Oracle® Communications

Cloud Native Core Security Guide

Release 24.3.0
G15056-01
October 2024

Oracle Communications Cloud Native Core Security Guide, Release 24.3.0

G15056-01

Contents

My Oracle Support

My Oracle Support (https://support.oracle.com) is your initial point of contact for all product support and training needs. A representative at Customer Access Support can assist you with My Oracle Support registration.

Call the Customer Access Support main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. When calling, make the selections in the sequence shown below on the Support telephone menu:

1. Select **2** for New Service Request.
2. Select **3** for Hardware, Networking and Solaris Operating System Support.
3. Select one of the following options:
   - For Technical issues such as creating a new Service Request (SR), select **1**.
   - For Non-technical issues such as registration or assistance with My Oracle Support, select **2**.

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

Acronyms

The following table lists the acronyms and the terminologies used in the document:

Table Acronyms and Terminologies

Acronym	Description
5GC	5th Generation Core
ACME	Automatic Certificate Management Environment
BSF	Binding Support Function
CNE	Cloud Native Environment
CNCC	Cloud Native Configuration Console
cnDRA	Cloud Native Diameter Routing Agent
DNSSEC	Domain Name System Security Extensions
DNS	Domain Name System
ETCD	The name “etcd” originated from two ideas, the unix “/etc” folder and “d"istributed systems. The “/etc” folder is a place to store configuration data for a single system whereas etcd stores configuration information for large scale distributed systems. Hence, a “d"istributed “/etc” is “etcd”.
FQDN	fully qualified domain name
GPG	Gnu Privacy Guard
iLO	Integrated Lights Out
Kyverno	Kyverno is a policy engine designed for Kubernetes.
mTLS	Mutual Transport Layer Security
NAPTR	Name Authority Pointer
NF	Network Function. A functional building block within a network infrastructure, which has well defined external interfaces and well defined functional behavior. In practical terms, a network function is often a network node or physical appliance.
NRF	Network Repository Function
NSSF	Network Slice Selection Function
CNE	Oracle Communications Cloud Native Core, Cloud Native Environment
OCIR	Oracle Cloud Infrastructure Registry
OCCM	Oracle Communications Certificate Management
OSSA	Oracle Software Security Assurance
OWASP	Open Web Application Security Project
PCF	Policy Control Function
PKI	Public Key Infrastructure
SCP	Service Communication Proxy
SEPP	Security Edge Protection Proxy
ToR	Top-of-Rack Switching
UDR	Unified Data Repository
UDSF	Unstructured Data Storage Function
YUM	Yellow Dog Updater, Modified is an open-source Linux package management application.

What's New in This Guide

This section introduces the documentation updates for Release 24.3.x in Oracle Communications Cloud Native Core, Security Guide.

Release 24.3.0 - G15056-01, October 2024

- Renamed the section title from "Cloud Native Core Ingress and Egress Gateways Specific Security Recommendations and Guidelines" to "Cloud Native Core Gateway Services Specific Security Recommendations and Guidelines".
- Added the following sections to configure Access Token and create different database users:
  - Gateway Services Access Token Secret Configuration
  - Gateway Services Access Token Secret Update
  - Gateway Services MySQL Secret Configuration
  - Kubernetes Secret Creation for Gateway Services Privileged Database Users
  - Kubernetes Secret Update for Gateway Services Privileged Database Users
  - Kubernetes Secret Creation for Gateway Services Application Database User
  - Kubernetes Secret Update for the Gateway Services Application Database Users
  - Network Policies
- Updated the YUM procedures in the Cloud Native Environment (CNE) Specific Security Recommendations and Guidelines section.

1 Introduction

The Security Guide provides an overview of the security related information applicable to Cloud Native Core (CNC) Network Functions, CNE, Console, and cnDBTier. In case there are specific aspects for the underlying scenarios or applications, these are described in NF specific chapters. This document contains recommendations (short statements on operating and managing the CNC software) and procedures (step-by-step instructions to assist the customer in tailoring or hardening the CNC system).

Install CNC as "secure by default" wherever possible. In a few cases where this isn't possible, an installation checklist procedure is created and listed on the Cloud Native Core Security Checklist. It is a short list of post-installation hardening activities that the customer must perform before placing the system into operation. The recommendations and other procedures found in this document are optional and must be considered in the context of your organization's approved security policies.

This security guide also provides a simplified trust model for the system.

1.1 Audience

- Technology consultants
- Installers
- Security consultants
- System administrators

1.2 References

The following references provide additional background on product operations and support:

- Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Configuration Console Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Network Slice Selection Function Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Service Communication Proxy Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Policy Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Unified Data Repository Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Network Repository Function Installation, Upgrade, and Fault Recovery Guide
- Oracle Communications Cloud Native Core, Security Edge Protection Proxy Installation, Upgrade, and Fault Recovery Guide

2 Overview

Deployment Environment

The 5G Cloud Native Core provides a variety of possible configuration and deployment environments:

Table 2-1 Deployment Environment

Type	Host	CNE	Description
Bare Metal	CNE-Supported Infrastructure	CNE	In this environment, a Kubernetes Cloud Native Environment is hosted directly on the bare metal hardware, while some other elements (DB or Bastion) are hosted using virtualized servers.
vCNE (Virtualized OC-CNE)	Customer Hypervisor (KVM/VMware ESXi)	OC-CNE	In this environment, all the system elements are hosted in virtualized servers deployed on a customer provided Openstack environment. The CNE is deployed on the openstack infrastructure.
Cloud	Customer CNE	Customer CNE	In this environment, the customer provides the CNE and deploys the 5G NFs directly into the environment. The Oracle provided common services and cnDBTier are used; equivalent functionality is provided by the customer.

Note

- Oracle Communications CNE provides basic CNE environment for on-premise deployment.
- Customer CNE provides CNE environment for running not just 5G microservices but also any kind of service, not just 5G. For example- observability frameworks or 4G microservices.
- With Customer CNE, a customer is responsible for ensuring the security of a Customer CNE.

Note

The cloud environment security recommendations and procedures focus on the CNE reference environment. Customers providing their own CNE must have security procedures already in place.

2.1 Cloud Native Core Network Functions

Network Function security is prescribed by the relevant 4G and 5G standards. This document details the administrative steps required to ensure secure 5G network operations.

Table 2-2 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
Network Repository Function	NRF	NRF provides registration, discovery, and authorization services to all the Network Functions (NF) in the 5G core network.	• 3GPP TS 29.510 v15.5 • 3GPP TS 29.510 v16.3.0 • 3GPP TS 29.510 v16.7
Service Communication Proxy	SCP	SCP provides a 5G-aware service mesh.	• 3GPP TS 29.500 R16 v16.6.0
Network Slice Selection Function	NSSF	NSSF works with the Access and Mobility Function (AMF) to select the network slice to be used by the User Equipment (UE).	• 3GPP TS 29.531 v15.5.0 • 3GPP TS 29.531 v16.5.0 • 3GPP TS 29.531 v16.8.0 • 3GPP TS 29.501 v16.10.0 • 3GPP TS 29.502 v16.10.0
Security Edge Protection Proxy	SEPP	In the roaming architecture, the home and visited networks are connected via the Security Edge Protection Proxy (SEPP) to manage the control plane of the inter-network interconnect.	• 3GPP TS 23.501 v17.6.0 • 3GPP TS 23.502 v17.6.0 • 3GPP TS 29.500 v17.8.0 • 3GPP TS 29.501 v17.7.0 • 3GPP TS 29.573 v17.6.0 • 3GPP TS 29.510 v17.7.0 • 3GPP TS 33.501 v17.7.0 • 3GPP TS 33.117 v17.1.0 • 3GPP TS 33.210 v17.1.0

Table 2-2 (Cont.) 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
Unified Data Repository	UDR/ UDSF	UDR is a repository of subscriber information, and is used by various NFs (including UDR, PCF, and NEF). The UDSF is a part of the Unified Data Management Function (UDF) and is used to store state information for Network Functions (NF).	• 3GPP TS 29.505 v15.4.0 • 3GPP TS 29.504 v16.2.0 • 3GPP TS 29.519 v16.2.0 • 3GPP TS 29.511 v17.2.0
Unified Data Repository (UDR) as Subscriber Location Function (SLF).	SLF	SLF supports the storage and retrieval of subscriber information through the nudr interface.	NA

Table 2-2 (Cont.) 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
Network Exposure Function	NEF	Securely exposes network capabilities and events to Application Functions (AF).	• 3GPP TS 29.338 v 17.1.0 • 3GPP TS 23.040 v 17.2.0 • 3GPP TS 29.122 v 16.10.0 , 17.10.0 • 3GPP TS 23.222 v 16.9.0 • 3GPP TS 23.501 v 16.10.0 • 3GPP TS 23.502 v 16.10.0 • 3GPP TS 29.514 v 16.10.0 • 3GPP TS 29.521 v 16.10 • 3GPP TS 29.503 v 16.14.0 • 3GPP TS 29.515 v 16.7 • 3GPP TS 29.222 v 16.5.0 • 3GPP TS 29.500 v 16.6.0 • 3GPP TS 29.501 v 16.6.0 • 3GPP TS 29.522 v 16.10.0, 17.10.0 • 3GPP TS 29.510 v 16.6.0 • 3GPP TS 29.591 v 16.3.0 • 3GPP TS 29.518 v 16.14.0

Table 2-2 (Cont.) 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
			• 3GPP TS 33.501 v 17.7.0 • 3GPP TS 29.504 v 16.10.0 • 3GPP TS 29.519 v 16.11.0 • 3GPP TS 29.508 v 16.11.0 • 3GPP TS 23.682 v 16.9.0 • 3GPP TS 29.337 v 16.1.0 • 3GPP TS 29.214 v 16.7.0 • 3GPP TS 32.291 v16.14 • 3GPP TS 32.290 v16.10.0 • 3GPP TS 32.254 v16.6.0

Table 2-2 (Cont.) 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
Policy Control Function	PCF	Implements a unified policy framework for implementing control plane rules.	• 3GPP TS 23.501 • 3GPP TS 23.502 • 3GPP TS 23.503 • 3GPP TS 29.500 • 3GPP TS 29.504 • 3GPP TS 29.510 • 3GPP TS 29.507 • 3GPP TS 29.512 • 3GPP TS 29.513 • 3GPP TS 29.514 • 3GPP TS 29.519 • 3GPP TS 29.521 • 3GPP TS 29.525 • 3GPP TS 29.594 • 3GPP TS 29.214
Binding Support Function	BSF	Provides PCF binding (mapping and selection) for User Equipment (UE).	• 3GPP TS 23.501 v17.7.0 • 3GPP TS 23.502 v17.7 • 3GPP TS 23.503 V17.7 • 3GPP TS 29.500 v17.7.0 • 3GPP TS 29.510 v17.7 • 3GPP TS 29.513 V17.7 • 3GPP TS 29.521 v17.7.0 • 3GPP TS 33.501 V17.7.0

Table 2-2 (Cont.) 4G and 5G Network Functions

Network Functions	Abbreviation	Description	3GPP Standard
Oracle Communications Cloud Native Core, Certificate Management	OCCM	Supports automation of certificate lifecycle management.	• 3GPP TS 33.310-h30 • 3GPP TR 33.876 v.0.3.0
Cloud Native Core Console	CNC C	Configuration and Operations portal and proxy for CNC NFs and components.	NA
Oracle Communications DBTier	cnDBTier	Containerized deployment and automation of MySQL Cluster database technology	NA

Table 2-3 5G Common Services

Network Function	Abbreviation	Description
Ingress Gateway, Egress Gateway, Alternative Routing Service	APIGW	Ingress and Egress Gateways for NFs
Automatic Test Suite	ATS	Automated Test Suite
Oracle Communications Certificate Manager	OCCM	Certificate Management
Network Repository Function - Client	NRF-Client	Product: Oracle Communications Cloud Native Core - 5G and Subcomponent: NRF-Client
Mediation	Mediation	Mediation modifies 5G Service Based Interface (SBI) message content, which includes HTTP2 header values and JSON message body, based on the user-defined mediation rule sets
Oracle Communication Cloud Native Core OCI Adapter	OCI Adaptor	Oracle Communications Cloud Native Core OCI Adapter

2.2 Secure Development Practices

Given below are the practices followed for a secure development environment:

2.2.1 Vulnerability Handling

For details about the vulnerability handling, refer Oracle Critical Patch Update Program. The primary mechanism to backport fixes for security vulnerabilities in Oracle products is quarterly Critical Patch Update (CPU) program.

In general, the CNC Software is on a quarterly release cycle, with each release providing feature updates and fixes and updates to relevant third party software. These quarterly releases provide cumulative patch updates.

You should have procedures in place to deploy security updates for each release cycle. For more information, see Implementing Security Recommendations and Guidelines.

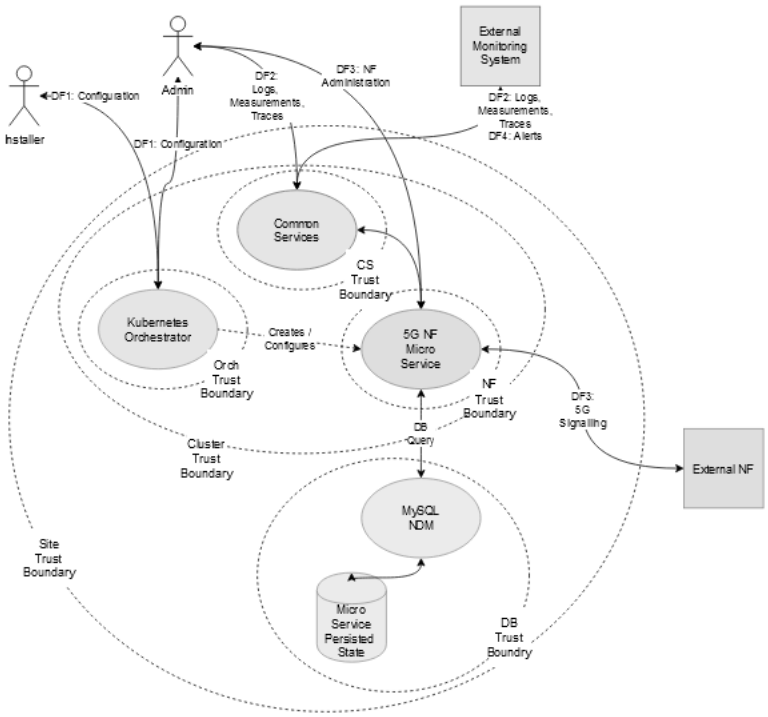
2.3 Trust Model

The following Trust Model depicts the reference trust model (regardless of the target environment). The model describes the critical access points and controls site deployment.

While the model shows a single 5G NF microservice deployed, several NFs are also deployed in individual clusters.

2.3.1 Context Diagram

Figure 2-1 Context Diagram



2.3.2 Key Trust Boundaries

Trust Boundaries identify areas at a similar level of trust and isolate them from other areas at a different level of trust. Following are the key trust boundaries:

Table 2-4 Key Trust Boundaries

Trust Boundary	Includes	Access Control
Site Trust Boundary	All the NFs and other supporting elements for a given site.	Cluster Access Policies are implemented using some kind of Access Control Group (or Security Group) mechanism.

Table 2-4 (Cont.) Key Trust Boundaries

Trust Boundary	Includes	Access Control
Cluster Trust Boundary	All the compute elements for a given cluster	Network Policies control traffic ingress and egress. Pod Security Policies manage the workloads allowed in the cluster (For example, no pods requiring privilege escalation).
DB Trust Boundary	All the cnDBTier elements for a given cluster	Firewall Policies control traffic ingress and egress. DB grants access and other permission mechanisms that provide authorization for users.
Orchestrator Trust Boundary (Orch Trust Boundary)	The orchestration interface and keys	Firewall Policies control the access to a Bastion server which provides orchestration services. Access to the Bastion host uses Secure Socket Shell (SSH) protocol. The cluster orchestration keys are stored on the Bastion host.
CS Trust Boundary	The common services implementing logging, tracing, and measurements.	Each of the common services provides independent Graphical User Interfaces (GUIs) that are currently open. The customer may want to introduce an api-gateway, implement authentication and authorization mechanisms to protect the OAM (Operations, Administrations, and Maintenance) data. The common services can be configured to use Transport Layer Security (TLS). When TLS is used, certificates must be generated and deployed through the orchestrator.
NF Trust Boundaries	A collection of 5G Network Functions deployed as a service.	5G NF microservices provide Signaling access through a TLS protected HTTP2 interface. The certificates for these interfaces are managed via the certificate manager.

2.3.3 External Data Flows

The following are external data flows:

Table 2-5 External Data Flows

Data Flow	Protocol	Description
DF1: Configuration	SSH	The installer or administrator accesses the orchestration system hosted on the Bastion Server. The installer or administrator must use ssh keys to access the bastion to a special orchestration account (not root). Password access is not allowed.
DF2: Logs, Measurements, Traces	HTTP/HTTPS	The administrator or operator interacts with the common services using web interfaces.
DF3: 5G Signaling	HTTP2 (w/TLS)	All signaling interaction between NFs at a site and NFs at an external site is sent through TLS protected HTTP2.
DF4: Alerts	SNMP (Trap)	Alerting is performed using SNMP traps.

The complete list of network flows including service types and ports are available in Port Flow Appendix.

3

Implementing Security Recommendations and Guidelines

3.1 Common Security Recommendations and Guidelines

This section provides details of the common security recommendations and guidelines, irrespective of the NFs.

3.1.1 4G and 5G Application Authentication and Authorization

Mutual Transport Layer Security (mTLS) is a type of authentication in which the two parties in a connection authenticate each other using the TLS protocol. mTLS ensures that the traffic is secure and trusted in both directions between a client and server. Cloud Native Core NFs support integration with platform service meshes and Mutual Transport Layer Security (mTLS) may be provided by the platform service mesh, thereby securing communication flows between all applications that participate in the platform service mesh. mTLS also encrypts the data flows so that only the endpoints of the mTLS session can access the contents of the communication data.

4G and 5G NFs use Mutual Transport Layer Security (mTLS) authentication to secure communication. All NFs require to establish a trust relationship with all peers by exchanging and trusting peer root or intermediate certificates. The peer certificates must be available in the truststore (K8s Secrets) to establish secure communication.

4G and 5G NFs also support manual importation and a semiautomatic import using the cert-manager external provider.

3.1.2 cnDBTier Authentication and Authorization

The cnDBTier provides a highly available multisite database that stores NF state and configuration. When installed, the MySQL DB is configured with a root account whose password is randomly generated. Each NF must have additional accounts for that particular NF.

The procedures in this section explain the following:

- Changing cnDBTier Passwords

> **Note**
>
> The roles of DB Administrator and Kubernetes cluster administrator must be kept separate. The DB Administrator must be responsible for securing and maintaining the cnDBTier MySQL NDB cluster. The Kubernetes cluster administrator must be responsible for securing and operating the Bastion Host and Kubernetes Cluster. When 5G NFs are installed, the DB Administrator is required to create new NF database and NF DB accounts (using the DB Root credentials). Once this is completed, the Kubernetes cluster administrator installs the NF.

3.1.3 Cloud Native Core Gateway Services Specific Security Recommendations and Guidelines

This section provides Ingress and Egress Gateways specific security recommendations and guidelines. Security recommendations common to all 5G and 4G are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- Enabling TLS and Ciphers in Ingress/Egress Gateway
- Certificate Management and Dynamic reload of certificates in Gateways
- Gateway Services Access Token Secret Configuration
- Gateway Services Access Token Secret Update
- Gateway Services MySQL Secret Configuration
  - Kubernetes Secret Creation for Gateway Services Privileged Database Users
  - Kubernetes Secret Update for Gateway Services Privileged Database Users
  - Kubernetes Secret Creation for Gateway Services Application Database User
  - Kubernetes Secret Update for the Gateway Services Application Database Users
- Network Policies

Enabling TLS and Ciphers in Ingress and Egress Gateway

Use the following procedure to enable TLS and Ciphers:

1. Helm configuration to enable TLS:
   To open HTTPS port in Ingress Gateway, set the `enableIncomingHttps` parameter to true.

   To configure the HTTPS client in Ingress Gateway, set the `enableOutgoingHttps` parameter to true.
2. Create the following files:
   a. RSA or ECDSA Private key (Example: `rsa_private_key_pkcs1.pem`)

b. Truststore password (Example:`trust.txt`)

c. Key store password (Example: `key.txt`)

d. Certificate chain for truststore (Example: `caroot.cer`)

e. Signed server certificate (Example: `ocingress.cer`) or Signed client certificate (Example: `ocegress.cer`)

> **Note**
>
> Creation of private keys, certificates, and passwords is at the discretion of user.

3. Run the following command to create secret:

```
$ kubectl create secret generic ocingress-secret --from-
file=rsa_private_key_pkcs1.pem
--from-file=trust.txt --from-file=key.txt --from-file=ocingress.cer --from-
file=caroot.cer -n ocingress
```

4. Enable the cipher suites:

   - Cipher Suites to be enabled on Server side (Ingress Gateway).
   - Cipher Suites to be enabled on Client side (Egress Gateway).

   Cipher Suites:

```
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_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
```

Certificate Management and Dynamic Reload of Certificates in Gateway Services

Whenever certificates get compromised or a new certificate chain is required to be added to the truststore, you can update the key and truststore used by the application.

To update the key and the truststore, update or replace the secret:

```
$ kubectl create secret generic ocingress-secret --from-
file=rsa_private_key_pkcs1.pem
 --from-file=trust.txt --from-file=key.txt --from-file=tmp.cer --from-
file=caroot.cer --dry-run -o yaml
 -n ocingress| kubectl replace -f - -n ocingress
```

Whenever there is an update in the certificate chain or signed certificate placed in secret, Kubernetes watcher implemented in update container checks for a change in file state and replaces the key and truststore accordingly in the mounted shared volume.

Dynamic reload of certificates is not supported in Ingress Gateway as of now, so a manual restart of pod is required when any update in the configuration is made with respect to https.

In case of Egress Gateway, update container will trigger the rest endpoint to reload key and truststore dynamically. Then Egress Gateway will pickup new store files from shared volume and reload trust and key managers. Egress Gateway will use the replaced store to establish new connections and gracefully terminate existing connections by sending a GOAWAY frame.

Gateway Services Access Token Secret Configuration

Use the following procedure to create an Access token secret:

1. Create the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for Gateway Services
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for Gateway Services

   > **Note**
   >
   > Creation of private keys, certificates and passwords are at the discretion of user.

2. Log in to Bastion Host or server from where you can run kubectl commands.
3. Create a namespace for the secret by performing the following procedure:
   a. Verify if the required namespace already exists in the system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:

      > **Note**
      >
      > This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      Example:

      ```
      $ kubectl create namespace ocegress
      ```

4. Create Kubernetes secret for the Access token by performing the following steps:
   a. To create Kubernetes secret for HTTPS, the following files are required:
      - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for Gateway Services
      - RSA private keys for algorithm RS256 and corresponding valid public certificates for Gateway Services

      > **Note**
      >
      > Creation process for private keys, certificates and passwords is at the user's or operators discretion. Unencrypted key and certificates is only supported. PKCS1 and PKCS8 are the only supported versions for RSA. PKCS8 is the only supported version for ECDSA.

b. Run the following command to create secret. The names used below are the same as provided in the `custom_values.yaml` file in Gateway Services deployment:

```
$ kubectl create secret generic <ocegressaccesstoken-secret-name>  --
from-file=<ecdsa_private_key.pem>
 --from-file=<rsa_private_key.pem>  --from-file=<ssl_truststore.txt> --
from-file=<keystore_password.txt>
  --from-file=rsa_certificate.crt --from-file=<ecdsa_certificate.crt> -
n <Namespace of ocegress AccessToken secret>
```

> **Note**
>
> Note down the command used during the creation of Kubernetes secret. This command will be used for future references.

```
$ kubectl create secret generic ocegressaccesstoken-secret  --from-
file=ecdsa_private_key.pem
--from-file=rsa_private_key.pem  --from-file=ssl_truststore.txt --from-
file=keystore_password.txt  --from-file=
rsa_certificate.crt --from-file=ecdsa_certificate.crt -n ocegress
```

c. Run the following command to verify if the secret is created:

```
$ kubectl describe secret <ocegressaccesstoken-secret-name> -n
<Namespace of Gateway Services AccessToken secret>
```

Example:

```
 $ kubectl describe secret ocegressaccesstoken-secret -n ocegress
```

Gateway Services Access Token Secret Update

Use the following procedure to update the Access token secret:

1. Update the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for Gateway Services
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for Gateway Services

   > **Note**
   >
   > Update of private keys, certificates and passwords are at the discretion of user.

2. Log in to Bastion Host or server from where you can run `kubectl` commands.
3. Update the secret with new or updated details by performing the following procedure:
   a. Copy the exact command used in above section during creation of secret.
   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of Access Token secret>".

c. Create secret command must look like:

```
$ kubectl create secret generic <ocegressaccesstoken-secret> --from-
file=<ecdsa_private_key.pem>
--from-file=<rsa_private_key.pem> --from-file=<rsa_certificate.crt> --
from-file=<ecdsa_certificate.crt>
 --dry-run -o yaml -n <Namespace of ocegress deployment> | kubectl
replace -f - -n <Namespace of ocegress deployment>
```

Example: The names used below are the same as provided in the `custom_values.yaml` in Gateway Services deployment:

```
$ kubectl create secret generic ocegressaccesstoken-secret --from-
file=ecdsa_private_key.pem
--from-file=rsa_private_key.pem --from-file=rsa_certificate.crt --from-
file=ecdsa_certificate.crt
--dry-run -o yaml -n ocegress | kubectl replace -f - -n ocegress
```

d. Run the updated command.

e. After successful secret update, the following message is displayed:

```
secret/<ocegressaccesstoken-secret> replaced
```

Gateway Services MySQL Secret Configuration

This section describes the secret creation for following types of Gateway Services users. Different users have different sets of permissions.

- Gateway Services privileged user: This user category has a complete set of permissions. The user can perform DDL and DML operations to install, upgrade, roll back or delete operations.
- Gateway Services application user: This user category has fewer sets of permissions and is used by Gateway Services applications during service operations handling. This user cannot create, alter, and drop the database and tables.

Kubernetes Secret Creation for Gateway Services Privileged Database Users

This section provides procedures to create Kubernetes secrets for accessing the Gateway Services database for the privileged user.

1. Log in to Bastion Host or server from where you can run the `kubectl` commands.
2. Create a namespace for the secret by performing the following procedure:
   a. Verify if the required namespace already exists in the system:

```
$ kubectl get namespaces
```

   b. In the output of the above command, check if the required namespace is available. If not available, create the namespace using the following command:

> **Note**
>
> This is an optional step. In case the required namespace already exists, proceed with the next set of procedures.

```
$ kubectl create namespace <required namespace>
```

For example:

```
$ kubectl create namespace ocegress
```

3. Create a Kubernetes secret for privileged users as follows:
   a. Create a Kubernetes secret for MySQL:

```
 $ kubectl create secret generic <privileged user secret name>
 --from-literal=dbUsername=<Gateway Services Privileged MySQL database
username>
--from-literal=dbPassword=<Gateway Services Privileged MySQL User
database password>
--from-literal=appDbName=<Gateway Services MySQL database name>
--from-literal=networkScopedDbName=<Gateway Services MySQL Network
database name>
--from-literal=commonConfigDbName=<Gateway Services MySQL Common
Configuration DB> -n
<Namespace of Gateway Services deployment>
```

> **Note**
>
> Note down the command used during the creation of the Kubernetes secret. This command is used for future references.

Example:

```
$ kubectl create secret generic privilegeduser-secret --from-
literal=dbUsername=ocegressPrivilegedUsr
--from-literal=dbPassword=ocegressPrivilegedPasswd --from-
literal=appDbName=ocegressApplicationDB --from-literal
=networkScopedDbName=ocegressNetworkDB --from-
literal=commonConfigDbName=commonConfigurationDB -n ocegress
```

   b. Verify the secret created using above command:

```
$ kubectl describe secret <database secret name> -n <Namespace of
Gateway Services deployment>
```

Example:

```
$ kubectl describe secret privilegeduser-secret -n ocegress
```

Kubernetes Secret Update for Gateway Services Privileged Database Users

This section provides procedures to update Kubernetes secrets for accessing the Gateway Services database for the privileged user.

1. Log in to Bastion Host or server from where you can run the `kubectl` commands.
2. Update Kubernetes secret for privileged user as follows:
   a. Copy the exact command used in section during creation of secret:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<Gateway Services Privileged MySQL database
username>
--from-literal=dbPassword=<Gateway Services Privileged MySQL database
password>
--from-literal=appDbName=<Gateway Services MySQL database name>
--from-literal=networkScopedDbName=<Gateway Services MySQL Network
database name>
--from-literal=commonConfigDbName=<Gateway Services MySQL Common
Configuration DB> -n
<Namespace of Gateway Services deployment>
```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<Gateway Services Privileged MySQL database
username>
--from-literal=dbPassword=<Gateway Services Privileged MySQL database
password>
--from-literal=appDbName=<Gateway Services MySQL database name>
--from-literal=networkScopedDbName=<Gateway Services MySQL Network
database name>
--from-literal=commonConfigDbName=<Gateway Services MySQL Common
Configuration DB> --dry-run -o yaml
-n <Namespace of Gateway Services deployment> | kubectl replace -f - -n
<Namespace of Gateway Services deployment>
```

   c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Kubernetes Secret Creation for Gateway Services Application Database User

This section provides procedures to create Kubernetes secrets for accessing the Gateway Services database for the application database user.

1. Log in to Bastion Host or server from where you can run the `kubectl` commands.
2. Create a namespace for the secret by performing the following procedure:

a. Verify if the required namespace already exists in the system:

```
$ kubectl get namespaces
```

b. In the output of the above command, check if required the namespace is available. If not available, create the namespace using the following command:

> **Note**
>
> This is an optional step. In case the required namespace already exists, proceed with the next set of procedures.

```
$ kubectl create namespace <required namespace>
```

Example:

```
$ kubectl create namespace ocegress
```

3. Create a Kubernetes secret for the Gateway Services application database user for configuring records as follows:

a. Create a Kubernetes secret for the Gateway Services application database user:

```
$ kubectl create secret generic <appuser-secret name> --from-
literal=dbUsername=<Gateway Services APPLICATION User Name> --from-
literal=dbPassword=<Password for Gateway Services APPLICATION User> --
from-literal=appDbName=<Gateway Services Application Database> -n
<Namespace of Gateway Services deployment>
```

> **Note**
>
> Note down the command used during the creation of Kubernetes secret. This command will be used for future references.

Example:

```
$ kubectl create secret generic appuser-secret --from-
literal=dbUsername=GatewayServicesApplicationUsr --from-
literal=dbPassword=GatewayServicesApplicationPasswd --from-
literal=appDbName=GatewayServicesApplicationDB -n ocegress
```

b. Verify the secret creation:

```
$ kubectl describe secret <appuser-secret name> -n <Namespace of
Gateway Services deployment>
```

Example:

```
$ kubectl describe secret appuser-secret -n ocegress
```

Kubernetes Secret Update for the Gateway Services Application Database Users

This section provides procedures to update Kubernetes secrets for accessing the Gateway Services database for the application database user.

1. Log in to Bastion Host or server from where you can run the `kubectl` commands.
2. This section explains how you can update the Kubernetes secret.
   a. Copy the exact command used in above section during creation of secret:

      ```
      $ kubectl create secret generic <appuser-secret name> --from-
      literal=dbUsername=<Gateway Services APPLICATION
       User Name> --from-literal=dbPassword=<Password for Gateway Services
      APPLICATION User> --from-literal=appDbName=<Gateway Services
      Application Database> -n <Namespace of Gateway Services deployment>
      ```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

      ```
      $ kubectl create secret generic <database secret name> --from-
      literal=dbUsername=<Gateway Services APPLICATION
      User Name> --from-literal=dbPassword=<Password for Gateway Services
      APPLICATION User> --from-literal=appDbName=<Gateway Services
      Application Database> --dry-run -o yaml -n <Namespace of Gateway
      Services deployment> | kubectl replace -f - -n <Namespace
      of Gateway Services deployment>
      ```

   c. Run the updated command. The following message is displayed:

      ```
      secret/<database secret name> replaced
      ```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

3.1.4 Automated Test Suite (ATS) Specific Security Recommendations and Guidelines

This section provides Automated Test Suite (ATS) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines Section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- Enabling TLS in ingress and egress gateways and selection of CIPHERS
- Enabling TLS in ATS Pod

Enabling TLS in ingress and egress gateways and selection of CIPHERS

For Enabling TLS in ingress and egress gateways and selection of CIPHERS, see Cloud Native Core Ingress/Egress Gateways Specific Security Recommendations and Guidelines.

Enabling TLS in ATS Pod

> **Note**
>
> - This procedure takes PCF as an example of procedure to be followed for TLS enabled server side. It applies to all TLS enabled server side NFs.
> - This procedure is run after successful deployment of TLS enabled server side (for example, PCF) and ATS.

Use the following procedure to enable TLS in ATS Pod:

1. Run the following command to copy the caroot.cer generated while PCF deployment to ATS pod in "cert" directory:

   ```
   kubectl cp  <path_to_file>/caroot.cer <namespace>/<ATS-Pod-name>: /var/lib/
   jenkins/cert/ -n <namespace>
   ```

   Example:

   ```
    kubectl cp cert/caroot.cer ocpcf/ocpcf-ocats-pcf-56754b9568-
   rkj8z:/var/lib/jenkins/cert/
   ```

2. Run the following command to log in to your ATS Pod:

   ```
   kubectl exec -it <ATS-Pod-name> bash -n <namespace>
   ```

3. Run the following commands from cert directory to create private key and certificates:

   a. `openssl req -x509 -nodes -sha256 -days 365 -newkey rsa:2048 -keyout rsa_private_key_client -out rsa_certificate_client.crt`

> **Note**
>
> User has to specify fqdn of PCF Ingress Gateway service, that is, <ingress-servicename>.<pcf_namespace>.svc in Common Name

b. `openssl rsa -in rsa_private_key_client -outform PEM -out rsa_private_key_pkcs1_client.pem`

c. `openssl rsa -in rsa_private_key_client -outform PEM -out rsa_private_key_pkcs1_client.pem`

> **Note**
>
> ssl.conf which was used while deploying PCF can be used or copied to ATS pod for this step.

4. Copy the `ocegress_client.csr` to the bastion and run the following command from the Bastion:

```
openssl x509 -CA caroot.cer -CAkey cakey.pem -CAserial serial.txt -req -in
ocegress_client.csr -out ocegress_client.cer -days 365 -extfile ssl.conf -
extensions req_ext
```

5. Copy the `ocegress_client.cer` from the Bastion to the ATS Pod.
6. Restart the ingress and egress gateway pods from the Bastion.
7. Run the following command to unset the http and https proxy in the ATS pod:

```
unset http_proxy
unset https_proxy
```

8. Run the following CURL command from the ATS pod **cert** directory:

> **Note**
>
> Update the Ingress-gateway-service-name and HTTPS port before running the command.

```
curl -X POST -v "https://<Ingress-gateway-service-
name.namespace.svc>:<HTTPS-port>/npcf-smpolicycontrol/v1/sm-policies"
 --cacert caroot.cer --cert ./ocegress_client.cer --key
rsa_private_key_client -H "Content-Type: application/json" -d
'{ "3gppPsDataOffStatus": true, "accNetChId": { "accNetChaIdValue":
"01020304", "sessionChScope": true }, "accessType":
"3GPP_ACCESS", "dnn": "dnn1", "gpsi": "9192503899", "ipv4Address":
"192.168.10.10", "ipv6AddressPrefix": "2001:1:22:3286::/64",
 "notificationUri": "http://smf-simulator:8080/smf/notify", "offline":
true, "online": false, "pduSessionId": 1, "pduSessionType":
"IPV4", "pei": "imei-100120210000001", "ratType": "EUTRA",
```

```
"servingNetwork": { "mcc": "450", "mnc": "08" }, "sliceInfo":
{ "sd": "abc123", "sst": 11 }, "smPoliciesUpdateNotificationUrl": "npcf-
smpolicycontrol/v1/sm-policies/{ueId}/notify",
"subSessAmbr": { "downlink": "1000000 Kbps", "uplink": "10000 Kbps" },
"subscribedDefaultQosInformation": "FFS", "supi":
"imsi-030003000000053", "supportedFeatures": "", "ueTimeZone": "+08:00",
"userLocationInformation": { "nrLocation": { "ncgi":
{ "nrCellId": "512", "plmnId": { "mcc": "450", "mnc": "08" } }, "tai":
{ "plmnId": { "mcc": "450", "mnc": "08" }, "tac": "1801" } } } }'
```

The output of the command will be:

```
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 10.75.233.76:31940...
* TCP_NODELAY set
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:--
--:--:--     0* Connected to 10.75.233.76 (10.75.233.76) port 31940 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: caroot.cer
  CApath: C:/Users/prup
} [5 bytes data]
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
} [512 bytes data]
* TLSv1.3 (IN), TLS handshake, Server hello (2):
{ [94 bytes data]
* TLSv1.2 (IN), TLS handshake, Certificate (11):
{ [924 bytes data]
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
{ [300 bytes data]
* TLSv1.2 (IN), TLS handshake, Request CERT (13):
{ [205 bytes data]
* TLSv1.2 (IN), TLS handshake, Server finished (14):
{ [4 bytes data]
* TLSv1.2 (OUT), TLS handshake, Certificate (11):
} [1866 bytes data]
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
} [37 bytes data]
* TLSv1.2 (OUT), TLS handshake, CERT verify (15):
} [264 bytes data]
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
} [1 bytes data]
* TLSv1.2 (OUT), TLS handshake, Finished (20):
} [16 bytes data]
* TLSv1.2 (IN), TLS handshake, Finished (20):
{ [16 bytes data]
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=IN; ST=Karnataka; L=Bangalore; O=Oracle; CN=10.75.233.76
```

```
*  start date: Mar 22 18:24:24 2020 GMT
*  expire date: Mar 22 18:24:24 2021 GMT
*  subjectAltName: host "10.75.233.76" matched cert's IP address!
*  issuer: C=IN; ST=Karnataka; L=Bangalore; O=Oracle; OU=CGBU; CN=pcfperf-
bastion-1; emailAddress=pruthvi.p@oracle.com
*  SSL certificate verify ok.
  0     0    0     0    0     0      0      0 --:--:--  0:00:01
--:--:--     0* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade:
len=0
} [5 bytes data]
* Using Stream ID: 1 (easy handle 0x671fe0)
} [5 bytes data]
> POST /npcf-smpolicycontrol/v1/sm-policies HTTP/2
> Host: 10.75.233.76:31940
> user-agent: curl/7.68.0
> accept: */*
> content-type: application/json
> content-length: 999
>
{ [5 bytes data]
* Connection state changed (MAX_CONCURRENT_STREAMS == 4294967295)!
} [5 bytes data]
* We are completely uploaded and fine
{ [5 bytes data]
< HTTP/2 201
< location: https://vzw-pcf-ingress-gateway.svc:443/npcf-
smpolicycontrol/v1/sm-policies/46c38683-d138-4691-a9b9-21557a50703a
< content-type: application/json
< date: Sun, 22 Mar 2020 18:28:02 GMT
< cache-control: no-cache, no-store, max-age=0, must-revalidate
< pragma: no-cache
< expires: 0
< x-content-type-options: nosniff
< x-frame-options: DENY
< x-xss-protection: 1 ; mode=block
< referrer-policy: no-referrer
<
100   999    0     0  100   999      0    434  0:00:02  0:00:02 --:--:--
434{ [5 bytes data]
100  1612    0   613  100   999    229    373  0:00:02  0:00:02 --:--:--
603{"sessRules":{"0_1":{"authDefQos":{"5qi":9,"arp":
{"priorityLevel":1,"preemptCap":"MAY_PREEMPT","preemptVuln":"NOT_PREEMPTABL
E"}},"sessRuleId":"0_1"}},"pccRules":{"0_0":{"flowInfos":
[{"flowDescription":"permit in ip from any to
any","flowDirection":"UPLINK"},{"flowDescription":"permit out ip from any
to
any","flowDirection":"DOWNLINK"}],"pccRuleId":"0_0","precedence":3000,"refQ
osData":["qosdata_0"]}},"qosDecs":{"qosdata_0":
{"5qi":9,"qosId":"qosdata_0","arp":
{"priorityLevel":1,"preemptCap":"MAY_PREEMPT","preemptVuln":"NOT_PREEMPTABL
E"}}},"policyCtrlReqTriggers":
["PLMN_CH","UE_IP_CH","DEF_QOS_CH","AC_TY_CH"]}
* Connection #0 to host 10.75.233.76 left intact
```

3.1.5 Oracle Communications Certificate Management (OCCM) Specific Security Recommendations and Guidelines

This section provides Oracle Communications Certificate Management (OCCM) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

Global Service Account Configuration

This section is optional and it describes how to manually create a service account, role, and rolebinding.

A custom service account can be provided for OCCM deployment in global.serviceAccountName of `occm_custom_values_<version>.yaml`.

A custom service account can be provided for helm in global.serviceAccountName:

```
global:
  dockerRegistry: cgbu-occncc-dev-docker.dockerhub-phx.oci.oraclecorp.com
  serviceAccountName: ""
```

Configuring Global Service Account to Manage NF Certificates with OCCM and NF in the Same Namespace

A sample OCCM Service account yaml file to create custom service account is as follows:

```
## Service account yaml file for occm-sa
apiVersion: v1
kind: ServiceAccount
metadata:
  name: occm-sa
  namespace: occm
  annotations: {}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: occm-role
  namespace: occm
rules:
- apiGroups:
  - "" # "" indicates the core API group
  resources:
  - services
```

```
  - configmaps
  - pods
  - secrets
  - endpoints
  verbs:
  - get
  - watch
  - list
  - create
  - delete
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: occm-rolebinding
  namespace: occm
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: occm-role
subjects:
- kind: ServiceAccount
  name: occm-sa
  namespace: occm
```

Configuring Global Service Account to Manage NF Certificates with OCCM and NF in Separate Namespaces

OCCM provides support for key and certificate management in multiple namespaces.

In this deployment model, OCCM is deployed in namespace different from the components' namespaces managed by it. It needs privileges to read, write, and delete Kubernetes secrets in the managed namespaces.

This is achieved by creating multiple namespace specific roles and binding them to the service account for OCCM.

- **AUTOMATIC Service Account Configuration:** Roles and role bindings are created for each namespace specified using the `occmAccessedNamespaces` field in `occm_custom_values.yaml`. A service account for OCCM is created automatically and the roles created are assigned using the corresponding role binding.
  Namespaces managed by OCCM service account:

  ```
  occmAccessedNamespaces:
    - ns1
    - ns2
  ```

  > **Note**
  >
  > Automatic Service Account Configuration is applicable for Single Namespace Management as well

- **Custom Service Account Configuration:** A custom service account can also be configured against the `serviceAccountName` field in occm_custom_values.yaml. If this is

provided, automatic service account creation doesn't get triggered. The `occmManagedNamespaces` field doesn't need to be configured.
A sample OCCM service account yaml file for creating a custom service account is as follows:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1
---
apiVersion: v1
kind: Namespace
metadata:
  name: ns2
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: occm-sa
  namespace: occm
  annotations: {}
---

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: ns1
  name: occm-secret-writer-role
rules:
- apiGroups:
  - "" # "" indicates the core API group
  resources:
  - secrets
  verbs:
  - get
  - watch
  - list
  - create
  - update
  - delete
---

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: ns2
  name: occm-secret-writer-role
rules:
- apiGroups:
  - "" # "" indicates the core API group
  resources:
  - secrets
  verbs:
  - get
  - watch
  - list
```

```
    - create
    - update
    - delete

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: occm-secret-writer-rolebinding
  namespace: ns1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: occm-secret-writer-role
subjects:
- kind: ServiceAccount
  name: occm-sa
  namespace: occm
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: occm-secret-writer-rolebinding
  namespace: ns2
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: occm-secret-writer-role
subjects:
- kind: ServiceAccount
  name: occm-sa
  namespace: occm
```

Helm Test Service Account Configuration

helmTestServiceAccountName is an optional field in the `occm_custom_values_<version>.yaml` file It should be added only if helm kubernetes resource is enabled. Custom service account can be provided for helm in `global.helmTestServiceAccountName::`

```
global:
  helmTestServiceAccountName: occm-helmtest-serviceaccount
```

A sample helm test service account yaml file is as follows:

```
helm test service account apiVersion: v1
kind: ServiceAccount
metadata:
  name: occm-helmtest-serviceaccount
  namespace: occm
  annotations: {}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
```

```
  name: occm-helmtest-role
  namespace: occm
rules:
- apiGroups:
  - "" # "" indicates the core API group
  resources:
  - services
  - configmaps
  - pods
  - secrets
  - endpoints
  - serviceaccounts
  verbs:
  - get
  - watch
  - list
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  verbs:
  - get
  - watch
  - list
  - update
- apiGroups:
  - apps
  resources:
  - deployments
  - statefulsets
  verbs:
  - get
  - watch
  - list
  - update
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - get
  - watch
  - list
  - update
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - roles
  - rolebindings
  verbs:
  - get
  - watch
  - list
  - update
- apiGroups:
  - monitoring.coreos.com
```

```
  resources:
  - prometheusrules
  verbs:
  - get
  - watch
  - list
  - update

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: occm-helmtest-rolebinding
  namespace: occm
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: occm-helmtest-role
subjects:
- kind: ServiceAccount
  name: occm-helmtest-serviceaccount
  namespace: occm
```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Certificate Management Installation, Upgrade, and Fault Recovery Guide.*

TLS configuration

OCCM does not directly interact with other NF microservices. The access is only through the CNC Console.

> **Note**
>
> - OCCM supports authentication of the CA generating the certificates using the CMPv2 MAC based and signing mechanism.
> - As an additional layer of security encryption of the traffic between OCCM and Certificate Authority using HTTPs is supported.
> - Configuration options are provided at REST API and helm deployment level. Refer to installation and user guide for details on configuration options.

For more information on TLS configuration, see CNC Console IAM LDAP Configuration in the Cloud Native Configuration Console (CNCC) Specific Security Recommendations and Guidelines section.

3.1.6 OCI Adaptor Specific Security Recommendations and Guidelines

This section provides OCI Adaptor specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> OCI Adaptor is deployed on OCI tenancy owned and managed by the customer. Therefore, customer is expected to develop an OCI security concept on their own.

OCI Adapter Registry Pull Secret (Automated)

While deploying OCI Adaptor, a registry pull secret must get created automatically (using Helm) and is used to pull OCI Adapter images from private Oracle Cloud Infrastructure Registry (OCIR).

Table 3-1 OCI Adaptor Secret

Secret Name	Secret Type	Secret Content
ocir-container-registry-secret	kubernetes.io/dockerconfigjson	**registry_name**: Must be provided by the user on OCI RM Stack UI. **registry_username**: Must be provided by the user on OCI RM Stack UI. **registry_password**: Must be provided by the user on OCI RM Stack UI.

3.1.7 Cloud Native Configuration Console (CNC Console) Specific Security Recommendations and Guidelines

This section provides Cloud Native Configuration Console (CNC Console) specific security recommendations and procedures. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines Section.

> **Note**
>
> kubectl commands might vary based on the platform deployment. Replace kubectl with Kubernetes environment-specific command line tool to configure Kubernetes resources through kube-api server. The instructions provided in this document are as per the Oracle Communications Cloud Native Environment (CNE) version of kube-api server.

> **Caution**
>
> User, computer and applications, and character encoding settings may cause an issue when copy-pasting commands or any content from PDF. PDF reader version also affects the copy-pasting functionality. It is recommended to verify the pasted content especially when the hyphens or any special characters are part of the copied content.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- CNC Console IAM MySQL Secret Configuration
- CNC Console IAM Default User (Admin) Secret Configuration
- OCI IAM Secret Configuration
- CNC Console IAM LDAPS Secret Configuration
- CNC Console IAM LDAP Configuration
- CNC Console TLS Secret configuration
- CNC Console Core Secret Configuration to Enable HTTPS
- CNC Console IAM SAML Configuration
- OCI IAM SAML Configuration
  - Adding a SAML Identity Provider in OCI IAM
  - JIT Configuration in OCI IAM
- Configuring Role Mapping in OCI IAM
- Network Policies

CNC Console IAM MySQL Secret Configuration

Use the following procedure to create MySQL Kubernetes secret:

1. Log in to Bastion Host or server from where kubectl can be executed
2. Create namespace for the secret by running the following commands:
   a. Verify whether the required namespace already exists in system by running the following command:

      ```
      $ kubectl get namespaces
      ```

   b. If the output of the above command does not display the required namespace, create the namespace by running following command:

      ```
      $ kubectl create namespace <required namespace>
      ```

      Example:

      ```
      $ kubectl create namespace cncc
      ```

3. Run the following command to create the Kubernetes secret for MySQL:

   ```
   kubectl create secret generic <database secret name> --from-
   literal=dbUserNameKey=<CNCC
   Mysql database username> --from-literal=dbPasswordKey=<CNCC Mysql database
   password> -n <Namespace of MySQL secret
   ```

   Example:

   ```
   $ kubectl create secret generic cncc-db-secret --from-
   literal=dbUserNameKey=root --from-
   literal=dbPasswordKey=mypass -n cncc
   ```

4. Run the following command to verify the secret creation:

   ```
   $ kubectl describe secret <database secret name> -n <Namespace of MySQL
   secret>
   ```

   Example:

   ```
   $ kubectl describe secret cncc-db-secret -n cncc
   ```

CNC Console IAM Default User (Admin) Secret Configuration

> **Note**
>
> Not applicable for OCI deployment.

Use the following procedure to create default user (Admin) secret :

1. Log in to Bastion Host or server from where kubectl can be executed
2. Create namespace for the secret by running the following commands:
   Verify whether the required namespace already exists in system by running the following command:

   ```
   $ kubectl get namespaces
   ```

3. If the output of the above command does not display the required namespace then create the namespace by running following command:

   ```
   $ kubectl create namespace <required namespace>
   ```

   Example:

   ```
   $ kubectl create namespace cncc
   ```

4. Run the following command to create the Kubernetes secret for MySQL for Admin User:

   ```
   $ kubectl create secret generic <secret-name> --from-
   literal=iamAdminPasswordKey=<password> --namespace <namespace>
   ```

   Example:

   ```
   $ kubectl create secret generic cncc-iam-secret --from-
   literal=iamAdminPasswordKey=cncciampasswordvalue --namespace cncc
   ```

5. Run the following command to verify the secret creation:

   ```
   $ kubectl describe secret <secret name> -n <namespace>
   ```

   Example:

   ```
   $ kubectl describe secret cncc-iam-secret -n cncc
   ```

OCI IAM Secret Configuration

> **Note**
>
> This section is applicable only for OCI deployment.

1. Login to Bastion Host or server from where kubectl can be run.
2. Run the following commands to create the oci iam secret:
   a. Run the following command to create the Kubernetes secret for OCI IAM clientId and clientSecret:

      ```
      $ kubectl create secret generic <secret-name> --from-
      literal=clientId='<clientId>' --from-
      literal=clientSecret='<clientSecret>' --namespace <namespace>
      ```

b. Run the following command to verify whether the secret is created:

```
$ kubectl describe secret <secret name> -n <namespace>
```

Example:

```
$ kubectl create secret generic oci-iam-secret --from-
literal=clientId='269d98xxxxbb5064' --from-
literal=clientSecret='6779exxxxx9602' --namespace cncc
$ kubectl describe secret oci-iam-secret -n cncc
```

OCI IAM Admin Secret Configuration

> **Note**
>
> This section is applicable only for OCI deployment.

1. Login to Bastion Host or server from where kubectl can be run.
2. Run the following commands to create the oci iam secret:
   a. Run the following command to create the Kubernetes secret for OCI IAM username and password:

```
$ kubectl create secret generic <secret-name> --from-
literal=username='<username>' --from-literal=password='<password>' --
namespace <namespace>
```

   b. Run the following command to verify whether the secret is created:

```
$ kubectl describe secret <secret name> -n <namespace>
```

Example:

```
$ kubectl create secret generic oci-iam-admin-secret --from-
literal=username=admin --from-literal=password='adminpass' --namespace
cncc
$ kubectl describe secret oci-iam-admin-secret -n cncc
```

CNC Console IAM LDAPS Secret Configuration

Use the following procedure to create the secrets to enable LDAPS:

> **Note**
>
> The value of `ssl_truststore.txt` and `ssl_truststore-password-key` value must be same.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the secret by running the following commands:

Verify whether the required namespace already exists in system by running the following command:

```
$ kubectl get namespaces
```

If the output of the above command does not display the required namespace then create the namespace by running following command:

```
$ kubectl create namespace <required namespace>
```

Example:

```
$ kubectl create namespace cncc
```

3. Create a secret by running the following command:

```
kubectl create secret generic <secret-name> --from-file=<caroot.cer>
--from-file=ssl_truststore.txt --from-literal=ssl_truststore-password-
key=<password> --namespace cncc
```

> **Note**
>
> The command is used for Kubernetes secret updates in future.

Example:

```
$ kubectl create secret generic cncc-iam-kc-root-ca --from-file=caroot.cer
      --from-file=ssl_truststore.txt --from-literal=ssl_truststore-
password-key=<password> --namespace
      cncc
```

Run the following to display the sample ssl_truststore.txt:

```
echo <password> > ssl_truststore.txt
```

4. On successfully running the above command, the following message is displayed: *secret/cncc-iam-kc-root-ca created*
5. Run the following command to verify the secret creation:

```
$ kubectl describe secret cncc-iam-kc-root-ca -n cncc
```

CNC Console IAM LDAP Configuration

Use the following procedure to configure CNC Console IAM LDAP :

1. Set up User Federation with CNC Console IAM by running following steps:
   a. Log in to CNC Console IAM application.
   b. Select **Cncc Realms** and then select **User Federation**; User federation Screen appears.

c. Fill the necessary parameters and save.

d. New buttons (**Synchronize changed users**, **Synchronize all users**, **Remove imported**, **Unlink users**) appear next to the **Save** and **Cancel**.

e. If a user has to be imported to CNCC-IAM, Click **Synchronize all users**.

f. The user can view the imported users by clicking **Users** under **Manage** in the left pane and click **View all users** in the right pane.

2. Steps to add Group-Mapper and Assign Roles:

a. Log in to CNC Console IAM application.

b. Select **Cncc Realms** and then select **User Federation**; User federation Screen appears.

c. Click **Configure** and select **User Federation**. Click **ldap** (Console Display Name) and select the **Mappers** tab, and click **Create**.

d. The Add User federation mapper page appears. Select '**group-ldap-mapper**' as **Mapper Type** from dropdown menu. Click **Save**.

e. Enter the details in the new screen and Save.

f. New buttons **Synchronize LDAP Groups to Keyclaok** and **Synchronize Keyclaok Groups to LDAP** appear.

g. Click **Synchronize LDAP Groups to Keyclaok**.

h. Select the **Groups** in the left pane and click the **View all groups** in the right pane.

i. Click any group and then click **Edit**. The following tabs appear: **Settings**, **Attributes**, **Role Mappings**, and **Members**.

j. Select **Role Mapping** tab to see a list of roles that are pre-defined in cncc-iam.

k. Select one or more roles from **Available Roles** and assign it to the group.

CNC Console TLS Secret configuration

Use the following procedure to configure CNC C TLS Secret:

1. To create Kubernetes secret for HTTPS, the following files are required:
   - ECDSA private key and CA signed certificate of CNC Console (if initial Algorithm is ES256)
   - RSA private key and CA signed certificate of CNC Console (if initial Algorithm is RSA256)
   - TrustStore password file
   - KeyStore password file
   - CA certificate
2. Create a secret by running the following command:

```
$ kubectl create secret generic <secret-name> --
fromfile=<ssl_ecdsa_private_key.pem>
      --from-file=<rsa_private_key_pkcs1.pem> --
fromfile=<ssl_truststore.txt>
      --from-file=<ssl_keystore.txt> --from-file=<caroot.cer> --
fromfile=<ssl_rsa_certificate.crt>
      --from-file=<ssl_ecdsa_certificate.crt> -n <Namespace of CNCC IAM
```

```
Ingress Gateway
    secret>
```

Example:

```
$ kubectl create secret generic cncc-iam-ingress-secret
      --fromfile=ssl_ecdsa_private_key.pem  --from-
file=rsa_private_key_pkcs1.pem
      --fromfile=ssl_truststore.txt --from-file=ssl_keystore.txt --from-
file=caroot.cer
      --fromfile=ssl_rsa_certificate.crt --from-
file=ssl_ecdsa_certificate.crt -n
    cncc
```

On successfully running the above command, the following message will be displayed:

```
secret/cncc-iam-ingress-secret created
```

Run the following command to verify the secret creation:

```
$ kubectl describe secret cncc-iam-ingress-secret -n cncc
```

3. This section explains how to update the secrets for enabling HTTPS, if they already exist: Create a secret by running the following command:

```
$ kubectl create secret generic <secret-name> --
fromfile=<ssl_ecdsa_private_key.pem>
      --from-file=<rsa_private_key_pkcs1.pem> --
fromfile=<ssl_truststore.txt>
      --from-file=<ssl_keystore.txt> --from-file=<caroot.cer> --
fromfile=<ssl_rsa_certificate.crt>
      --from-file=<ssl_ecdsa_certificate.crt> --dry-run -o yaml -n
<Namespace of CNCC IAM Ingress
      Gateway secret> | kubectl replace -f - -n <Namespace of CNCC IAM
Ingress Gateway
    secret>
```

Example:

```
$ kubectl create secret generic cncc-iam-ingress-secret
      --fromfile=ssl_ecdsa_private_key.pem  --from-
file=rsa_private_key_pkcs1.pem
      --fromfile=ssl_truststore.txt --from-file=ssl_keystore.txt --from-
file=caroot.cer
      --fromfile=ssl_rsa_certificate.crt --from-
file=ssl_ecdsa_certificate.crt --dry-run -o yaml -n
      cncc | kubectl replace -f - -n cncc
```

On successfully running the above command, the following message will be displayed:

```
secret/cncc-iam-ingress-secret replaced
```

CNC Console Core Secret Configuration to Enable HTTPS

> **Note**
>
> Not applicable for OCI deployment.

Use the following procedure to configure CNC Console Core Secret to Enable HTTPS:

1. To create Kubernetes secret for HTTPS, the following files are required:
   - ECDSA private key and CA signed certificate of CNC Console (if initial Algorithm is ES256)
   - RSA private key and CA signed certificate of CNC Console (if initial Algorithm is RSA256)
   - TrustStore password file
   - KeyStore password file
   - CA certificate
2. Create a secret by running the following command:

   ```
   $ kubectl create secret generic <secret-name> --
   fromfile=<ssl_ecdsa_private_key.pem>
         --from-file=<rsa_private_key_pkcs1.pem> --
   fromfile=<ssl_truststore.txt>
         --from-file=<ssl_keystore.txt> --from-file=<caroot.cer> --
   fromfile=<ssl_rsa_certificate.crt>
         --from-file=<ssl_ecdsa_certificate.crt> -n <Namespace of CNCC Core
   Ingress Gateway
       secret>
   ```

   Example:

   ```
   kubectl create secret generic cncc-core-ingress-secret --
   fromfile=ssl_ecdsa_private_key.pem
         --from-file=rsa_private_key_pkcs1.pem --fromfile=ssl_truststore.txt
         --from-file=ssl_keystore.txt --from-file=caroot.cer --
   fromfile=ssl_rsa_certificate.crt
         --from-file=ssl_ecdsa_certificate.crt -n cncc
   ```

   On successfully running the above command, the following message will be displayed:

   ```
   secret/cncc-core-ingress-secret created
   ```

   Run the following command to verify the secret creation:

   ```
   $ kubectl describe secret cncc-core-ingress-secret -n cncc
   ```

3. This section explains how to update the secrets for enabling HTTPS if they already exist:

Create a secret by running the following command:

```
$ kubectl create secret generic <secret-name> --
fromfile=<ssl_ecdsa_private_key.pem>
      --from-file=<rsa_private_key_pkcs1.pem> --
fromfile=<ssl_truststore.txt>
      --from-file=<ssl_keystore.txt> --from-file=<caroot.cer> --
fromfile=<ssl_rsa_certificate.crt>
      --from-file=<ssl_ecdsa_certificate.crt> --dry-run -o yaml -n
<Namespace of CNCC Core Ingress
      Gateway secret> | kubectl replace -f - -n <Namespace of CNCC Core
Ingress Gateway
    secret>
```

Example:

```
$ kubectl create secret generic cncc-core-ingress-secret
      --fromfile=ssl_ecdsa_private_key.pem  --from-
file=rsa_private_key_pkcs1.pem
      --fromfile=ssl_truststore.txt --from-file=ssl_keystore.txt --from-
file=caroot.cer
      --fromfile=ssl_rsa_certificate.crt --from-
file=ssl_ecdsa_certificate.crt --dry-run -o yaml -n
      cncc | kubectl replace -f - -n cncc
```

On successfully running the above command, the following message will be displayed:

```
secret/cncc-core-ingress-secret replaced
```

CNC Console IAM SAML Configuration

Use the following procedure to configure CNC Console IAM SAML:

1. To configure SAML identity provider **(IdP)** in CNC Console IAM, log in to CNC Console IAM Console using admin credentials provided during installation of CNC Console IAM .
2. Select **Cncc** realm and the **Identity Provider** tab in the left pane. **Identity Providers** screen appears in the right pane.
3. From the **Add provider** drop down list select the **saml** entry and the **Add Identity Provider** screen appears.
4. To create custom 'First Login Flow', click **Authentication** tab In the left pane. The **Authentication** screen appears.
5. Click **New** at the right pane. **Create Top Level Form** screen appears.
   Enter the appropriate alias and click **Save**.
6. The **Authentication** screen with the newly created custom flow selected in the drop down list appears. Click **Add Execution** in the right pane .
7. **Create Authenticator Execution** screen appears.
   Select **Create User If Unique** from the **Provider** drop down list. Click **Save**.
8. The **Authentication** screen appears with the newly created custom flow selected in the drop down. Under **Requirement** section, select **Alternative**.
9. Select **Identity Provider** in the left pane. Select the custom flow from **First Login Flow** drop down list.

OCI IAM SAML Configuration

SAML (Security Assertion Markup Language) enables applications to authenticate a user using an identity provider. The identity provider authenticates the user and returns the assertion information about the authenticated user and the authentication event to the application. If the user tries to access any other application that uses the same identity provider for user authentication, the user shall not be required to log in a second time and will be granted access. This is the principle of SSO (Single Sign On).

> **Note**
>
> - OCI IAM provides option to implement SAML SSO. This is an optional step.
> - Applicable only for OCI deployment.

The following section describes the steps to implement SAML SSO in OCI IAM.

Adding a SAML Identity Provider in OCI IAM

1. Navigate to the identity domain: Open the navigation menu and click **Identity & Security**. Under **Identity**, click **Domains**.
2. Click the name of the identity domain that you want to work in. You might need to change the compartment to find the domain that you want.
3. Then, click **Security** and then **Identity providers**.
4. Click **Add IdP**, and then click **Add SAML IdP**.
5. Enter the following information:
   - **Name**: Enter the name of the IdP.
   - (Optional) **Description**: Enter a description of the IdP.
   - (Optional) **Identity provider icon**: Drag and drop a supported image, or click **select one** to browse for the image.
6. Click **Next**.
7. On the **Exchange metadata** screen, do one of the following:
   - **Import IdP metadata**: Select this option if you have an XML file exported from your IdP. Drag and drop the XML file to upload the metadata, or click **select one** to browse for the metadata file.
   - **Enter IdP metadata**: Select this option if you want to manually enter the IdP metadata. Provide the following details:
     - **Identity provider issuer URI**
     - **SSO service URI**
     - **SSO service binding**
     - **Upload identity provider signing certificate**
     - **Enable global logout**
     - **Identity provider logout request URL**
     - **Identity provider logout response URL**

8. On the **Map User Identity Screen,** keep the **Requested Name ID Format** as **None**
9. Map user's identity attributes received from the IdP to an Oracle Cloud Infrastructure identity domain. Mapping options vary based on identity provider. You might be able to directly assign an IdP value to an Oracle Cloud Infrastructure identity domain value. For example, **NameID** might map to **UserName**. If you select SAML assertion attribute as the source, select the Assertion attribute name and then enter the Oracle Cloud Infrastructure identity domain.
10. Click **Submit**.
11. On the **Review and create** screen, review your SAML identity provider settings. If the settings are correct, click **Create**. Click **Edit** next to the set of settings, if you need to change them.
12. The console displays a message when the SAML identity provider is created. You can do the following from the overview page:
    - Click **Test** to verify that the SAML SSO connection is working correctly.
    - Click **Activate** to activate the IdP so the identity domain can use it.
    - Click **Assign to IdP policy rule** to assign this SAML identity provider to an existing policy rule you have created.
13. Click **Close**.

- At this point, the SAML IDP is configured, but any user created in SAML IDP needs to be created in OCI IAM in-order to allow a successful Login
- In order to directly login to CNCC Core via SAML IdP, JIT Provisioning needs to configured (Just-In-Time)

1. OCI IAMs SAML Metadata can be exported, by clicking on "Export SAML Metadata" on the "Identity Providers" Tab as shown above.
2. The exported file, can be used to configure SAML Client in an IdP

3. JIT Configuration in OCI IAM
JIT stands for Jut-In-Time Provisioning. It allows the federated user to be created in OCI IAM users List whenever a federated user logs in for the first time. Follow below steps to configure this in OCI-IAM

1. On the IDP Configuration Page, click **Configure JIT**
2. Enable Just-In_Time(JIT) provisioning
3. Select one/both of the following options as required:
    - **Create a new identity domain user**: Create an identity user in the identity domain, if the user doesn't exist when sign in with the identity provider.
    - **Update the existing identity domain user**: Merge and overwrite identity domain user account data from the mapped IdP. The existing data is overwritten by the user data from the IdP.
4. In the **Map user attributes** area , map a user account from the IdP to a user account from the identity domain.
    a. Select a value in the **IdP user attribute type** row.
        - If you select **Attribute**, then enter the IdP user attribute name.
        - If you select **NameID**, you don't need to enter the IdP user attribute name.
    b. (Optional) Select the identity domain user attribute.

c. (Optional) Add more identity domain attributes.

5. For example, as per above screenshot **NameID value** to **userName** is the default mapping, and the **name** to **familyName** mapping is configured by the user.
   - OCI IAM will look for an attribute named ***name***in the Assertions coming from SAML IdP, as mentioned below:

```
<saml:AttributeStatement>
      <saml:Attribute FriendlyName="name"
                      Name="name"
                      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
        <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
                             xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
                             xsi:type="xs:string">SamlUser</
saml:AttributeValue>
      </saml:Attribute>
```

The user needs to make sure the SAML IDP is populating these attributes in the assertions , else JIT Configuring will not work, thus failing the SAML SSO Authentication

4. Configuring Role Mapping in OCI-IAM
Role Mapping for SAML in OCI IAM is configured as a part of JIT Provisioning configuration. Below mentioned steps can be followed

1. On the IDP Configuration Page, click **Configure JIT**
2. To enable group mapping, **click Assign group mapping**
3. For **Group membership attribute name** enter the IdP attribute name that contains group memberships.
4. To import the group settings, select one of the following options:
   - **Define explicit group mapping**: This option requires you to provide the group name to map between the IdP and identity domain. If you select this option, enter the IdP group name and select an available identity domain group name.
   - **Assign implicit group mapping**: This option maps an IdP group to an identity domain group that has the same name. No other action is required.
     - OCI-IAM reads Roles as Groups, "Group membership attribute name" has assigned as **groups**
     - After this CNC Console (OCI-IAM) Groups can be mapped to the IdP Groups as mentioned in the below screenshot
5. Under **Assignment rules**, specify actions to take when assigning group memeberships:
   a. If users are assigned to existing groups, select whether to merge with existing group memberships or replace existing group memberships.
6. When a group isn't found, select to take one of the following actions:
   a. **Ignore the missing group**: The user successfully signs in.
   b. **Fail the entire request**: The sign-in attempt fails.
7. Click **Save Changes**.

- OCI-IAM will be reading Groups from the SAML Assertions.

- As the Role Mapping is created through 'groups' attribute name, OCI-IAM will look for 'groups' attribute in SAML Assertions.
- Ensure the IDP is sending the required attributes in assertions

5. Assigning Idp to Identity provider (IdP) policies

1. Under your domain , click **Security** and then **Idp policies.**
2. Create **Idp policy** if required else use **Default Identity Provider Policy**.
3. Under your **Identity Provider Policy ,** click on Add **Idp Rule** to create a new Rule if required else to use **Default IDP Rule** click on the **Edit IdP rule**.
4. Assign the name of IdP under **Assign identity providers**.
5. Click **Save** to save the changes and exit.

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Configuration Console, Installation, Upgrade, and Fault Recovery Guide*.

3.1.8 Cloud Native Environment (CNE) Specific Security Recommendations and Guidelines

After installation, audit the CNE security system stance before deploying the system into service. This primarily consists of changing credentials and unique SSH keys to trusted servers. The following table lists all the credentials that need to be checked, changed, and retained:

> **Note**
>
> kubectl commands might vary based on the platform deployment. Replace kubectl with Kubernetes environment-specific command line tool to configure Kubernetes resources through kube-api server. The instructions provided in this document are as per the Oracle Communications Cloud Native Core, Cloud Native Environment (CNE) version of kube-api server.

> **⚠ Caution**
>
> User, computer and applications, and character encoding settings may cause an issue when copy-pasting commands or any content from PDF. PDF reader version also affects the copy-pasting functionality. It is recommended to verify the pasted content especially when the hyphens or any special characters are part of the copied content.

Table 3-2 Credentials

Credential Name	Deployment	Credential Type	Associated Resource	Initial Setting for Credential Type	Credential Rotation
TOR Switch	BareMetal Only	username and password	Cisco Top or Rack Switch	username and password from PreFlight Checklist	Reset postinstallation
Enclosure Switch	BareMetal Only	username and password	HP Enclosure Switch	username and password from PreFlight Checklist	Reset postinstallation
OA Admin	BareMetal Only	username and password	On-board Administrator Console	username and password from PreFlight Checklist	Reset postinstallation
ILO Admin	BareMetal Only	username and password	HP Integrated Lights Out Manger	username and password from PreFlight Checklist	Reset postinstallation
Server Super User (root)	All	username and password	Server Super User	Set to well-known Oracle default during server installation	Reset postinstallation
Server Super User (admusr)	All	username and password	Server Super User	Set to well-known Oracle default during server installation	Reset postinstallation
Server Admin User SSH	All	SSH Key Pair	Server Admin User	Key Pair generated at install time	Can rotate keys at any time; key distribution manual procedure

If factory or Oracle defaults were used for any of these credentials, they must be changed before placing the system into operation. The customer must store these credentials safely and securely offsite. It is recommended that the customer must plan a regular schedule for updating (rotating) these credentials. Specific procedures and recommendations for CNE credential management are provided below:

> **ⓘ Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

- Network Security Recommendations and Procedures
  - Network Policies Recommendations
  - DNS Recommendations

Network Security Recommendations and Procedures

Network Policies Recommendations

> **Note**
>
> **Recommendation**: It is recommended to keep the default configuration for the network policies to provide an extra security layer on the common services.

CNE has implemented network policies on common services. The network policies created during installation or upgrade to 24.2.x. on the following services:

- AlertManager

- Prometheus
- Grafana
- Jaeger
- OpenSearch

DNS Recommendations

At present, CNE supports two options for DNS:

1. External DNS via Bastion Host forwarding to external server.
2. Local (internal) DNS with DNS server on Bastion host.

CNE's Bastion Host has the Domain Name System Security Extensions (DNSSEC) feature. DNSSEC adds a layer of trust on top of DNS, by providing authentication. When a FQDN is looked into the DNS resolver, then DNS performs an extra validation by authenticating the information of the published of the FQDN.

> **Note**
>
> **Recommendation:**You must enable DNSSEC to add authentication into the DNS resolution. CNE has enabled by default the DNSSEC, but disabled by default the validation of the DNS records using DNSSEC. It is important to enable the feature once your infrastructure can provide authentication whenever an FQDN is being resolved by the DNS server. Having it enabled ensures that the resolved FQDN comes from a valid party.
>
> Not enabling DNSSEC increases the risk of a DNS hijacking attacks.

Credential Management Procedures

Setting Top Of Rack Switch Credentials

> **Note**
>
> **Recommendation**: Follow the configuring the TOR switches procedures.
>
> The *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide* provides the detailed procedures of how to configure the TOR switches and configure them for remote monitoring. Deviations from the standard installation time configurations are not recommended.

For more information, see *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide.*
This procedure is used to set the credentials on the cisco TOR switch as deployed with the BareMetal deployment option. Steps for creating and deleting accounts and for setting account passwords are given below:

For more details, refer to Nexus commands to configure Top of Rack switch username and password.

1. Log in to the TOR switch (from the Bastion Host):

```
$ ssh <username>@<switch IP address>

User Access Verification
Password: <password>

Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
...
...
<switch name>#
```

2. Change the password for <username>:

```
# configure
Enter configuration commands, one per line. End with CNTL/Z.
(config)# username <username> password <newpassword>
(config)#exit
```

3. Create a new user (if required):

```
# configure
Enter configuration commands, one per line. End with CNTL/Z.
(config)# username <newusername> password <newpassword> role [network-
operator|network-admin|vdc-admin|vdc-operator]
(config)#exit
```

4. Verify the account changes by exiting the ssh session (type exit) and repeat step 1.

```
# exit
Connection to <switch IP address> closed.
$
$ ssh <newusername>@<switch IP address>
User Access Verification
Password: <newpassword>

Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
...
...
<switch name>#
```

5. Delete an unrequired user account:

```
# configure
Enter configuration commands, one per line. End with CNTL/Z.
(config)# no username <username>
(config)#exit
```

6. Change the enable secret:

```
(config)# enable secret <newenablepassword>
(config)# exit
```

7. Save the configuration changes:

```
# copy running-config startup-config
[########################################] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.
```

> **Note**
>
> - Change TOR passwords before placing site into service: The TOR switch credentials show the changes prior to placing the site into service.
> - Use Strong Passwords: The Network Administrator must choose complex TOR Switch passwords as per their organization's security guidelines.

Setting Enclosure Switch Credentials

This procedure is used to set the credentials on the HP enclosure switch as deployed with the BareMetal deployment option. Steps for creating and deleting accounts and for setting account passwords is given below. For additional information, refer to *HP commands to configure enclosure switch username and password* section in the *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide.* .

Setting Enclosure Switch Credentials

1. Log in to the iLO with username and password (from the):

```
[root@winterfell ~]# ssh <username>@<iLO address>
<username>@<iLO address>'s password: <password>
User:<username> logged-in to ...(<iLO address> / <ipv6 address>)

iLO Advanced 2.61 at  Jul 27 2018
Server Name: <server name>
Server Power: On

</>hpiLO->
</>hpiLO-> set /map1/accounts1/<username> password=<newpassword>

status=0
status_tag=COMMAND COMPLETED
Tue Aug 20 13:27:08 2019

</>hpiLO->
```

2. Change the password for the current username:

```
[switchname]local-user <username>class <currentclass>
[switchname-luser-manage-<username>]password simple <newpassword>
[switchname-luser-manage-<username>]quit
```

3. Create a new user account:

```
</>hpiLO-> create /map1/accounts1 username=<newusername>
password=<newpassword> group=admin,config,oemHP_rc,oemHP_power,oemHP_vm
status=0
status_tag=COMMAND COMPLETED
Tue Aug 20 13:47:56 2019

User added successfully.
```

4. Verify the account changes by exiting the ssh session (type exit) and repeat step 1.

```
</>hpiLO-> exit

status=0
status_tag=COMMAND COMPLETED
Tue Aug 20 13:30:52 2019



CLI session stopped
Received disconnect from <iLO address> port 22:11:  Client Disconnect
Disconnected from <iLO address> port 22

[bastion host]# ssh <newusername>@<iLO address>
<newusername>@<iLO address>'s password: <newpassword>
User:<newusername> logged-in to ...(<iLO address> / <ipv6 address>)

iLO Advanced 2.61 at  Jul 27 2018
Server Name: <server name>
Server Power: On

</>hpiLO->
```

5. Delete the user account that is not required:

```
</>hpiLO-> delete /map1/accounts1/<username>

status=0
status_tag=COMMAND COMPLETED
Tue Aug 20 13:59:04 2019

User deleted successfully.
```

> **Note**
>
> - Set Enclosure Switch Credentials before Placing Into Service: The HP Enclosure switch credentials show are to be changed prior to placing the site into service.
> - Use Strong Passwords: The Network Administrator must choose complex Enclosure Switch passwords as per their organization's security guidelines.

Hosting Environment Security Recommendations and Procedures

The best way to keep your CNE environment secure is to keep it updated. New CNE releases are typically carried out every three months. The CNE upgrade does not affect the service and typically installs the newer versions of:

- Host OSs
- Kubernetes and associated containers
- Common service containers

The upgrade process ensures that the uplifts do not affect active service. See *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide* for more details.

> **Note**
>
> Given below are some Repository Management Recommendations to be followed:
>
> - Do not perform any YUM updates on hosts outside of the upgrade pipeline, as this can interfere and affect negatively the cluster.
> - Some updates may require system reboots to complete and thus should only be performed on nodes not actively providing service.
> - The Oracle Linux 9 (OL9) security guide is available at: https://docs.oracle.com/cd/F61088_01/security/. This guide provides additional details for specific security procedures, several of the procedures found in the general OL9 guide are not appropriate for the CNE environment. Contact My Oracle Support before attempting any hardening activity that are not recommended.

Repository Management Recommendations

As part of the CNE installation, a central repository must be set. The central repository must contain the following:

- HTTP Repository: Serves Python binaries and other required packages.
- YUM Oracle Repository: Serves YUM packages. This is a mirror from Oracle Yum Repository.
- Container Image Repository: Provides the images required for a successful Kubernetes cluster deployment.

This central repository hosts the required artifacts for a successful CNE installation and upgrade procedure.

System Update (YUM) Recommendations:

- Keep central YUM repositories updated:

  - Ensure that you update the YUM packages in the central repositories to the latest version. YUM updates are performed whenever you install or upgrade CNE. Keeping the YUM repository up-to-date ensures that the fixes for all publish vulnerabilities are applied.
  - The deployed YUM Oracle server is a mirror from official Oracle YUM repository. There are secure mechanisms that are already implemented when performing the retrieval of this repository (for example, the usage of Gnu Privacy Guard (GPG) keys and HTTPS connection). The security controls are part of the unbreakable Linux network. For more information about, GPG keys, see https://linux.oracle.com/security/gpg/index.html.

- Scan YUM Server prior installation:
  When you complete setting up a central repository, scan the YUM Oracle server located at the central repository. For more information about setting up the central repository, see the "Setting Up a Central Repository" section in *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide*.

  It is a good practice to retrieve the YUM Oracle Server in a secure way from Oracle's server. For more information, see https://linux.oracle.com/security/gpg/index.htm.

  Use any of Software Composition Analysis tools (such as Trivy or Grype) to perform the scan. If needed, you may perform a Malware scan to the Yum server with any malware scanning tool.

Container Repository Recommendations

Scan Container Registry prior installation:

When you complete setting up a central repository, scan the container registry located at the central repository. For more information about setting up the central repository, see the "Setting Up a Central Repository" section in *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide.*

Use any of Software Composition Analysis tools (such as Trivy or Grype) to perform the scan. All images are scanned and vulnerabilities assessed at product development time, but new exploits / vulnerabilities may be reported / fixed later.

Scan tools typically use a database of known vulnerabilities. Refer to tool vendor for instructions on creating off-line (internet isolated) vulnerability databases.

- Scan docker image repositories regularly: Scan your docker image repositories regularly using a tool such as clair or anchore-engine. All images are scanned and vulnerabilities are assessed at product development time, but new exploits or vulnerabilities may be reported or fixed later.

  Scan tools use a database of known vulnerabilities. Refer to tool vendor for instructions on creating off-line (internet isolated) vulnerability databases.

Credential Management Procedures

The given below procedures to manage your credentials:

Setting HP Onboard Administrator (OA) Credentials.

This procedure is applicable only to BareMetal deployments. This procedure is used to set the credentials on the HP Onboard Administrator as deployed with the BareMetal deployment option. Steps for creating and deleting accounts and for setting account passwords are shown. For additional information, refer to *HP commands to configure OA username and password* section in the *Oracle Communications Cloud Native Core, Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide*.

1. Log in to the OA:

```
$ ssh <username>@<OA address>
```

> **Note**
>
> This is a private system. Do not attempt to log in unless you are unauthorized user. Any authorized or unauthorized access and use may be monitored and can result in criminal or civil prosecution under applicable law.

```
Firmware Version: 4.85
Built:04/06/2018@06:14OA
 Bay Number:1
OA Role: Active
<username>@<OA address>'s password: <password>
HPE BladeSystem Onboard Administrator
(C) Copyright 2006-2018 Hewlett Packard Enterprise Development LP
Type 'HELP' to display a list of valid commands.
Type 'HELP <command>' to display detailed information about a specific
command.
Type 'HELP HELP' to display more detailed information about the help
system.
OA-A45D36FD5FB1>
```

2. Change the current password:

```
OA-A45D36FD5FB1> set password <newpassword>
Changed password for the"<username>"user account.
OA-A45D36FD5FB1>
```

3. Add new user:

```
OA-A45D36FD5FB1> add user <newusername>
New Password: <newpassword>
Confirm : <newpassword>
User"<newusername>"created.
You may set user privileges with the 'SET USER ACCESS' and 'ASSIGN'
commands.
OA-A45D36FD5FB1> set user access <newusername> [ADMINISTRATOR|OPERATOR|
USER] "<newusername>"
has been given [administrator|operator|user] level privileges.
```

4. Assign full access to the enclosure for the user:

```
OA-A45D36FD5FB1> assign server all <newusername>
<newusername> has been granted access to the valid requested bay(s
OA-A45D36FD5FB1> assign interconnect all <newusername>
<newusername> has been granted access to the valid requested bay(s)
```

```
OA-A45D36FD5FB1> assign oa <newusername>
<newusername> has been granted access to the OA.
```

5. Verify the new account:

```
OA-A45D36FD5FB1> exit
Connection to <OA address> closed.
[bastion host]# ssh <newusername>@<OA address>
WARNING: This is a private system. Do not attempt to log in unless you are
unauthorized user.
Any authorized or unauthorized access and use may be monitored and can
result in criminal or
civil prosecution under applicable law.
Firmware Version : 4.85
Built : 04/06/2018 @ 06:14
OA Bay Number : 1
OA Role : Active
<newusername>@<OA address>'s password: <newpassword>
HPE BladeSystem Onboard Administrator
(C) Copyright 2006-2018 Hewlett Packard Enterprise Development LP
Type 'HELP' to display a list of valid commands.
Type 'HELP <command>' to display detailed information about a specific
command.
Type 'HELP HELP' to display more detailed information about the help
system. OA-A45D36FD5FB1>
```

6. Delete the user account that is not required:

```
OA-A45D36FD5FB1> remove user <username>
Entering anything other than 'YES' will result in the command not
executing.
Are you sure you want to remove testuser1? yes
User"<username>"removed.
```

Setting HP Integrated Lights Out Manger (ILO) Credentials

This procedure is applicable only to BareMetal deployments. This procedure is used to set the credentials on the HP Integrated Lights Out Managers as deployed with the BareMetal deployment option. Steps for creating and deleting accounts and for setting account passwords is shown.

1. Log in to the iLO:

```
$ ssh <username>@<iLO address>


<username>@<iLO address>'s password: <password>User:<username>
logged-in to ...(<iLO address> / <ipv6 address>)
iLO Advanced2.61at Jul272018
Server Name: <server name>
Server Power: On
</>hpiLO->
```

2. Change the current password:

```
</>hpiLO-> set /map1/accounts1/ <username> password= <newpassword>
status=0
status_tag=COMMAND COMPLETED
Tue Aug2013:27:082019
</>hpiLO->
```

3. Create a new user account:

```
</>hpiLO-> create /map1/accounts1 username= <newusername> password=
<newpassword>
group=admin,config,oemHP_rc,oemHP_power,oemHP_vm
status=0
status_tag=COMMAND COMPLETED
Tue Aug2013:47:562019
User added successfully.
```

4. Verify the new user account:

```
</>hpiLO-> exit
status=0
status_tag=COMMAND COMPLETED
Tue Aug2013:30:522019CLI session stoppedReceived disconnect from <iLO
address> port22:11: Client Disconnect
Disconnected from <iLO address> port22
[bastion host]# ssh <newusername>@<iLO address>
<newusername>@<iLO address>'s password: <newpassword>
User:<newusername> logged-in to ...(<iLO address> / <ipv6 address>)
iLO Advanced2.61at Jul272018
Server Name: <server name>Server
 Power: On</>hpiLO->
```

5. Delete the user account that is not required:

```
</>hpiLO-> delete /map1/accounts1/ <username>
status=0
status_tag=COMMAND COMPLETED
Tue Aug2013:59:042019
User deleted successfully.
```

Setting Root Passwords for All Cluster Nodes

The procedure to reset the root account requires that the administrator log in to each and every server. This procedure is applicable to all CNE deployments.

To reset the root account, perform the following steps for each and every server in the cluster:

1. Log in to the next server:

```
$ ssh admusr@ <cluster server IP>
```

2. Perform the root password change:

```
$ sudo passwd root


New password: <new password>
Retype new password: <new password>
Retype new password:<new password>
```

3. Repeat step 1 and step 2 for each and every server in the cluster.

> **Note**
>
> The administrator (admusr) account is provided without a usable password hash. Thus requiring the use of SSH keys to access the account. The SUDO user access is configured without the requirement of a password. If you would like to enable the SUDO passwords for the administrator, you also need to assign a password to the administrator account using a procedure very similar to the one outlined above.

Reset or Delete Credentials for the admusr account on Each and Every Server

The procedure to reset or delete the admusr account. This procedure requires root privileges and must be applied on each and every server.

To reset or delete the admusr account, perform the following steps for each and every server in the cluster:

1. Log in to the next server:

```
$ ssh admusr@ <cluster server IP>
```

or

```
$ ssh cloud-user@<cluster server IP>
```

2. Perform the root password change:

Admusr:

```
$ sudo passwd -l admusr
New password: <new password>
Retype new password: <new password>
Retype new password: <new password>
```

Cloud-user:

```
$ sudo passwd -l cloud-user
New password: <new password>
Retype new password: <new password>
Retype new password: <new password>
```

3. Repeat step 1 and step 2 for each and every server in the cluster.

Updating admusr SSH Keys for All Cluster Nodes

There are two sets of SSH keys used in a deployed cluster: The key used to access the and the key used to access the cluster servers. This procedure is applicable to all CNE deployments.

These key-pairs are generated at install time and are only usable on the cluster they were generated for. The public key portion of the key pair is typically provided to administrators who will manage the cluster. The key pair used to access the cluster servers should be kept local to the cluster:

Table 3-3 Updating admusr SSH Keys

Key Pair Name	Public Key Distribution	Private Key Distribution
	Place copy in the authorized_keys file on the .	**Cluster Admin**: Place in the cluster admin key agent (e.g., ssh-agent or pageant) external to the cluster. Do not copy to any host on the cluster.
Cluster Hosts	Place a copy in the authorized_keys files on each and every cluster host; do not configure on the .	**Bastion Host**: ~admusr/.ssh directory. This will be used when performing orchestration activities (install / upgrade).

To replace either of these key pairs starts with an openssh request to generate a new keypair:

```
ssh-keygen -b 4096 -t rsa -C "New SSH Key" -f
      .ssh/new_occne_id_rsa -q -N ""
```

This command generates the following key pair:

Table 3-4 Key pair

Key Name	Purpose
new_occne_id_rsa	The private key
new_occne_id_rsa.pub	The public key

Updating the keys

1. Log in to the and generate a new key pair using the ssh-keygen command given above:

   ```
   $ ssh-keygen -b 4096 -t rsa -C "New SSH Key" -f /var/occne/cluster/
   <cluster_name>/.ssh/new_occne_id_rsa -q -N ""
   ```

2. Copy the private key portion of the key off cluster and make it available to your ssh agent of choice or store it in the .ssh directory of your client machine. See instructions for your specific SSH client (for example, putty or openssh)
3. Add the new public key to the authorized key file on the :

   ```
   $ cat ~/.ssh/new_occne_id_rsa.pub >> ~/.ssh/authorized_keys
   ```

4. Confirm the permissions of the .ssh directory and files:

```
$ ls -la ~/.ssh
total 32
drwx------. 2 admusr admusr 4096 Feb 25 15:48 .
drwx------. 42 admusr admusr4096 Feb 24 15:14 ..
-rw-------. 1 admusr admusr 796 Jan 28 14:43 authorized_keys
-rw-------. 2 admusr admusr 545 Feb 12 13:58 config
-rw-------. 1 admusr admusr 3239 Feb 25 15:48 new_occne_id_rsa
-rw-r--r–. 1 admusr admusr 737 Feb 25 15:48 new_occne_id_rsa.pub
```

   In general, the .ssh directory should be mode 700 and the files under that directory should be mode 600.

5. Confirm that the new key works. Remove the old key from your ssh client's agent (see instructions for your client) and confirm that you can still log in.
6. Assuming that you were able to Log in using the new key pair, remove the old key pair from the authorized_keys file using your favorite editor.

   In general, the authorized_keys file should at this point have two keys in it - the old one and the new one. The new one should be at the bottom.

> **Note**
>
> Access to Bastion Host container registry is TLS enabled and only CNE has access to it.

Change Kubernetes Secrets Encryption Key

The procedure is to change the key used to encrypt Secrets stored in the CNE Kubernetes cluster. Secret encryption is enabled by default during CNE install or upgrade.

To change Kubernetes secrets encryption key, perform the following steps:

1. Locate in

```
$ ssh <username>@<OA address>
```

2. Generate a new key with Approved Oracle Linux Randomness:

```
$ NEW_KEY=$(head -c 32 /dev/urandom | base64)
```

3. Generate a new key with Approved Oracle Linux Randomness:

```
$ KEY_NAME=$(cat /dev/random | tr -dc '[:alnum:]' | head -c 10)
```

4. Run the following command:

```
$ kubectl get nodes | awk '/control-plane/ {print $1}' | xargs -I{} ssh {}
" sudo sed -i '/keys:$/a\        - name: key_$KEY_NAME\n\
secret: $NEW_KEY' /etc/kubernetes/ssl/secrets_encryption.yaml; sudo
cat /etc/kubernetes/ssl/secrets_encryption.yaml"
```

Output shows new encryption key, key name and the contents of /etc/kubernetes/ssl/secrets_encryption.yaml file:

```
This site is for the exclusive use of Oracle and its authorized customers
and partners. Use of this site by customers and partners is subject to the
Terms of Use and Privacy Policy for this site, as well as your contract
with Oracle. Use of this site by Oracle employees is subject to company
policies, including the Code of Conduct. Unauthorized access or breach of
these terms may result in termination of your authorization to use this
site and/or civil and criminal penalties.
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets

    providers:
    - secretbox:
        keys:
        - name: key_ZOJ1Hf5OCx
          secret: l+CaDTmMkC85LwJRiWJ0LQPYVtOyZ0TdtNZ2ij+kuGA=
        - name: key
          secret: ZXJ1Ulk2U0xSbWkwejdreTlJWkFrZmpJZjhBRzg4U00=
    - identity: {}
    1m
5
```

5. Restart api server by executing following command. This ensures that all the new secrets will be encrypted with the new key.

```
kubectl get nodes | awk '/control-plane/ {print $1}' | xargs -I{} ssh {} "
sudo mv /etc/kubernetes/manifests/kube-apiserver.yaml ~; sleep 2; sudo mv
~/kube-apiserver.yaml /etc/kubernetes/manifests"
```

6. Run the following command to encrypt all the existing secrets with a new key:

```
$kubectl get secrets --all-namespaces -o json | kubectl replace -f-
```

Output:

```
secret/occne-cert-manager-webhook-ca replaced
...
secret/sh.helm.release.v1.occne-tracer.v1 replaced
secret/webhook-server-cert replaced
Error from server (Conflict): error when replacing "STDIN": Operation
cannot be fulfilled on secrets "alertmanager-occne-kube-prom-stack-kube-
alertmanager-generated": the object has been modified; please apply your
changes to the latest version and try again
Error from server (Conflict): error when replacing "STDIN": Operation
cannot be fulfilled on secrets "alertmanager-occne-kube-prom-stack-kube-
alertmanager-tls-assets-0": the object has been modified; please apply
your changes to the latest version and try again
Error from server (Conflict): error when replacing "STDIN": Operation
```

```
cannot be fulfilled on secrets "alertmanager-occne-kube-prom-stack-kube-
alertmanager-web-config": the object has been modified; please apply your
changes to the latest version and try again


Note: There may be exists some errors depending on how the secret was
created, but you can verify the content of encrypted secret using the
following commands.
```

7. For each controller node (i.e. ctrl-1), locate in the controller node.
8. Runwith sudo) the following command. This will show all existing secrets that would launch this command from a controller node after select any secret to verify the information related the new encryption key, using **cert** and **key pem** files:

   ```
   sudo ETCDCTL_API=3 /usr/local/bin/etcdctl --cert /etc/ssl/etcd/ssl/<cert
   pem file> --key /etc/ssl/etcd/ssl/<key pem file> get --keys-only=true --
   prefix /registry/secrets
   ```

9. To verify whether the new key is being used for encrypting existing secrets by running the following command from a controller node: Replace **<cert pem file>**, **<key pem file>** and **<secret>** with their corresponding values.

   ```
   sudo ETCDCTL_API=3 /usr/local/bin/etcdctl --cert /etc/ssl/etcd/ssl/<cert
   pem file> --key /etc/ssl/etcd/ssl/<key pem file> get /registry/secrets/
   <namespace>/<secret> -w fields | grep Value
   ```

   Output:

   ```
   [cloud-user@occne3-user-k8s-ctrl-3 ~]$ sudo ETCDCTL_API=3 /usr/local/bin/
   etcdctl
   --cert /etc/ssl/etcd/ssl/node-occne3-user-k8s-ctrl-1.pem
   --key /etc/ssl/etcd/ssl/node-occne3-user-k8s-ctrl-1-key.pem get /registry/
   secrets/default/secret1 -w fields | grep Value
   "Value" : "k8s:enc:secretbox:v1:key_ZOJ1Hf5OCx:<ENCRYPTED_DATA>"

   In this example, the new key key_ZOJ1Hf5OCx is being used to encrypt
   secret1 secret.
   ```

10. Repeat steps 8 and 9 for each and every **controller** server in the cluster.

General Security Administration Recommendations and Procedures

> **Note**
>
> **Record configuration changes**: In a disaster recovery scenario, Oracle provided procedures will only restore base system behavior (they will not include restoration of an special configurations or tweaks). We recommend that all post-delivery customization be logged or automated using tools such as Ansible.

Password Policy Administration Procedures

In general, the host environments use a user account named **admusr** which is not configured with a password; the only way to access this account is using SSH keys. We recommend using

SSH keys rather than passwords for all non-root accounts. The root account cannot be accessed via ssh; the only access is via the console. For this account, we recommend setting a password and storing it off-site to be used only for break-glass console access to the host.

User Administration Recommendations

Customers may want to create additional accounts to manage separate concerns (Example: a dbadmin account, a k8sadmin account, and so on). This can be done using normal Linux user administration procedures.

SSHD Policy Administration Procedures

The customer may want to create augment the standard sshd configuration to perform additional hardening; this can be done using normal Linux ssh administration procedures. In a disaster recovery scenario, Oracle provided procedures will only restore base system behavior (they will not include restoration of an special configurations or tweaks).

> **Note**
>
> **Review changes with Oracle Support**: We recommend reviewing any planned changes to sshd configuration with your Oracle Support contact. Improper sshd configuration can either open the system up to attacks or prevent proper system operation.

Auditd Policy Administration Procedures

Customers may want to augment the standard auditd configuration to perform additional monitoring; this can be done using normal Linux auditd administration procedures. Place all customizations in a separate file in the `/etc/audit/rules.d` directory, do not modify any of the other existing audit configuration files.

Container Security Recommendations and Procedures

The following are the container security recommendations and procedures:

Container Repository Management Recommendations and Procedures

The following are the container repository management recommendations and procedures:

System Update (Container) Recommendations

> **Note**
>
> Recommendation: Keep central Image repositories up-to-date.
>
> Keep central repositories up-to-date with latest recommended container packages; container updates are performed on-site whenever a fresh install or upgrade is performed. An up-to-date container repository is required for both fresh install and upgrade operations.

General Container Security Administration Recommendations and Procedures

Kubernetes Control Plane Certification Administration Procedures

Recommendation: keep monitoring the Kubernetes Certificates expire date.

Kubernetes uses many different TLS certificates to secure access to internal services. These certificates are automatically renewed during upgrade. However, if upgrade is not performed regularly, these certificates may expire and cause the Kubernetes cluster to fail. For more details, refer to the Renewing Kubernetes Certificates procedure in*Oracle Communications Cloud Native Core, Cloud Native Environment User Guide*.

Kubernetes Policy Engine (Kyverno)

CNE is deploying Kyverno. This provides policies to ensure that malicious applications do not corrupt the Kubernetes controller, worker nodes and cluster data.

Policies can be used to control and monitor workloads running on CNE, and also can be used to audit workloads running on Kubernetes. Kyverno framework is deployed as common service in CNE.

Note

- **Importance of Kyveno**
  - CNE has deployed a set of baseline policies, in order to guarantee an essential set of controls from known privilege escalations.
  - In the end, these will represent the minimum standard of Policy protection that all OC-CNE cluster will have by default.
  - Kyverno policies should not be modified nor disabled, otherwise the security risk is heavily increased and it is open to unknown attacks.
- **Kyverno default setting**: From 23.2.x and onwards, CNE has set all the Kyverno policies validation mode as "enforced". This means that, if a policy is being violated, the "enforce" mode will block any resource creation or updates that does not comply.
- **Keep the Kyverno metrics ON**: Keeping Kyverno metrics ON populates the Grafana's dashboards with the Policy enforcement and compliance monitoring.

3.2 Cloud Native Core Network Function Specific Security Recommendations and Guidelines

Note

kubectl commands might vary based on the platform deployment. Replace kubectl with Kubernetes environment-specific command line tool to configure Kubernetes resources through kube-api server. The instructions provided in this document are as per the Oracle Communications Cloud Native Core, Cloud Native Environment (CNE) version of kube-api server.

> **Caution**
>
> User, computer and applications, and character encoding settings may cause an issue when copy-pasting commands or any content from PDF. PDF reader version also affects the copy-pasting functionality. It is recommended to verify the pasted content especially when the hyphens or any special characters are part of the copied content.

3.2.1 Network Repository Function (NRF) Specific Security Recommendations and Guidelines

This section provides Network Repository Function (NRF) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- NRF Access Token Secret Configuration
- NRF Access Token Secret Update
- NRF MySQL Secret configuration
  - Kubernetes secret creation for NRF privileged database user
  - Kubernetes secret update for NRF privileged database user
  - Kubernetes secret creation for NRF application database user
  - Kubernetes secret update for NRF application database user
  - Creating Secrets for DNS NAPTR - Alternate route service
- Network Policies

NRF Access Token Secret Configuration

Use the following procedure to create access token secret:

1. Create the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NRF
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for NRF

   Note: Creation of private keys, certificates, and passwords are at the discretion of user.
2. Log in to Bastion Host or server from where kubectl can be executed.
3. Create namespace for the secret by performing the following steps:

a. Verify required namespace already exists in system:

```
$ kubectl get namespaces
```

b. In the output of the above command, check if required namespace is available. If not available, create the namespace using following the command:
Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

```
$ kubectl create namespace <required namespace>
```

Example:

```
$ kubectl create namespace ocnrf
```

4. Create Kubernetes secret for Access token by performing the following steps:
   a. To create Kubernetes secret for HTTPS, following files are required:
      - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NRF
      - RSA private keys for algorithm RS256 and corresponding valid public certificates for NRF

> **Note**
>
> Creation process for private keys, certificates and passwords is based on the discretion of the user or operator. Only unencrypted keys and certificates are supported. PKCS1 and PKCS8 are the only supported versions for RSA, and PKCS8 is the only supported version for ECDSA.

   b. Run the following command to create secret. The names used below are same as provided in custom values.yaml in NRF deployment:

```
$ kubectl create secret generic <ocnrfaccesstoken-secret-name>  --from-
file=<ecdsa_private_key.pem>
 --from-file=<rsa_private_key.pem>  --from-file=<ssl_truststore.txt> --
from-file=<keystore_password.txt>
  --from-file=rsa_certificate.crt --from-file=<ecdsa_certificate.crt> -
n <Namespace of NRF AccessToken secret>
```

Note: Note down the command used during the creation of Kubernetes secret, this command will be used for updates in future.

