Oracle® Communications

Cloud Native Core Security Guide

Release 1.2.0
F33145-01
July 2020

ORACLE

Oracle Communications Cloud Native Core Security Guide, Release 1.2.0

F33145-01

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.

What's New in This Guide

New or Updated Features in Release 1.2.0

This section introduces the documentation updates for Release 1.2.0 in Cloud Native Core Security Guide.

- Added security recommendations and procedures for:
  - InterWorking and Mediation Function (IWF)
  - Binding Support Function (BSF)
  - CNC Policy
- Updated security recommendations and procedures for:
  - Network Repository Function (NRF)
  - Cloud Native Core Console (CNCC)

List of Tables

1 Introduction

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

Install the CNC system software as "secure by default" where possible. In the few cases where this isn't possible, an installation time checklist procedure is created and listed on the Cloud Native Core Security Checklist. It is a short list of post-installation hardening activities that must be performed by the customer 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.

Audience

- Technology consultants
- Installers
- Security consultants
- System administrators

References

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

- Oracle Communications Signaling, Cloud Native Environment (OC-CNE) Installation Guide
- Cloud Native Core Console (CNCC) Installation Guide
- Network Slice Selection Function (NSSF) Cloud Native Installation Guide
- Service Communication Proxy (SCP) Cloud Native Installation Guide
- Cloud Native Core Policy Installation Guide
- Cloud Native Unified Data Repository (UDR) Installation and Upgrade Guide
- Network Repository Function (NRF) