for Siebel and observability Helm releases"
```
4. Push the commit to the remote GitOps repository so that Flux can reconcile the updated desired state:
```
git push
```
   After you push the Helm release changes, Flux reconciles the updated GitOps state and applies the Helm upgrade.
5. Monitor reconciliation until the affected Helm release resources become ready, as follows:
```
cd /home/opc/siebel/<ENV_ID>/ source k8sprofile
flux get helmrelease -n <env_namespace> helm ls -n <env_namespace>
kubectl get pods -n <env_namespace>
```
   Wait for reconciliation to complete. Confirm that the affected Helm releases show `READY=True`.
6. Check the Helm status:
```
helm ls -n <env_namespace>
```
   Confirm that the affected releases show `STATUS=deployed`.
7. Check the pod status:
```
kubectl get pods -n <env_namespace>
```
   Confirm that the affected pods are running successfully.

Updating Siebel Operator Manifest with Custom Non-Root User ID and Group ID

To update the Siebel Operator manifest:

1. Open the Siebel Operator manifest file:
```
vi /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts/siebel-operator/manifest/siebel-operator.yaml
```
2. Add the `securityContext` section under `spec.template.spec`:
```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: siebel-controller
```

```
spec:
 template:
 spec:
 serviceAccountName: get-and-create-configmaps
 containers:
 - name: controller
 image: siebel-operator-base:<siebel_operator_base_tag>
 command: ["/bin/bash", "-c", "python /hooks/sync.py"]
 securityContext:
 runAsUser: <siebel_run_as_user>
 runAsGroup: 0
 fsGroup: <siebel_fs_group>
```

3. Commit, push, and reconcile the Siebel Operator changes:

```
cd /home/opc/siebel/<ENV_ID>/<env_namespace>-helmcharts
git status
git diff siebel-operator/manifest/siebel-operator.yaml
git add siebel-operator/manifest/siebel-operator.yaml
git commit -m "Update Siebel operator runtime security context"
git push
```

4. Reconcile the Git source and the Siebel Operator Kustomization:

```
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile kustomization siebel-operator -n <env_namespace>
```

5. Verify that the Kustomization is ready and that the operator pod is running:

```
flux get kustomization siebel-operator -n <env_namespace>
kubectl get pods -n <env_namespace> | grep siebel-controller
```

Migrating Siebel CRM Containers to OpenSSL Encryption

This topic describes the steps to migrate Siebel CRM containers to OpenSSL encryption. It includes the following sections:

- *Pre-Encryption Migration Phase*
- *Encryption Migration Phase*
- *Post-Encryption Migration Phase*

Starting with Siebel CRM 25.10, you can use an improved encryption model based on OpenSSL libraries when you update existing container-based Siebel CRM deployments from a release earlier than 25.10 to 25.10 or later. The OpenSSL encryption migration affects multiple layers of the Siebel CRM ecosystem, such as Kubernetes workloads, the persistent file system content, and ZooKeeper or registry configuration. As a result, you must follow a controlled update sequence.

You can use SCM to run the encryption migration through an automated process. This process standardizes the steps and helps minimize downtime during the update. The migration process includes the following three phases:

- Pre-encryption migration phase: Scales down key workloads and creates backups and a Git checkpoint tag.
- Encryption migration phase: Runs a Kubernetes job that performs the encryption migration.
- Post-encryption migration phase: Updates manifests to the target version, scales workloads back up, and validates health of the environment.

The automated migration capability gives you a consistent operator experience. It reduces manual work, improves repeatability, and helps you minimize downtime when you update SCM created Siebel CRM environments to Siebel CRM 25.10 or later with OpenSSL-based encryption.

> **Note:** The OpenSSL encryption migration is a breaking change. After you enable it, you can't start Siebel CRM servers earlier than 25.10 with the updated encryption state.

Pre-Encryption Migration Phase

In the pre-encryption migration phase, SCM scales down the Siebel Enterprise Server (SES) and Siebel Gateway (CGW) workloads in a controlled way. It mounts the shared file system and backs up critical runtime directories and the Gateway registry configuration. As a checkpoint, SCM also creates a Git tag in the environment's Flux repository (for example, `<siebel_version>_pre_encryption`). This tag captures the exact desired state before you make the encryption change.

This phase includes the following tasks:

- *Running the Pre-Encryption Migration Script*
- *Verifying Pre-Encryption Migration Updates*

Running the Pre-Encryption Migration Script

You must run the `pre_openssl_encryption.sh` script to perform the pre-encryption migration tasks.

To run the `pre_openssl_encryption.sh` script:

1. Go to the `cmapp` directory:
   ```
   cd /home/opc/siebel-cloud-manager/scripts/cmapp
   ```
2. Run the pre-encryption migration shell script:
   ```
   bash pre_openssl_encryption.sh
   ```
3. Enter the ID of the environment that you want to encrypt.

The pre-encryption migration script performs tasks such as the following:

- Scale down the CGW and SES StatefulSet replicas to 0.
- Create local backups of `gtwysrvr/registry/conf` and `siebsrvr/sys`.
- Create and push a Git tag named `<siebel_source_version>_pre_encryption` to the Flux repository.
- Roll out CGW pods for the target Siebel version.
- Scale the CGW StatefulSet back to the desired replica count.

> **Note:** The above list shows only a subset of the tasks that the pre-encryption migration script performs. It does not include every task the script performs

Verifying Pre-Encryption Migration Updates

You can verify that the pre-encryption migration script ran successfully as follows:

1. Verify the log to confirm that the CGW pods are running. Look for a message such as "Pre-steps completed successfully," which confirms that the pre-steps completed successfully.

2. Verify the StatefulSet replica status as follows:
   a. Load the Kubernetes profile for your environment:
      ```
      source ~/siebel/<ENV_ID>/k8sprofile
      ```
   b. Check the StatefulSet replica status using `kubectl`:
      ```
      kubectl -n <env_namespace> get sts siebelcgw edge
      ```

   Confirm that CGW (`siebelcgw`) pods show the expected READY count (for example, 3/3) and that the SES component (for example, edge) reflects the pre-migration scale state (for example, 0/0).
3. Verify that the backup directories are created:
   ```
   cd /home/opc/siebel/<ENV_ID>/
   ls -tlr <ENV_ID>-encryption_migration_backup/
   ```

   Confirm that the backup folder exists and includes component directories such as `cgw` and `edge` (for the SES component).
4. Verify that SCM created the pre-encryption Git tag:
   ```
   cd /home/opc/siebel/<ENV_ID>/<env_namespace>-cloud-manager
   git tag -l "*pre_encryption"
   ```

   Confirm that you see a tag such as `<siebel_source_version>_pre_encryption` (for example, 25.9_pre_encryption).

Encryption Migration Phase

In the encryption migration phase, SCM deploys a Kubernetes job that runs the encryption migration in a consistent way with clear logging. The job uses environment-specific connectivity and storage settings, runs with the correct target Siebel CRM container image, and is monitored until it completes with automated failure detection and diagnostic collection. This phase includes the following tasks:

- *Running the Encryption Migration Script*
- *Verifying Encryption Migration Updates*

Running the Encryption Migration Script

You must run the `openssl_encryption_migration.sh` script to perform the encryption migration. When you run this script, it prepares the migration job manifest and applies the encryption migration to the Siebel containers.

To run the `openssl_encryption_migration.sh` script:

1. Go to the `cmapp` directory:
   ```
   cd /home/opc/siebel-cloud-manager/scripts/cmapp
   ```
2. Run the encryption migration shell script:
   ```
   bash openssl_encryption_migration.sh
   ```
3. Enter the ID of the environment that you want to encrypt.

Verifying Encryption Migration Updates

You can verify that the encryption migration script ran successfully as follows:

1. Verify the status of the encryption migration job, as follows:
   a. Load the Kubernetes profile for your environment:

```
source ~/siebel/<ENV_ID>/k8sprofile
```

b. Check the status of the encryption migration job pod using `kubectl`:

```
kubectl -n <env_namespace> get pods
```

2. Review the encryption migration job log:

```
kubectl -n <env_namespace> logs po/<pod name>
```

If the log includes:

- A success message such as "Encryption Upgrade Utility executed successfully" and a reference to the detailed log file (for example, `/siebel/mde/siebsrvr/log/encryptupg.log`), the migration is successful.
- An error message such as "ORA-00955: name is already used by an existing object" and "Error during s_app_ver backup", it indicates that the `S_APP_VER_BAK` table already exists. To resolve this issue, you must do the following:
  a. Drop the `S_APP_VER_BAK` table from the database schema, as follows:
     i. Connect to the database using the table owner user.
     ii. Execute the following SQL commands:

```
DROP TABLE S_APP_VER PURGE;
ALTER TABLE S_APP_VER_BAK RENAME TO S_APP_VER;
GRANT SELECT, INSERT, UPDATE, DELETE ON S_APP_VER TO SSE_ROLE;
UPDATE S_APP_VER SET ENCRYPT_PWD_FL_KEY = NULL;
```

  b. Rerun the encryption migration shell script:

```
bash openssl_encryption_migration.sh
```

Post-Encryption Migration Phase

In the post-encryption migration phase, SCM updates the Flux-managed Siebel CRM manifests to the target image level, triggers Flux reconciliation, scales the SES workloads back to the desired capacity, and confirms that the environment returns to a healthy Running or Ready state. This phase includes the following tasks:

- *Running the Post-Encryption Migration Script*
- *Verifying Post-Encryption Migration Updates*

Running the Post-Encryption Migration Script

You must run the `post_openssl_encryption.sh` script to perform the post-encryption migration tasks.

To run the `post_openssl_encryption.sh` script:

1. Go to the `cmapp` directory:

```
cd /home/opc/siebel-cloud-manager/scripts/cmapp
```

2. Execute the post-encryption migration shell script:

```
bash post_openssl_encryption.sh
```

3. Enter the ID of the environment that you want to encrypt.
   The post-encryption migration script performs tasks such as the following:
   - Roll out new target Siebel version pods for Siebel Server (for example, `edge`).
   - Scale the Siebel server (for example, `edge`) StatefulSet to the desired replicas.

Verifying Post-Encryption Migration Updates

You can verify that the post-encryption migration script ran successfully as follows:

1. Verify the status of the post-encryption migration job, as follows:
   a. Load the Kubernetes profile for your environment:
      `source ~/siebel/<ENV_ID>/k8sprofile`
   b. Check that the SES pods are running:
      `kubectl -n <env_namespace> get pods`
2. Verify the logs. Look for a message such as "Post-steps completed successfully", which confirms that the post-steps completed successfully.

Configuring the Time Zone for SCM and Siebel Pods

This topic describes how to configure a custom time zone for SCM and Siebel components, including Siebel Server, SAI Server, CGW, and SMC pods. It also describes how to verify that the updated time zone is applied successfully. It includes the following sections:

- *Configuring SCM to Use a Specific Time Zone*
- *Configuring Siebel Server Pods to Use a Specific Time Zone*

Configuring SCM to Use a Specific Time Zone

To configure SCM to use a specific time zone:

1. Verify the current time zone of the SCM pod as follows:

```
kubectl exec -it scm-5765f9f4d6-jplmd -n scm-261 -- bash
bash-5.1$ date
```

   Sample output:

```
Thu Feb 5 06:06:58 UTC 2026
```

2. Update the SCM Helm chart deployment file to set the required time zone, as follows:

   In the `<scm Helm chart directory>/templates/deployment.yaml` file, under the `cloudmanager` container definition, add the following environment variable:

```
- name: cloudmanager
 image: {{ template "scm.image" . }}
 imagePullPolicy: {{ .Values.image.imagePullPolicy }}
 ports:
 {{- if and (eq $serviceType "LoadBalancer") (eq .Values.service.loadBalancer.useDualTls "False") }}
 - name: cm-port-http
 containerPort: {{ .Values.port.scmPortHTTP }}
 protocol: TCP
 {{- else }}
 - name: cm-port-https
 containerPort: {{ .Values.port.scmPortHTTPS }}
 protocol: TCP
 {{- end }}
```

```
env:
- name: oci_cli_auth
value: api_key
- name: POD_NAMESPACE
valueFrom:
fieldRef:
fieldPath: metadata.namespace
- name: TZ
value: "Asia/Kolkata"
```

3. Upgrade the Helm release to apply the changes. For example:

```
helm upgrade scm-261 . -n scm-261
```

Sample output:

```
Release "scm-261" has been upgraded. Happy Helming!
NAME: scm-261
LAST DEPLOYED: Thu Feb 5 06:17:11 2026
NAMESPACE: scm-261
STATUS: deployed
REVISION: 3
```

4. Verify the time zone of the SCM Pod. For example:

```
kubectl exec -it scm-864fc946f6-xxxpx -n scm-261 – bash
bash-5.1$ date
```

Sample output:

```
Thu Feb 5 11:48:29 IST 2026
```

After the upgrade, the SCM pod runs with the IST (`Asia/Kolkata`) time zone configured in the deployment YAML.

Configuring Siebel Server Pods to Use a Specific Time Zone

This topic describes how to configure a custom time zone for Siebel Server, SAI Server, CGW, and SMC by using environment variables. It also describes how to apply the configuration changes through SCM and verify that the updated time zone is in effect.

> **Note:** If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

To configure a Siebel Server pod to use a specific time zone:

1. Verify the current time zone of the Siebel Server pod. For example:

```
kubectl -n sbl261 exec -it edge-0 -- bash
bash-5.1$ date
```

Sample output:

```
Tue Feb 10 09:22:49 UTC 2026
```

2. Configure the time zone for Siebel Server and SAI Server pods as follows:
   a. Open the `siebel.yaml` file in the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory.

b. Add the `envlist` parameter under the appropriate server definitions. For example:
   - For each profile (for example, `quantum` and `edge`), add the following configuration under the corresponding server definition:

```
envlist:
- name: TZ
 value: "Asia/Kolkata"
```

   - SAI Server example:

```
saiServer:
- profile: ai_financialservices_greenfield
 replicas: 1
 sai_prefix: quantum
 envlist:
 - name: TZ
 value: "Asia/Kolkata"
```

   - Siebel Server example:

```
siebelServer:
- profile: siebel
 replicas: 1
 sesResources:
 limits:
 cpu: 4
 memory: 24Gi
 requests:
 cpu: 1
 memory: 8Gi
 siebsrvr_prefix: edge
 envlist:
 - name: TZ
 value: "Asia/Kolkata"
```

c. Commit the changes to the SCM Git repository:

```
git pull
git add .
git commit -m "<message>"
git push
```

3. Configure the time zone for CGW and SMC pods as follows:
   a. Open the `siebel-gateway.yaml` file, in the `/home/opc/siebel/<env_id>/<namespace>-cloud-manager/flux-crm/apps/base/siebel` directory.
   b. Add the `envlist` parameter under the `cgw` and `smc` sections. For example:

```
envlist:
- name: TZ
 value: "Asia/Kolkata"
```

   - CGW example:

```
cgw:
 affinity:
 podAntiAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - podAffinityTerm:
 labelSelector:
 matchExpressions:
 - key: app.siebel.tier
 operator: In
 values:
 - cgw
 topologyKey: kubernetes.io/hostname
 weight: 100
 replicas: 3
 envlist:
```

```
    - name: TZ
    value: "Asia/Kolkata"
```

- SMC example:

```
smc:
 affinity: {}
 envlist:
 - name: TZ
 value: "Asia/Kolkata"
```

c. Commit the changes to the SCM Git repository:

```
git pull
git add .
git commit -m "<message>"
git push
```

4. Verify the time zone configuration:

   After reconciliation completes, all Siebel component pods, and the ingress controller will run with the configured IST (`Asia/Kolkata`) time zone instead of the default UTC time zone. You can verify the time zone configuration as follows:

```
kubectl -n sbl261 exec -it edge-0 -- bash
bash-5.1$ date
```

   Sample output:

```
Tue Feb 10 14:41:57 IST 2026
```

> **Note:** This example uses the IST (`Asia/Kolkata`) time zone. You can configure any supported time zone based on your requirements.

Enabling Authentication for Siebel CRM Observability

SCM protects the public Prometheus and OpenSearch Dashboards URLs with HTTP Basic Authentication. Starting with SCM 26.8, you must configure authentication for each enabled dashboard.

You must configure authentication separately for each observability component:

- When Siebel CRM Observability - Monitoring is enabled, configure Prometheus authentication in the `prometheus_auth` section of the Siebel CRM deployment payload.
- When Siebel CRM Observability - Logging is enabled with Oracle OpenSearch, configure OpenSearch Dashboards authentication in the `opensearch_dashboard_auth` section of the Siebel deployment payload.

For more information about the parameters in each section, see *Payload Parameters for Siebel CRM Deployment*.

You can configure one or more users for each dashboard under the `users` section.

- For monitoring, configure `observability.prometheus_auth.basic_auth.users`.
- For Oracle OpenSearch logging, configure `observability.opensearch_dashboard_auth.basic_auth.users`.

You do not need to create these users in advance. User names must be unique within the corresponding authentication configuration. SCM configures dashboard access from the values you provide in the deployment payload. You must configure users separately for each dashboard. Their credentials grant access only to the corresponding dashboard.

You can enable authentication for:

- A new Siebel CRM environment during provisioning. For more information on enabling authentication during environment provisioning, see *Enabling Authentication During Environment Provisioning*.
- An existing Siebel CRM environment. For more information on enabling authentication for an existing environment, see *Enabling Authentication for an Existing Environment*.

Enabling Authentication During Environment Provisioning

When you provision a new environment, you can include the authentication parameters in the `observability` section of the Siebel CRM deployment payload.

The following example enables monitoring and Oracle OpenSearch logging, and configures authentication for both dashboards:

```
{
   "observability": {
      "siebel_monitoring": true,
      "siebel_logging": true,
      "enable_oracle_opensearch": true,
      "prometheus_auth": {
         "basic_auth": {
            "users": [
               {
                  "username": "<prometheus_username_1>",
                  "password": "<prometheus_password_or_secret_ocid_1>"
               }
            ]
         }
      },
      "opensearch_dashboard_auth": {
         "basic_auth": {
            "users": [
               {
                  "username": "<opensearch_dashboard_username_1>",
                  "password": "<opensearch_dashboard_password_or_secret_ocid_1>"
               }
            ]
         }
      }
   }
}
```

For more information about the parameters in each section, see *Payload Parameters for Siebel CRM Deployment*.

Enabling Authentication for an Existing Environment

You can use this procedure to enable authentication when monitoring or Oracle OpenSearch logging is already enabled for an existing Siebel CRM environment. This procedure updates the generated GitOps repository and reconciles the changes through Flux.

Before you begin, you must obtain the environment ID, environment namespace, generated GitOps repository path, and Traefik load balancer IP address.

To enable authentication for an existing environment:

Note: If you use SSH for Git, you must set the `GIT_SSH_COMMAND` environment variable in the current shell before you run your first remote Git command. For more information, see the "Configuring SSH for Manual Git Operations" section in *Using Existing Git Repositories*.

1. Source the Kubernetes profile:
```
sudo podman exec -it cloudmanager bash
cd /home/opc/siebel/<ENV_ID>/
source k8sprofile
```
2. Suspend Traefik reconciliation:
```
flux suspend kustomization traefik-controller traefik-resources -n <env_namespace>
flux suspend helmrelease traefik -n <env_namespace>
```
3. Generate an APR1 password hash for each user. Copy only the generated value that begins with `$apr1$`.
   - For Prometheus:
```
read -rsp 'Prometheus auth password: ' PROM_PASS; echo
printf '%s\n' "$PROM_PASS" | openssl passwd -apr1 -stdin
unset PROM_PASS
```
   - For OpenSearch Dashboards:
```
read -rsp 'OpenSearch Dashboard auth password: ' OS_PASS; echo
printf '%s\n' "$OS_PASS" | openssl passwd -apr1 -stdin
unset OS_PASS
```
4. Update the Traefik Helm Release as follows:
   a. Open `<Cloud manager repository name>/traefik/traefik-controller/release.yaml` in edit mode.
   b. Under `spec.values`, add or update the following section:
```
spec:
   values:
      observabilityAuth:
         labels:
            oracle.siebel.namespace-envid: <env_namespace>-<env_id_lower>
         prometheus:
            enabled: true
            secretName: siebel-prometheus-basic-auth
            middlewareName: siebel-prometheus-basic-auth
            users: |
               PromMon:$apr1$<hash>
            opensearchDashboard:
               enabled: true
               secretName: siebel-opensearch-basic-auth
               middlewareName: siebel-opensearch-basic-auth
               users: |
                  LogMon:$apr1$<hash>
```
5. Update the Prometheus IngressRoute as follows:
   a. Open `<Cloud manager repository name>/traefik/traefik-resources/siebel-ingress-prometheus.yaml` in edit mode.
   b. Add the `siebel-prometheus-basic-auth` middleware to the Prometheus route:
```
routes:
   - match: PathPrefix(`/prometheus`)
      kind: Rule
      middlewares:
      - name: siebel-prometheus-basic-auth
        services:
      - name: prometheus-service
         port: 8080
```

6. Update the OpenSearch Dashboards IngressRoute as follows:
   a. Open `<Cloud manager repository name>/traefik/traefik-resources/siebel-ingress-opensearch.yaml` in edit mode.
   b. Add the `siebel-opensearch-basic-auth` middleware to the OpenSearch Dashboards route:

```
routes:
   - match: PathPrefix(`/opensearch`)
      kind: Rule
      middlewares:
      - name: siebel-opensearch-basic-auth
      services:
      - name: opensearch-dashboards
        port: 5601
```

7. Commit and push the changes:

```
git status
git add <Cloud manager repository name>/traefik/traefik-controller/release.yaml
git add <Cloud manager repository name>/traefik/traefik-resources/siebel-ingress-prometheus.yaml
git add <Cloud manager repository name>/traefik/traefik-resources/siebel-ingress-opensearch.yaml
git commit -m "enable observability ingress auth"
git push
git status
```

8. Resume and reconcile Traefik resources:

```
cd /home/opc/siebel/<ENV_ID>/
 source k8sprofile

flux resume kustomization traefik-controller traefik-resources -n <env_namespace>
flux resume helmrelease traefik -n <env_namespace>

flux reconcile source git siebel-repo -n <env_namespace>
flux reconcile source git <env_namespace>-repo -n <env_namespace>
flux reconcile kustomization traefik-controller -n <env_namespace>
flux reconcile kustomization traefik-resources -n <env_namespace>
flux reconcile helmrelease traefik -n <env_namespace>
```

9. Validate access.

   After Flux reconciliation is complete, you can access the enabled dashboards using the configured Basic Authentication credentials:

   - Use the Prometheus user name and password to access the Prometheus dashboard.
   - Use the OpenSearch Dashboards user name and password to access OpenSearch Dashboards.

   Note: You can add a user after observability authentication is enabled. To add a user:

   a. Open `/home/opc/siebel/<ENV_ID>/<Cloud manager repository name>/flux-crm/traefik/traefik-controller/release.yaml` in edit mode.
   b. Add a new `username:APR1-password-hash` entry on a separate line under the applicable `users` section:
      - `spec.values.observabilityAuth.prometheus.users` for Prometheus.
      - `spec.values.observabilityAuth.opensearchDashboard.users` for OpenSearch Dashboards.

      Note: Retain all existing user entries.
   c. Generate an APR1 password hash for the new user.
   d. Commit and push the updated `release.yaml` file.
   e. Reconcile the `traefik-controller` Kustomization.

   For information about generating an APR1 password hash, see step 3 “Generate an APR1 password hash for each user.”