```
$ kubectl create secret generic ocnrfaccesstoken-secret  --from-
file=ecdsa_private_key.pem
--from-file=rsa_private_key.pem  --from-file=ssl_truststore.txt --from-
file=keystore_password.txt  --from-file=
rsa_certificate.crt --from-file=ecdsa_certificate.crt -n ocnrf
```

c. Run the following command to verify secret created:

```
$ kubectl describe secret <ocnrfaccesstoken-secret-name> -n <Namespace
of NRF AccessToken secret>
```

Example:

```
 $ kubectl describe secret ocnrfaccesstoken-secret -n ocnrf
```

NRF Access Token Secret Update

Use the following procedure to update access token secret:

1. Update the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NRF
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for NRF

   Note: Updating private keys, certificates, and passwords are at the user's discretion.
2. Log in to Bastion Host or server from where kubectl can be executed.
3. Update the secret with new or updated details by performing the following steps:
   a. Copy the exact command used in above section during creation of secret.
   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of Access Token secret>".
   c. Sample format of the create secret command is given below:

```
$ kubectl create secret generic <ocnrfaccesstoken-secret> --from-
file=<ecdsa_private_key.pem>
 --from-file=<rsa_private_key.pem> --from-file=<rsa_certificate.crt> --
from-file=<ecdsa_certificate.crt>
 --dry-run -o yaml -n <Namespace of NRF deployment> | kubectl replace -
f - -n <Namespace of NRF deployment>
```

   Example: The names used below are same as provided in custom_values.yaml in NRF deployment:

```
$ kubectl create secret generic ocnrfaccesstoken-secret --from-
file=ecdsa_private_key.pem
 --from-file=rsa_private_key.pem --from-file=rsa_certificate.crt --from-
file=ecdsa_certificate.crt
--dry-run -o yaml -n ocnrf | kubectl replace -f - -n ocnrf
```

   d. Run the updated command.
   e. After successful secret update, the following message is displayed:

```
secret/<ocnrfaccesstoken-secret> replaced
```

NRF MySQL Secret Configuration

This section describes the secret creation for two types of NRF users. Different users have different sets of permissions.

- NRF privileged user: This user category has the complete set of permissions. The user can perform DDL and DML operations to install, upgrade, roll back or delete operations.
- NRF application user: This user category has fewer permissions and is used by NRF applications during service operations handling. The user can insert, update, get, and remove the records but cannot create, alter, and drop the database and tables.

Kubernetes secret creation for NRF privileged database user

This section explains the steps to create Kubernetes secrets for accessing NRF database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be executed.
2. Create namespace for the secret by performing the following steps:
   a. Verify if required namespace already exists in the system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:
      Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      For example:

      ```
      $ kubectl create namespace ocnrf
      ```

3. Create Kubernetes secret for privileged user as follows:
   a. Create Kubernetes secret for MySQL:

      ```
      $ kubectl create secret generic <privileged user secret name> --from-
      literal=dbUsername=<NRF Privileged Mysql database username> --from-
      literal=dbPassword=<NRF Privileged Mysql User database passsword> --
      from-literal=appDbName=<NRF Mysql database name> --from-
      literal=networkScopedDbName=<NRF Mysql Network database name> --from-
      literal=commonConfigDbName=<NRF Mysql Common Configuration DB> --from-
      literal=leaderElectionDbName=<Perf-Info DB> -n <Namespace of NRF
      deployment>
      ```

      > **Note**
      >
      > Note down the command used during the creation of Kubernetes secret, this command is used for updates in future.

      Example:

      ```
      $ kubectl create secret generic privilegeduser-secret --
      fromliteral=dbUsername=nrfPrivilegedUsr --
      ```

```
fromliteral=dbPassword=nrfPrivilegedPasswd --
fromliteral=appDbName=nrfApplicationDB --
fromliteral=networkScopedDbName=nrfNetworkDB --
fromliteral=commonConfigDbName=commonConfigurationDB --
fromliteral=leaderElectionDbName=leaderElectionDB -n ocnrf
```

b. Verify the secret created using above command:

```
$ kubectl describe secret <database secret name> -n <Namespace of NRF
deployment>
```

Example:

```
$ kubectl describe secret privilegeduser-secret -n ocnrf
```

Kubernetes secret update for NRF privileged database user

This section explains the steps to update Kubernetes secrets for accessing NRF database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be executed.
2. This section describes the steps to update the secrets. Update Kubernetes secret for privileged user as follows:
   a. Copy the exact command used in section during creation of secret:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<NRF Privileged MySQL database username>
--from-literal=dbPassword=<NRF Privileged MySQL database password>
--from-literal=appDbName=<NRF MySQL database name>
--from-literal=networkScopedDbName=<NRF MySQL Network database name>
--from-literal=commonConfigDbName=<NRF MySQL Common Configuration DB> -
n
<Namespace of NRF deployment>
```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<NRF Privileged MySQL database username>
--from-literal=dbPassword=<NRF Privileged MySQL database password>
--from-literal=appDbName=<NRF MySQL database name>
--from-literal=networkScopedDbName=<NRF MySQL Network database name>
--from-literal=commonConfigDbName=<NRF MySQL Common Configuration DB> --
dry-run -o yaml
-n <Namespace of NRF deployment> | kubectl replace -f - -n <Namespace
of NRF deployment>
```

   c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Kubernetes secret creation for NRF application database user

This section explains the steps to create Kubernetes secrets for accessing NRF database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be executed.
2. Create namespace for the secret by performing the following steps:
   a. Verify if required namespace already exists in the system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:
      Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      Example:

      ```
      $ kubectl create namespace ocnrf
      ```

3. Create Kubernetes secret for NRF application database user for configuring records is as follows:
   a. Create Kubernetes secret for NRF application database user:

      ```
      $ kubectl create secret generic <appuser-secret name> --from-
      literal=dbUsername=<NRF APPLICATION User Name> --from-
      literal=dbPassword=<Password for NRF APPLICATION User> --from-
      literal=appDbName=<NRF Application Database> -n <Namespace of NRF
      deployment>
      ```

      > **Note**
      >
      > Note down the command used during the creation of Kubernetes secret, this command will be used for updates in future.

      Example:

      ```
      $ kubectl create secret generic appuser-secret --from-
      literal=dbUsername=nrfApplicationUsr --from-
      literal=dbPassword=nrfApplicationPasswd --from-
      literal=appDbName=nrfApplicationDB -n ocnrf
      ```

   b. Verify the secret creation:

      ```
      $ kubectl describe secret <appuser-secret name> -n <Namespace of NRF
      deployment>
      ```

Example:

```
$ kubectl describe secret appuser-secret -n ocnrf
```

Kubernetes secret update for NRF application database user

This section explains the steps to update Kubernetes secrets for accessing NRF database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be executed.
2. This section explains how to update the Kubernetes secret.
   a. Copy the exact command used in above section during creation of secret:

      ```
      $ kubectl create secret generic <appuser-secret name> --from-
      literal=dbUsername=<NRF APPLICATION
       User Name> --from-literal=dbPassword=<Password for NRF APPLICATION
      User> --from-literal=appDbName=<NRF
      Application Database> -n <Namespace of NRF deployment>
      ```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

      ```
      $ kubectl create secret generic <database secret name> --from-
      literal=dbUsername=<NRF APPLICATION
      User Name> --from-literal=dbPassword=<Password for NRF APPLICATION
      User> --from-literal=appDbName=<NRF
      Application Database> --dry-run -o yaml -n <Namespace of NRF
      deployment> | kubectl replace -f - -n <Namespace
      of NRF deployment>
      ```

   c. Run the updated command. The following message is displayed:

      ```
      secret/<database secret name> replaced
      ```

Creating Secrets for DNS NAPTR - Alternate route service

This section provides information about how to create secret for DNS NAPTR in Alternate Route service.

1. Run the following command to create secret:

   ```
   $ kubectl create secret generic <DNS NAPTR Secret> --from-
   literal=tsigKey=<tsig key generated of DNS Server> --from-
   literal=algorithm=<Algorithm used to generate key> --from-
   literal=keyName=<key-name used while generating key> -n <Namespace of NRF
   deployment>
   ```

> **Note**
>
> Note down the command used during the creation of the secret. Use the command for updating the secrets in the future.

Example:

```
$ kubectl create secret generic tsig-secret  --from-
literal=tsigKey=kUVdLp2SYshV/mkE985LEePLt3/
K4vhM63suWJXA9T6DAl3hJFQQpKAcK5imcIKjI5IVyYk2AJBkq3qtQvRTGw== --from-
literal=algorithm=hmac-sha256 --from-literal=keyName=ocnrf-tsig -n ocnrf
```

2. Run the following command to verify the secret created:

```
$ kubectl describe secret <DNS NAPTR Secret> -n <Namespace of NRF
deployment>
```

Example:

```
$ kubectl describe secret tsig-secret -n ocnrf
```

> **Note**
>
> Creating DNS Server Key is on discretion of the operator.

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Network Repository Function Installation, Upgrade, and Fault Recovery Guide*.

3.2.2 Service Communication Proxy (SCP) Specific Security Recommendations and Guidelines

This section provides Service Communication Proxy Function (SCP) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- SCP MySQL Secret configuration
  - Kubernetes secret creation for SCP privileged database user
  - Kubernetes secret update for SCP privileged database user
  - Kubernetes secret creation for SCP application database user
  - Kubernetes secret update for SCP application database user
- Network Policies

SCP Kubernetes Secret Configuration

The following SCP users have different sets of permissions:

- SCP privileged user: This user category has a complete set of permissions. The user can perform DDL and DML operations to install, upgrade, roll back or delete operations.
- SCP application user: This user category has fewer permissions and is used by SCP applications during service operations handling. The user can insert, update, get, and remove the records. This user cannot create, alter, and drop the database and tables.

> **Note**
>
> Do not use the same credentials in different Kubernetes secrets, and the passwords stored in the secrets must follow the password policy requirements as recommended in #unique_32.

Kubernetes Secret Creation for SCP Privileged Database User

This section explains the steps to create Kubernetes secrets for accessing SCP database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the secret by performing the following steps:
   a. Verify if required namespace already exists in the system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:
      Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

For example:

```
$ kubectl create namespace ocscp
```

3. Create Kubernetes secret for privileged user as follows:
   a. Create Kubernetes secret for MySQL:

```
  $ kubectl create secret generic <privileged user secret name>
 --from-literal=dbUsername=<SCP Privileged Mysql database username>
--from-literal=dbPassword=<SCP Privileged Mysql User database password>
--from-literal=appDbName=<SCP Mysql database name>
--from-literal=networkScopedDbName=<SCP Mysql Network database name>
--from-literal=commonConfigDbName=<SCP Mysql Common Configuration DB> -
n
<Namespace of SCP deployment>
```

> **Note**
>
> Note down the command used during the creation of Kubernetes secret, this command is used for updates in future.

Example:

```
 $ kubectl create secret generic privilegeduser-secret --from-
literal=dbUsername=nrfPrivilegedUsr
--from-literal=dbPassword=nrfPrivilegedPasswd --from-
literal=appDbName=nrfApplicationDB --from-literal
=networkScopedDbName=nrfNetworkDB --from-
literal=commonConfigDbName=commonConfigurationDB -n ocscp
```

   b. Verify the secret created using above command:

```
$ kubectl describe secret <database secret name> -n <Namespace of SCP
deployment>
```

Example:

```
$ kubectl describe secret privilegeduser-secret -n ocscp
```

Kubernetes secret update for SCP privileged database user

This section explains the steps to update Kubernetes secrets for accessing SCP database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. This section describes the steps to update the secrets. Update Kubernetes secret for privileged user as follows:

a. Copy the exact command used in section during creation of secret:

```
   $ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<SCP Privileged Mysql database username>
--from-literal=dbPassword=<SCP Privileged Mysql database password>
--from-literal=appDbName=<SCP Mysql database name>
--from-literal=networkScopedDbName=<SCP Mysql Network database name>
--from-literal=commonConfigDbName=<SCP Mysql Common Configuration DB> -
n
<Namespace of SCP deployment>
```

b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<SCP Privileged Mysql database username>
--from-literal=dbPassword=<SCP Privileged Mysql database password>
--from-literal=appDbName=<SCP Mysql database name>
--from-literal=networkScopedDbName=<SCP Mysql Network database name>
--from-literal=commonConfigDbName=<SCP Mysql Common Configuration DB> --
dry-run -o yaml
-n <Namespace of SCP deployment> | kubectl replace -f - -n <Namespace
of SCP deployment>
```

c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Kubernetes Secret Creation for SCP Application Database User

This section explains the steps to create Kubernetes secrets for accessing SCP database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the secret by performing the following steps:
   a. Verify required namespace already exists in system:

```
 $ kubectl get namespaces
```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:
   Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

```
 $ kubectl create namespace <required namespace>
```

   Example:

```
 $ kubectl create namespace ocscp
```

3. Create Kubernetes secret for SCP application database user for configuring records is as follows:

a. Create Kubernetes secret for SCP application database user:

```
 $ kubectl create secret generic <appuser-secret name> --from-literal=
dbUsername=<SCP APPLICATION User Name> --from-literal=
dbPassword=<Password for SCP APPLICATION User> --from-literal=
appDbName=<SCP Application Database> -n <Namespace of SCP deployment>
```

> **Note**
>
> Note down the command used during the creation of Kubernetes secret, this command will be used for updates in future.

Example:

```
 $ kubectl create secret generic appuser-secret --from-literal
=dbUsername=nrfApplicationUsr --from-literal=dbPassword
=nrfApplicationPasswd --from-literal=appDbName=nrfApplicationDB -n ocscp
```

b. Verify the secret creation:

```
 $ kubectl describe secret <appuser-secret name> -n <Namespace of SCP
deployment>
```

Example:

```
 $ kubectl describe secret appuser-secret -n ocscp
```

Kubernetes Secret Update for SCP Application Database User

This section explains the steps to update Kubernetes secrets for accessing SCP database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. This section explains how to update the Kubernetes secret.
   a. Copy the exact command used in above section during creation of secret:

```
 $ kubectl create secret generic <appuser-secret name> --from-literal
=dbUsername=<SCP APPLICATION
 User Name> --from-literal=dbPassword=<Password for SCP APPLICATION
User> --from-literal=appDbName=<SCP
Application Database> -n <Namespace of SCP deployment>
```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
 $ kubectl create secret generic <database secret name> --from-
literal=dbUsername=<SCP APPLICATION
User Name> --from-literal=dbPassword=<Password for SCP APPLICATION
User> --from-literal=appDbName=<SCP
Application Database> --dry-run -o yaml -n <Namespace of SCP
```

```
deployment> | kubectl replace -f - -n <Namespace
of SCP deployment>
```

c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core,Service Communication Proxy Installation, Upgrade, and Fault Recovery Guide*.

3.2.3 Network Exposure Function (NEF) Specific Security Recommendations and Guidelines

This section provides specific recommendations and guidelines for Network Exposure Function (NEF) security. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- NEF Access Token Secret Configuration
- NEF Access Token Secret Update
- NEF MySQL Secret configuration
  - Kubernetes secret creation for NEF privileged database user
  - Kubernetes secret update for NEF privileged database user
  - Kubernetes secret creation for NEF application database user
  - Kubernetes secret update for NEF application database user

- Network Policies

NEF Access Token Secret Configuration

Use the following procedure to create an Access token secret :

1. Create the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NEF
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for NEF

   Note: Creation of private keys, certificates and passwords are at the discretion of user.
2. Log in to Bastion Host or server from where you can run kubectl commands.
3. Create a namespace for the secret by performing the following steps:
   a. Verify if the required namespace already exists in the system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using the following command:
      Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      Example:

      ```
      $ kubectl create namespace ocnef
      ```

4. Create Kubernetes secret for the Access token by performing the following steps:
   a. To create Kubernetes secret for HTTPS, following files are required:
      - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NEF
      - RSA private keys for algorithm RS256 and corresponding valid public certificates for NEF

      > **Note**
      >
      > Creation process for private keys, certificates and passwords is at the user's or operators discretion. Unencrypted key and certificates is only supported. PKCS1 and PKCS8 are the only supported versions for RSA. PKCS8 is the only supported version for ECDSA.

   b. Run the following command to create secret. The names used below are same as provided in custom values.yaml in NEF deployment:

      ```
      $ kubectl create secret generic <ocnefaccesstoken-secret-name>  --from-file=<ecdsa_private_key.pem>
       --from-file=<rsa_private_key.pem>  --from-file=<ssl_truststore.txt> --from-file=<keystore_password.txt>
      ```

```
  --from-file=rsa_certificate.crt --from-file=<ecdsa_certificate.crt> -
n <Namespace of ocnef AccessToken secret>
```

Note: Note down the command used during the creation of Kubernetes secret, this command will be used for updates in future.

```
$ kubectl create secret generic ocnefaccesstoken-secret  --from-
file=ecdsa_private_key.pem
--from-file=rsa_private_key.pem  --from-file=ssl_truststore.txt --from-
file=keystore_password.txt  --from-file=
rsa_certificate.crt --from-file=ecdsa_certificate.crt -n ocnef
```

c. Run the following command to verify if the secret is created:

```
$ kubectl describe secret <ocnefaccesstoken-secret-name> -n <Namespace
of NEF AccessToken secret>
```

Example:

```
 $ kubectl describe secret ocnefaccesstoken-secret -n ocnef
```

NEF Access Token Secret Update

Use the following procedure to update the Access token secret:

1. Update the following files:
   - ECDSA private keys for algorithm ES256 and corresponding valid public certificates for NEF
   - RSA private keys for algorithm RS256 and corresponding valid public certificates for NEF

   Note: Update of private keys, certificates and passwords are at the discretion of user.
2. Log in to Bastion Host or server from where you can run kubectl commands.
3. Update the secret with new or updated details by performing the following steps:
   a. Copy the exact command used in above section during creation of secret.
   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of Access Token secret>".
   c. Create secret command must look like:

```
$ kubectl create secret generic <ocnefaccesstoken-secret> --from-
file=<ecdsa_private_key.pem>
--from-file=<rsa_private_key.pem> --from-file=<rsa_certificate.crt> --
from-file=<ecdsa_certificate.crt>
 --dry-run -o yaml -n <Namespace of ocnef deployment> | kubectl replace
-f - -n <Namespace of ocnef deployment>
```

Example: The names used below are same as provided in custom_values.yaml in NEF deployment:

```
$ kubectl create secret generic ocnefaccesstoken-secret --from-
file=ecdsa_private_key.pem
--from-file=rsa_private_key.pem --from-file=rsa_certificate.crt --from-
file=ecdsa_certificate.crt
--dry-run -o yaml -n ocnef | kubectl replace -f - -n ocnef
```

d. Run the updated command.

e. After successful secret update, the following message is displayed:

```
secret/<ocnefaccesstoken-secret> replaced
```

NEF MySQL Secret Configuration

This section describes the secret creation for two types of NEF users. Different users have different sets of permissions.

- NEF privileged user: This user category has a complete set of permissions. The user can perform DDL and DML operations to install, upgrade, roll back or delete operations.
- NEF application user: This user category has fewer sets of permissions and is used by NEF applications during service operations handling. This user cannot create, alter, and drop the database and tables.

Kubernetes secret creation for NEF privileged database user

This section explains the steps to create Kubernetes secrets for accessing NEF database for the privileged user.

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Create a namespace for the secret by performing the following steps:
   a. Verify if the required namespace already exists in the system:

   ```
   $ kubectl get namespaces
   ```

   b. In the output of the above command, check if the required namespace is available. If not available, create the namespace using the following command:
   Note: This is an optional step. In case the required namespace already exists, proceed with the next set of procedures.

   ```
   $ kubectl create namespace <required namespace>
   ```

   For example:

   ```
   $ kubectl create namespace ocnef
   ```

3. Create a Kubernetes secret for privileged user as follows:
   a. Create a Kubernetes secret for MySQL:

   ```
   $ kubectl create secret generic <privileged user secret name>
   --from-literal=dbUsername=<NEF Privileged MySQL database username>
   ```

```
--from-literal=dbPassword=<NEF Privileged MySQL User database password>
--from-literal=appDbName=<NEF MySQL database name>
--from-literal=networkScopedDbName=<NEF MySQL Network database name>
--from-literal=commonConfigDbName=<NEF MySQL Common Configuration DB> -
n
<Namespace of NEF deployment>
```

> **Note**
>
> Note down the command used during the creation of the Kubernetes secret; this command is used for updates in the future.

Example:

```
$ kubectl create secret generic privilegeduser-secret --from-
literal=dbUsername=ocnefPrivilegedUsr
--from-literal=dbPassword=ocnefPrivilegedPasswd --from-
literal=appDbName=ocnefApplicationDB --from-literal
=networkScopedDbName=ocnefNetworkDB --from-
literal=commonConfigDbName=commonConfigurationDB -n ocnef
```

b. Verify the secret created using above command:

```
$ kubectl describe secret <database secret name> -n <Namespace of NEF
deployment>
```

Example:

```
$ kubectl describe secret privilegeduser-secret -n ocnef
```

Kubernetes secret update for NEF privileged database user

This section explains the steps to update Kubernetes secrets for accessing NEF database for the privileged user.

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. This section describes the steps to update the secrets. Update Kubernetes secret for privileged user as follows:
   a. Copy the exact command used in section during creation of secret:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<NEF Privileged MySQL database username>
--from-literal=dbPassword=<NEF Privileged MySQL database password>
--from-literal=appDbName=<NEF MySQL database name>
--from-literal=networkScopedDbName=<NEF MySQL Network database name>
--from-literal=commonConfigDbName=<NEF MySQL Common Configuration DB> -
n
<Namespace of NEF deployment>
```

b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
$ kubectl create secret generic <privileged user secret name>
--from-literal=dbUsername=<NEF Privileged MySQL database username>
--from-literal=dbPassword=<NEF Privileged MySQL database password>
--from-literal=appDbName=<NEF MySQL database name>
--from-literal=networkScopedDbName=<NEF MySQL Network database name>
--from-literal=commonConfigDbName=<NEF MySQL Common Configuration DB> --
dry-run -o yaml
-n <Namespace of NEF deployment> | kubectl replace -f - -n <Namespace
of NEF deployment>
```

c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Kubernetes secret creation for NEF application database user

This section explains the steps to create Kubernetes secrets for accessing NEF database for the application database user.

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Create a namespace for the secret by performing the following steps:
   a. Verify if the required namespace already exists in the system:

   ```
   $ kubectl get namespaces
   ```

   b. In the output of the above command, check if required the namespace is available. If not available, create the namespace using the following command:
   Note: This is an optional step. In case the required namespace already exists, proceed with the next set of procedures.

   ```
   $ kubectl create namespace <required namespace>
   ```

   Example:

   ```
   $ kubectl create namespace ocnef
   ```

3. Create a Kubernetes secret for NEF application database user for configuring records as follows:
   a. Create a Kubernetes secret for NEF application database user:

   ```
   $ kubectl create secret generic <appuser-secret name> --from-
   literal=dbUsername=<NEF APPLICATION User Name> --from-
   literal=dbPassword=<Password for NEF APPLICATION User> --from-
   literal=appDbName=<NEF Application Database> -n <Namespace of NEF
   deployment>
   ```

> **Note**
>
> Note down the command used during the creation of Kubernetes secret, this command will be used for updates in future.

Example:

```
$ kubectl create secret generic appuser-secret --from-
literal=dbUsername=NEFApplicationUsr --from-
literal=dbPassword=NEFApplicationPasswd --from-
literal=appDbName=NEFApplicationDB -n ocnef
```

b. Verify the secret creation:

```
$ kubectl describe secret <appuser-secret name> -n <Namespace of NEF
deployment>
```

Example:

```
$ kubectl describe secret appuser-secret -n ocnef
```

Kubernetes secret update for NEF application database user

This section explains the steps to update Kubernetes secrets for accessing NEF database for the application database user.

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. This section explains how you can update the Kubernetes secret.
   a. Copy the exact command used in above section during creation of secret:

```
$ kubectl create secret generic <appuser-secret name> --from-
literal=dbUsername=<NEF APPLICATION
 User Name> --from-literal=dbPassword=<Password for NEF APPLICATION
User> --from-literal=appDbName=<NEF
Application Database> -n <Namespace of NEF deployment>
```

   b. Update the same command with string "--dry-run -o yaml" and "kubectl replace -f - -n <Namespace of MySQL secret>". After update, the command will be as follows:

```
$ kubectl create secret generic <database secret name> --from-
literal=dbUsername=<NEF APPLICATION
User Name> --from-literal=dbPassword=<Password for NEF APPLICATION
User> --from-literal=appDbName=<NEF
Application Database> --dry-run -o yaml -n <Namespace of NEF
deployment> | kubectl replace -f - -n <Namespace
of NEF deployment>
```

   c. Run the updated command. The following message is displayed:

```
secret/<database secret name> replaced
```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Network Exposure Function Installation, Upgrade, and Fault Recovery Guide.*

3.2.4 Network Slice Selection Function (NSSF) Specific Security Recommendations and Guidelines

This section provides Network Slice Selection Function (NSSF) specific security recommendations and guidelines. Recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- NSSF Access Token Secret Configuration
- NSSF Access Token Secret Update
- NSSF MySQL Secret Configuration
  - Kubernetes Secret Creation for NSSF Privileged Database User
  - Kubernetes Secret Update for NSSF Privileged Database User
  - Kubernetes Secret Creation for NSSF Application Database User
  - Kubernetes Secret Update for NSSF Application Database User
- Network Policies

NSSF Access Token Secret Configuration

Use the following procedure to create access token secret:

1. Create the following files:

- ECDSA private key (Example: `ecdsa_private_key_pkcs8.pem`)
- RSA private key (Example: `rsa_private_key_pkcs1.pem`)
- TrustStore password file (Example: `trustStorePassword.txt`)
- KeyStore password file (Example: `keyStorePassword.txt`)
- CA signed ECDSA NSSF certificate (Example: `ecdsa_ocnssf_certificate.crt`)
- CA signed RSA NSSF certificate (Example: `rsa_ocnssf_certificate.crt`)

Note: Creation of private keys, certificates and passwords are at the discretion of user.

2. Log in to Bastion Host or server from where kubectl can be run.
3. Create namespace for the secret by executing the following command:
```
$ kubectl create namespace ocnssf
```
4. Create Kubernetes secret for NF Access token by executing the following command:

```
$ kubectl create secret generic
         ocnssfaccesstoken-secret --from-file=ecdsa_private_key_pkcs8.pem
         --from-file=rsa_private_key_pkcs1.pem --from-
file=trustStorePassword.txt
         --from-file=keyStorePassword.txt --from-
file=ecdsa_ocnssf_certificate.crt--from-file=rsa_ocnssf_certificate.crt -n
      ocnssf
```

5. Verify that secret is created successfully by executing the following command:
```
$ kubectl describe secret ocnssfaccesstoken-secret -n ocnssf
```

NSSF Access Token Secret Update

Use the following procedure to update access token secret:

1. Update the following files:
   - ECDSA private key (Example: `ecdsa_private_key_pkcs8.pem`)
   - RSA private key (Example: `rsa_private_key_pkcs1.pem`)
   - TrustStore password file (Example: `trustStorePassword.txt`)
   - KeyStore password file (Example: `keyStorePassword.txt`)
   - CA signed ECDSA NSSF certificate (Example: `ecdsa_ocnssf_certificate.crt`)
   - CA signed RSA NSSF certificate (Example: `rsa_ocnssf_certificate.crt`)

   Note: Update private keys, certificates, and passwords are at the user's discretion.
2. Log in to Bastion Host or server from where kubectl can be run.
3. Update the secret with new or updated details by executing the following commands:
   Delete the secret:

   ```
   $ kubectl delete secret ocnssfaccesstoken-secret -n ocnssf
   ```

   Create the secret again with updated details:

   ```
   $ kubectl create secret generic ocnssfaccesstoken-secret --from-
   file=ecdsa_private_key_pkcs8.pem
    --from-file=rsa_private_key_pkcs1.pem --from-file=trustStorePassword.txt
   ```

```
--from-file=keyStorePassword.txt
--from-file=ecdsa_ocnssf_certificate.crt--from-
file=rsa_ocnssf_certificate.crt -n ocnssf
```

NSSF MySQL Secret Configuration

Kubernetes Secret Creation for NSSF Privileged Database User

This section explains the steps to create Kubernetes secrets for accessing NSSF database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the secret by following:
   a. Verify required namespace already exists in system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using following command:
      Note: This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      For example:

      ```
      $ kubectl create namespace ocnssf
      ```

3. Create a yaml file with the username and password with the syntax as follows:

```
apiVersion: v1
kind: Secret
metadata:
name: <secret-name>
type: Opaque
data:
mysql-username: cm9vdA==
mysql-password: cm9vdHBhc3N3ZA==
```

> **Note**
>
> The values for "mysql-username" and "mysql-password" must be Base64 encoded.

4. Run `kubectl create -f <yaml_file_name> -n <namespace>` to create the secret.
5. Verify whether the secret is created by running the following command:
   `$ kubectl describe secret <secret-name> -n <namespace>`

Kubernetes Secret Update For NSSF Privileged Database User

This section explains the steps to update Kubernetes secrets for accessing NSSF database for the privileged user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Delete the Kubernetes secret for MySQL:

   ```
   # Delete the secret
   $ kubectl delete secret <secret name> -n <namespace>
   ```

3. Update yaml file from step 3 in secret creation with new values for MySQL-username and MySQL-password
4. Run `kubectl create -f <yaml_file_name> -n <namespace>` to create the secret.
5. Verify whether the secret is created by running the following command:
   `$ kubectl describe secret <secret-name> -n <namespace>`

Kubernetes Secret Creation for NSSF Application Database User

This section explains the steps to create Kubernetes secrets for accessing NSSF database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the secret by following:
   a. Verify required namespace already exists in system:

      ```
      $ kubectl get namespaces
      ```

   b. In the output of the above command, check if required namespace is available. If not available, create the namespace using following command:

      > **Note**
      >
      > This is an optional step. In case required namespace already exists, proceed with next procedures.

      ```
      $ kubectl create namespace <required namespace>
      ```

      For example:

      ```
      $ kubectl create namespace ocnssf
      ```

3. Create a yaml file with the username and password with the syntax as follows:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   name: <secret-name>
   type: Opaque
   data:
   mysql-username: bnNzZnVzZXI=
   mysql-password: bnNzZnBhc3N3ZA==
   ```

> **Note**
>
> The values for "mysql-username" and "mysql-password" must be Base64 encoded.

4. Run `kubectl create -f <yaml_file_name> -n <namespace>` to create the secret.
5. Verify whether the secret is created by running the following command:
   `$ kubectl describe secret <secret-name> -n <namespace>`

Kubernetes Secret Update for NSSF Application Database User

This section explains the steps to update Kubernetes secrets for accessing NSSF database for the application database user.

1. Log in to Bastion Host or server from where kubectl can be run.
2. Delete the Kubernetes secret for MySQL:

   ```
   # Delete the secret
   $ kubectl delete secret <secret name> -n <namespace>
   ```

3. Update yaml file from step 3 in secret creation with new values for MySQL-username and MySQL-password
4. Run `kubectl create -f <yaml_file_name> -n <namespace>` to create the secret.
5. Verify whether the secret is created by running the following command:
   `$ kubectl describe secret <secret-name> -n <namespace>`

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Network Slice Selection Function Installation, Upgrade, and Fault Recovery Guide.*

.

3.2.5 Security Edge Protection Proxy (SEPP) Security Recommendations and Procedures

This section provides Security Edge Protection Proxy (SEPP) specific security recommendations and procedures. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- SEPP Secret Configuration for HTTPS and HTTP over TLS
  - Creating Secret for HTTPS and HTTP over TLS
  - Updating Access Token Secret
- SEPP MySQL Secret Configuration
  - Creating Secret for MySQL
  - Updating Secret for MySQL
- Network Policies

SEPP Secret Configuration for HTTPS and HTTP over TLS

Use the following procedures to configure secret for HTTPS and HTTP over TLS and to update Access Token Secret:

Creating Secret for HTTPS and HTTP over TLS

Use the following procedure to create secret for HTTPS and HTTP over TLS:

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Create the following files:
   - RSA or ECDSA Private key (For example: `rsa_private_key_pkcs1.pem`)
   - Truststore password (For example: `trust.txt`)
   - Key store password (For example: `key.txt`)
   - Certificate chain for truststore (For example: `caroot.cer`)
   - Signed server certificate (For example: `ocsepp.cer`) or Signed client certificate (For example: `ocsepp.cer`)

   Note: Creation of private keys, certificates, and passwords is at the discretion of the user.
3. To verify and create the Kubernetes namespace, do the following:
   a. Run the following command to verify if the required namespace exists in the system:

      ```
      $ kubectl get namespaces
      ```

b. Run the following command to create a Kubernetes namespace if the output of the above command does not display the required namespace:

```
$ kubectl create namespace <required namespace>
```

> **Note**
>
> This is an optional step. In case the required namespace already exists, skip this procedure.

Example:

```
$ kubectl create namespace seppsvc
```

4. Run the following commands to create Kubernetes secrets:

- For **Creating secrets**

```
$ kubectl create secret generic <secret-name>
                  --from-file=<ssl_ecdsa_private_key.pem> --from-
file=<rsa_private_key_pkcs1.pem>
                  --from-file=<ssl_truststore.txt> --from-
file=<ssl_keystore.txt> --from-file=<signed.cer>
                  --from-file=<caroot.cer> --from-
file=<ssl_rsa_certificate.crt> --from-file
                  <ssl_ecdsa_certificate.crt> -n <Namespace of SEPP
deployment>
```

- For **HTTP over TLS => For n32 interface**

```
$ kubectl create secret generic ocsepp-n32-secret
                  --from-file=rsa_private_key_pkcs1.pem --from-
file=trust.txt --from-file=key.txt
                  --from-file=caroot.cer --from-
file=rsa_certificate.crt --from-file=ocsepp.cer -n
                  seppsvc
```

- For **HTTPS => For Plmn interface**

```
$ kubectl create secret generic
                    ocsepp-plmn-secret --from-
file=rsa_private_key_pkcs1.pem --from-file=trust.txt
                    --from-file=key.txt --from-file=caroot.cer --from-
file=rsa_certificate.crt
                    --from-file=ocsepp.cer -n seppsvc
```

Updating Access Token Secret

Use the following procedure to update access token secret:

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Update the secret with new or updated details.
   Run the following commands to create the secrets again with updated details:

- for **HTTP over TLS For n32 interface**

```
$ kubectl create secret generic ocsepp-n32-secret --from-
file=rsa_private_key_pkcs1.pem
      --from-file=trust.txt --from-file=key.txt --from-file=caroot.cer
      --from-file=rsa_certificate.crt --from-file=ocsepp.cer --dry-run -
o yaml  -n seppsvc | kubectl replace -f - -n seppsvc
```

- for **HTTPS For PLMN interface**

```
$ kubectl create secret generic ocsepp-plmn-secret --from-
file=rsa_private_key_pkcs1.pem
      --from-file=trust.txt --from-file=key.txt --from-file=caroot.cer
      --from-file=rsa_certificate.crt --from-file=ocsepp.cer --dry-run -
o yaml  -n seppsvc |
      kubectl replace -f - -n seppsvc
```

Note: Update of private keys, certificates and passwords are at the discretion of the user.

SEPP MySQL Secret Configuration

Creating Secret for MySQL

Use the following procedure to create MySQL Secret:

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Create namespace for the secret. Skip this step, if already created.

```
$ kubectl create namespace seppsvc
```

Note: Creation of private keys, certificates and passwords are at the discretion of the user.

3. Create a yaml file with the username, password, and DB name with the syntax shown below:

```
apiVersion: v1
kind: Secret
metadata:
name: ocsepp-mysql-cred
type: Opaque
data:
mysql-username: c2VwcF91c3I=
mysql-password: RHVrdzFAbT8=
dbName: c2VwcGRi
```

> **Note**
>
> Note: The values for "mysql-username", "mysql-password" and dbName should be Base64 encoded.

4. Run the following commands to create Kubernetes secrets:

```
$ kubectl apply -f <yaml_file_name> -n <namespace>
```

Or

```
kubectl create secret generic ocsepp-mysql-cred --from-literal=mysql-
username='<USR_NAME>'
--from-literal=mysql-password='<PWD>' --from-literal=dbName='<Db Name>' -n
seppsvc
```

5. Verify the secret creation:

```
$ kubectl describe secret <secret-name> -n <namespace>
```

Updating Secret for MySQL

Use the following procedure to update MySQL Secret :

1. Log in to Bastion Host or server from where you can run kubectl commands.
2. Update the Kubernetes secret for MySQL:

```
# Delete the secret:

$ kubectl delete secret database-secret -n <namespace>
```

```
# Create the secret with updated details:

$ kubectl create secret generic <secretName> -from-literal=mysql-
username='<USR_NAME>'
--from-literal=mysql-password='<PWD>' --from-literal=dbName='<Db Name>' -n
<namespace>
```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Security Edge Protection Proxy Installation, Upgrade, and Fault Recovery Guide*.

3.2.6 Unified Data Repository (UDR) and Unstructured Data Storage Function (UDSF) Specific Security Recommendations and Guidelines

This section provides Unified Data Repository (UDR), Unstructured Data Storage Function (UDSF), and Equipment Identity Register (EIR) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines Section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- Oauth Token Validation Configuration
  - Rest Configuration
  - Public key Update for Changed Access Token
  - Disabling the Signature Validation for Oauth
- UDR MySQL Kubernetes secret for storing Database Username and Password
- TLS Certificate for HTTPs Support
- Remote File Transfer Support
- Network Policies

Oauth Token Validation Configuration

Use the following procedure for Oauth Token validation configuration:

1. NRF creates access tokens using following private keys:
   - **ECDSA private key**
     Example:

     ```
     ecdsa_private_key_pkcs8.pem
     ```

   - **RSA private key**
     Example:

     ```
     rsa_private_key_pkcs1.pem
     ```

     In order to validate access token secret needs to be created and configured in ocudr ingress gateway with certificates fetched from nrf.

Example:

```
6faf1bbc-6e4a-4454-a507-a14ef8e1bc5c_ES256.crt
```

2. Log in to Bastion Host or server from where kubectl can be executed.
3. Create namespace for the secret.

```
$ kubectl create namespace ocudr
```

4. Create Kubernetes secret for NF Access token validation

> **Note**
>
> The file names in below command are same as in Step 1.

```
$ kubectl create secret generic oauthsecret --from-file=6faf1bbc-6e4a-4454-
a507-a14ef8e1bc5c_ES256.crt-n ocudr
```

5. Run the following command to verify if the secret is created successfully:

```
$ kubectl describe secret oauthsecret -n ocudr
```

Rest Configuration

We need REST based configurations to distinguish certificates configured from different NRF and use them properly to validate token received from specific NRF. These configurations can be added from CNCC GUI which internally uses config API and payload as below:

```
“/udr/nf-common-component/v1/igw/oauthvalidatorconfiguration”

Payload:

{
              "keyIdList": [{

                            "keyId": "664b344e74294c8fa5d2e7dfaaaba407",
                            "kSecretName": "samplesecret1",
                            "certName": "samplecert1.crt",
                            "certAlgorithm": "ES256"
                          }],
              "instanceIdList": [{

                            "instanceId": "664b344e74294c8fa5d2e7dfaaaba407",
                            "kSecretName": "samplesecret2",
                            "certName": "samplecert2.crt",
                            "certAlgorithm": "ES256"

                             }],
              "oAuthValidationMode": "INSTANCEID_ONLY"

}
```

The multiple **keyId** and **instanceId** object of different NRFs can be configured.

Using **oAuthValidationMode** mode of validation can be selected.

Example: INSTANCEID_ONLY, KID_ONLY or KID_PREFERRED

KID_PREFERRED is a fall back mode where it checks for keyId in token, if token contains keyId then validation mode is KID_ONLY or else it falls back to INSTANCEID_ONLY.

Public key Update for Changed Access Token

Use the following procedure for public key update for changed access token:

1. Log in to Bastion Host or server from where kubectl can be executed.
2. Update the secret with new or updated details:

```
# Delete the secret and recreate it
$ kubectl delete secret oauthsecret -n ocudr

# Fetch updated certificates from nrf

# Recreate the secret with updated details
$ kubectl create secret generic oauthsecret --from-file=0263663c-
f5c2-4d1b-9170-f7b1a9116337_ES256.crt

-n ocudr
```

3. Certificate configuration update request needs to be sent using CNCC GUI with the updated **keyIdList** and **instanceIdList** with new certificates.

Disabling the Signature Validation for Oauth

If **serviceMeshCheck** flag is enabled under ingress gateway in custom-values file, signature validation is disabled by default.

In this case, only header and payload are validated, and request is successful even if token has wrong signature.

UDR MySQL Kubernetes Secret for storing Database Username and Password

Use the following procedure to create MySQL Kubernetes secret for storing database username and password:

1. Log in to Bastion Host or server from where kubectl can be executed.
2. Create namespace for the MySQL secret. Skip this step, if already created.

```
 $ kubectl create namespace <namespace>
```

3. Run the following command for creating the secret:

```
kubectl create secret generic ocudr-secrets --from-literal=
dbname=<dbname> --from-literal=configdbname=<configdbname> --from-literal=
privilegedUsername=<privilegedUsername> --from-literal=
privilegedPassword=<privilegedPassword> --from-
literal=dsusername=<udruserName> --from-literal=
```

```
dspassword=<udruserPassword> --from-literal=encryptionKey='My secret
passphrase' -n <ocudr-namespace>
```

Example:

```
kubectl create secret generic ocudr-secrets --from-literal=dbname=udrdb --
from-literal=
configdbname=udrconfigdb --from-literal=privilegedUsername=root --from-
literal=
privilegedPassword=rootPasswd --from-literal=dsusername=udruser --from-
literal=dspassword=udrpasswd --from-literal=
encryptionKey='My secret passphrase' -n <ocudr-namespace>
```

4. Verify the whether the secret is created by executing the following command:

```
$ kubectl describe secret <secret-name> -n <namespace>
```

TLS certificate for HTTPs support

UDR and EIR has two Ingress Gateway services to handle the signaling and provisioning traffic. Hence, you must configure two separate TLS certificates to support HTTPS on both the gateways.

For information on the procedure to enable TLS certificates, see Cloud Native Core - Ingress/ Egress Gateways - Security Recommendations / Guidelines for TLS configuration.

Updating Keys and Certificates in the Existing Secrets

Prerequsite: The certificates and files that need to be updated must be present in the secret.

Perform the following steps to update the existing certificates in secrets:

1. Run the following command to add a certificate:

```
TLS_CRT=$(base64 < "<certificate-name>" | tr -d '\n')
kubectl patch secret <secret-name> -p "{\"data\":{\"<certificatename>\":\"$
{TLS_CRT}\"}}"
```

Here,

<certificate-name> is the certificate file name.

<secret-name> is the name of the secret, for example, `ocudr-gateway-secret`.

Example:

Run the following command to add a Certificate Authority (CA) Root from the `caroot.cer` file to the `ocudr-gateway-secret`.

```
TLS_CRT=$(base64 < "caroot.cer" | tr -d '\n')
kubectl patch secret ocudr-gateway-secret -p "{\"data\":{\"caroot.cer\":\"$
{TLS_CRT}\"}}" -n udr
```

Similarly, you can also add other certificates and keys to the `ocudr-gateway-secret`.

2. Run the following command to update an existing certificate:

```
TLS_CRT=$(base64 < "<updated-certificate-name>" | tr -d '\n')
kubectl patch secret <secret-name> -p "{\"data\":{\"<certificatename>\":\"$
{TLS_CRT}\"}}"
```

Here,

`<updated-certificate-name>` is the certificate file that contains the updated content.

Example:

Run the following command to update the private key present in the `rsa_private_key_pkcs1.pem` file to the `ocudr-gateway-secret`:

```
TLS_CRT=$(base64 < "rsa_private_key_pkcs1.pem" | tr -d '\n')
kubectl patch secret ocudr-gateway-secret -p "{\"data\":
{\"rsa_private_key_pkcs1.pem\":\"${TLS_CRT}\"}}" -n udr
```

Similarly, you can also update other certificates and keys to the `ocudr-gateway-secret`.

3. Run the following command to remove an existing certificate:

```
kubectl patch secret <secret-name> -p "{\"data\":
{\"<certificatename>\":null}}"
```

Here,

<certificate-name> is the name of the certificate to be removed.

The certificate must be removed when it expires or needs to be revoked.

Example:

Run the following command to remove the CA Root from the ocudr-gateway-secret:

```
kubectl patch secret ocudr-gateway-secret -p "{\"data\":
{\"caroot.cer\":null}}" -n udr
```

Similarly, you can also remove other certificates and keys from the `ocudr-gateway-secret`.

> **Note**
>
> The following are the certificate update and renewal impacts:
>
> - Updating, adding, deleting the certificate, ot terminates all the existing connections gracefully and re-establishes new connections for new requests.
> - When the certificates expires, no new connections are established for new requests, however, the existing connections remain active. After the renewal of the certificates all the existing connections are gracefully terminated. And, new connections are established with the renewed certificates.

Remote File Transfer Support

UDR supports the transfer of files to the remote sever using Secure File Transfer Protocol (SFTP) in the subscriber bulk import tool and the subscriber export tool as below:

- In subscriber bulk import tool the files will be transferred from the remote server to Persistent Volume Claim (PVC) and vice versa using SFTP
- In subscriber export tool the files will be transferred from PVC to the remote server using SFTP

To support the file transfer, you must run the below command to configure the private and public keys in to the kuberenetes secrets. The operator will get the private and public keys from the remote server.

Keys

```
kubectl create secret generic ocudr-ssh-private-key --from-file=id_rsa=/home/
cloud-user/ocudr/secrets/id_rsa -n <namespace>
kubectl create secret generic ocudr-ssh-public-key --from-file=id_rsa.pub=/
home/cloud-user/ocudr/secrets/id_rsa.pub -n <namespace>
```

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Unified Data Repository Installation, Upgrade, and Fault Recovery Guide* and *Oracle Communications Cloud Native Core, Unified Data Repository User Guide*.

3.2.7 Binding Support Function (BSF) Specific Security Recommendations and Guidelines

This section provides Binding Support Function (BSF) specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines Section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedure is:

- Creating BSF MySQL Kubernetes Secret for Storing Database Username and Password
- Network Policies

Creating BSF MySQL Kubernetes Secret for Storing Database Username and Password

Use the following procedure to create BSF MySQL Kubernetes secret for storing database username and password:

1. Log in to Bastion Host or server from where kubectl can be Run.
2. Create namespace, if already does not exists, by entering the command:

   ```
   kubectl create namespace <namespace>
   ```

   where:

   `<namespace>` is the deployment BSF namespace.
3. Create a Kubernetes secret for an admin user and an application user. To create a Kubernetes secret for storing database username and password for these users follow the procedure below:

   Create a YAML file with the application user's username and password with the syntax shown below:

   > **Note**
   >
   > The values mentioned in the syntax are sample values.

   ```
   apiVersion: v1
               kind: Secret
               metadata:
               name: <secret-name>
               type: Opaque
               data:
               mysql-username: YnNmdXNy
               mysql-password: YnNmcGFzc3dk
   ```

   Create a YAML file with the admin user's username and password with the syntax shown below:

> **Note**
>
> The values mentioned in the syntax are sample values.

```
apiVersion: v1
          kind: Secret
          metadata:
          name: <secret-name>
          type: Opaque
          data:
           mysql-username: YnNmcHJpdmlsZWdlZHVzcg==
           mysql-password: YnNmcHJpdmlsZWdlZHBhc3N3ZA==
```

> **Note**
>
> The values for **mysql-username** and **mysql-password** should be Base64 encoded

4. Run the following command to create the secret:

   ```
   kubectl create -f <yaml_file_name> -n <namespace>
   ```

5. Verify whether the secret is created by executing the following command:

   ```
   $ kubectl describe secret <secret-name> -n <namespace>
   ```

For more information, see cnDBTier Authentication and Authorization.

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Binding Support Function Installation, Upgrade, and Fault Recovery Guide*.

3.2.8 Cloud Native Core Policy Specific Security Recommendations and Guidelines

This section provides Cloud Native Core Policy specific security recommendations and guidelines. Security recommendations common to all 4G and 5G NFs are available in the Common Security Recommendations and Guidelines Section.

> **Note**
>
> The following procedures can be performed by any authenticated user who has privileged access to the system. This user can create different roles for specific operations. For creation of role and role binding, see the NF or component-specific Installation and Upgrade Guide.

The procedures are:

- Access Token configuration
- Update Keys to Sign JSON Web Token (JWTs) for Access Token
- Create CNC Policy MySQL Kubernetes Secret for Storing Database Username and Password for Admin and Application Users
- Create a Kubernetes Secret for Storing LDAP credentials
- Network Policies

Access Token configuration

Use the following procedure to create access token :

1. Create following files:
   - ECDSA private key (Example: `ecdsa_private_key_pkcs8.pem`)
   - RSA private key (Example: `rsa_private_key_pkcs1.pem`)
   - TrustStore password file (Example: `trustStorePassword.txt`)
   - KeyStore password file (Example: `keyStorePassword.txt`)
   - CA signed ECDSA OCPolicy certificate (Example: `ecdsa_occnp_certificate.crt`)
   - CA signed RSA OCPolicy certificate (Example: `rsa_occnp_certificate.crt`)
2. Log in to Bastion Host or server from where kubectl can be run.
3. Create namespace for the secret:

   ```
   $ kubectl create namespace occnp
   ```

4. Create Kubernetes secret for NF Access token :
   Note: The filenames in below command are same as in Step 1

   ```
   $ kubectl create secret generic ocpcfaccesstoken-secret --from-file=
   ecdsa_private_key_pkcs8.pem --from-file=rsa_private_key_pkcs1.pem --from-
   file=
   ```

```
trustStorePassword.txt --from-file=keyStorePassword.txt --from-file=
ecdsa_ocpcf_certificate.crt--from-file=rsa_ocpcf_certificate.crt -n ocpcf
```

5. Verify that secret is created successfully:

```
$ kubectl describe secret ocpcfaccesstoken-secret -n ocpcf
```

Update Keys to Sign JSON Web Token (JWTs) for Access Token

Use the following procedure to update keys to sign JSON web token (JWTs) for access token:

1. Update the following files:
   - ECDSA private key (Example: `ecdsa_private_key_pkcs8.pem`)
   - RSA private key (Example: `rsa_private_key_pkcs1.pem`)
   - CA signed ECDSA OCPolicy certificate (Example: `ecdsa_occnp_certificate.crt`)
   - CA signed RSA OCPolicy certificate (Example: `rsa_occnp_certificate.crt`)

   > **Note**
   >
   > Update of private keys, certificates and passwords are at the discretion of user

2. Log in to Bastion host or server from where kubectl can be run.
3. Update the secret with new or updated details by performing the following steps:
   - Delete the secret by executing the following command:

     ```
     $ kubectl delete secret ocpcfaccesstoken-secret -n ocpcf
     ```

   - Create the secret with updated details:

     ```
     $ kubectl create secret generic ocpcfaccesstoken-secret
                 --from-file=ecdsa_private_key_pkcs8.pem --from-
     file=rsa_private_key_pkcs1.pem
                 --from-file=trustStorePassword.txt --from-
     file=keyStorePassword.txt
                 --from-file=ecdsa_occnp_certificate.crt--from-
     file=rsa_ocpcf_certificate.crt -n
                 occnp
     ```

Create CNC Policy MySQL Kubernetes Secret for Storing Database Username and Password for Admin and Application Users

Use the following procedure to create OCPolicy MySQL Kubernetes secret for storing database username and password:

1. Log in to Bastion Host or server from where kubectl can be run.
2. Create namespace for the MySQL secret. Skip this step, if already created.

```
$ kubectl create namespace <namespace>
```

3. To create a Kubernetes secret for storing database username and password for an admin user and an application user:
   a. Create a YAML file with the application user's username and password with the syntax shown below:

      Note

      The values mentioned in the syntax are sample values.

```
apiVersion: v1
kind: Secret
metadata:
  name: occnp-db-pass
type: Opaque
data:
  mysql-username: b2NjbnB1c3I=
  mysql-password: b2NjbnBwYXNzd2Q=
```

   b. Create a YAML file with the admin user's username and password with the syntax shown below:

      Note

      The values mentioned in the syntax are sample values.

```
apiVersion: v1
kind: Secret
metadata:
  name: occnp-admin-db-pass
type: Opaque
data:
  mysql-username: b2NjbnBhZG1pbnVzcg==
  mysql-password: b2NjbnBhZG1pbnBhc3N3ZA==

'
```

      Note

      `name` will be used to contain `dbCredSecretName` and `privilegedDbCredSecretName` parameters in the CNC Policy custom-values.yaml file.

      Note

      The values for `mysql-username` and `mysql-password` should be Base64 encoded.

c. Run the following commands to add the Kubernetes secrets in a namespace:

```
kubectl create -f yaml_file_name1 -n release_namespace
kubectl create -f yaml_file_name2 -n release_namespace
```

where:

- `release_namespace` is the deployment namespace used by the helm command.
- `yaml_file_name1` is a name of the YAML file that is created in step a.
- `yaml_file_name2` is a name of the YAML file that is created in step b.

4. Verify whether the secret is created by executing the following command:

```
$ kubectl describe secret <secret-name> -n <namespace>
```

For more information, see cnDBTier Authentication and Authorization.

Create a Kubernetes Secret for Storing LDAP credentials

Use the following procedure to create a Kubernetes secret for storing LDAP credentials:

1. Create a YAML file with the following syntax:

> **Note**
>
> The values mentioned in the syntax are sample values.

```
apiVersion: v1
kind: Secret
metadata:
  name: secretarial
  labels:
    type: ocpm.secret.ldap
type: Opaque
stringData:
  name: "ldap1"
  password: "camiant"
  authDn: "uid=PolicyServer,ou=samplename,c=hu,o=samplename"
```

where:

- `name` is the configured LDAP server name.
- `password` is the LDAP credential for that data source.
- `authDN` is the authentication DN for that LDAP data source.
- `samplename` is the sample operator name.

2. Create the secret by executing the following command:

```
kubectl apply -f yaml_file_name -n <namespace>
```

Here:

- `yaml_file_name` is a name of the YAML file that is created in step 1.
- `<namespace>` is the deployment namespace used by the helm command.

Network Policies

The network policies allow ingress or egress rules to be defined based on Kubernetes resources such as Pod, Namespace, IP, and Ports. These rules are selected based on Kubernetes labels in the application. These network policies enforces access restrictions for all the applicable data flows except communication from Kubernetes node to pod for invoking container probe.

> **Note**
>
> Configuring network policy is optional. Based on the security requirements, network policy can be configured.

For more information on the network policy, see https://kubernetes.io/docs/concepts/services-networking/network-policies/.

For more information on configuring the network policy, see *Oracle Communications Cloud Native Core, Converged Policy Installation, Upgrade, and Fault Recovery Guide*.

A

Cloud Native Core Network Port Flows

This section describes network port flows for the Cloud Native Core.

Network Port Flows

- Cluster IP addresses are reachable outside of the cluster and are typically assigned by using a Network Load Balancer.
- Node IP addresses are reachable from the bastion host (and may be exposed outside of the cluster).

CNE Port Flows

Table A-1 CNE Port Flows

Name	Server/ Container	Ingress Port ext[:int]/ Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
SSH Access	ALL	22/TCP	Y		SSH Access	Administrative SSH Access. Only root or key is not allowed.
Repository	Bastion Host	80/TCP, 443/TCP, 5000/TCP	N (for 80/TCP) Y		Repository Access	Access repositories (YUM, Docker, Helm, and so on.)
RPC Bind	All	111/TCP, UDP	N		RPCBind	It is used for installation, pre booting of NFS mounted images.
BGP	Kubernetes Nodes	179/TCP	N		BGP	Used on bare metal environments in load balancing.
MySQL Query	MySQL SQL Node	3306/TCP	N	Replication Traffic	Microservice SQL Access	The SQL Query interfaces are used for 5G NFs to access the database and for remote sites to replicate data.
MySQL Management	MySQL Management Node	1186/TCP	N	Management Console Access		The SQL Management interface is used to access the management interfaces for the data cluster.
MySQL Data	MySQL Data Node	2202/TCP	N		SQL Query Backend	The SQL Data interface provide a backend DBMS interface for the SQL Query Nodes.
ILO	ILO Management Port	443/TCP	Y		Installation or Management	This interface is used to manage the frame and provides low-level management for all frame HW assets.

Table A-1 (Cont.) CNE Port Flows

Name	Server/ Container	Ingress Port ext[:int]/ Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
ETCD Client	Kubernetes Master Nodes	2379/TCP	Y		Client Access	Keystore DB used by Kubernetes
ETCD Peer	Kubernetes Master Nodes	2380/TCP	Y		Peer Access	ETCD Server Communication
Kube API Server	Kubernetes Master Nodes	6443/TCP	Y		Kubernetes Orchestration	The Kube API Server provides an orchestration API for the creation of Kubernetes resources.
Kubelet cAdvisor	Kubernetes Nodes	4149/TCP	Y		Container Metrics	Default cAdvisor port used to query container metrics.
Kubelet API	Kubernetes Nodes	10250/TCP	Y		Control Plane Node Access	API which allows full node access.
Kube-scheduler	Kubernetes Nodes	10251/TCP	N		Scheduler Access	Serve HTTP insecurely
Kube-controller	Kubernetes Nodes	10252/TCP	N		Controller Access	Serve HTTP insecurely
Kubelet Node State	Kubernetes Nodes	10255/TCP	Y		Node State Access	Unauthenticated read-only port, allowing access to node state.
Kube-proxy	Kubernetes Nodes	10256/TCP	N		Health Check	Health check server for Kube Proxy.
Kube-controller	Kubernetes Nodes	10257/TCP	Y		Controller Access	HTTPS Access
Kube-Scheduler	Kubernetes Node	10259/TCP	Y		Scheduler Access	HTTPS Access
Kibana	Kubernetes Nodes	80:5601/TPC	N	GUI		Logging Visualization
ElasticSearch	Kubernetes Nodes	9200/TCP	N	GUI		Search API access
ElasticSearch	Kubernetes Nodes	9300/TCP	N		Logging	Internal Logging
Jaeger Agent	Kubernetes Nodes	6831/UDP	N		Agent	Accept jaeger.thrift over compact thrift protocol.

Table A-1 (Cont.) CNE Port Flows

Name	Server/ Container	Ingress Port ext[:int]/ Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
Jaeger Agent	Kubernetes Nodes	6832/UDP	N		Agent	Accept jaeger.thrift over binary thrift protocol.
Jaeger Agent	Kubernetes Nodes	5778/TCP	N		Agent	Serve Configs
Jaeger Query	Kubernetes Nodes	80:16686/TCP	N	GUI		Service Frontend
Jaeger Collector	Kubernetes Nodes	14268/TCP	N		Collector	Accept jaeger.thrift directly from clients.
Jaeger Collector	Kubernetes Nodes	9411/TCP	N		Collector	Zipkin compatible endpoint (optional).
Prometheus Server	Kubernetes Nodes	80:9090/TCP	N	GUI		Prometheus Server
Prometheus Push Gateway	Kubernetes Nodes	9091/TCP	N		Push Gateway	Prometheus Push Gateway
Alertmanager	Kubernetes Nodes	80:9093/TCP	N	GUI		Alertmanager
Alertmanager clustering	Kubernetes Nodes	9094/TCP	N		Alertmanager Clustering	Alertmanager Clustering
Prometheus Exporters	Kubernetes Nodes	9100-9551/TCP 24231/TCP (fluent) 9099/TCP (snmp)	N		Prometheus Exporters	Prometheus Exporters
Grafana	Kubernetes Nodes	80:3000/TCP	N	GUI		Grafana
NSF	Bastion Host	2049 TCP/UDP	N			
NFS Statd	Bastion Host	44239/ NSF	N			statd will use a random ephemeral port.
Jenkins Server to Server	Bastion Host	50000/ TCP	N			

NF Port Flows

Table A-2 NF Port Flows

Name	Server / Container	Ingress Port [external]:internal	TLS	Cluster IP (Service IP)	Node IP	Notes
5G NRF	Kubernetes Nodes/NRF Service	80/TCP 443/TCP	N	NfConfiguration Ingress Gateway	NfRegistration NfSubscription NfDiscovery NfAccessToken Egress Gateway	5G NRF
5G SCP	Kubernetes Nodes/SCP Worker	8000/TCP	N		5G	5G SCP
5G SCP	Kubernetes Nodes/scp-configuration	8082/TCP	N	Proxy Configuration		5G SCP Configuration
5G SCP	Kubernetes Nodes/Istio		N		Mesh State Sharing	5G SCP Mesh Management
5G NSSF	Kubernetes Nodes/NSSF Service	80/TCP 443/TCP	Y	NSSF configuration Ingress Gateway	NS-selection, NS-availability, NS-subscription Egress Gateway NRF-Client	5G NSSF
5G UDR	Kubernetes Nodes/UDR Service	80/TCP	N		Nudr-dr/Nudr-prov	5G UDR: Signaling network can be used for management API exposed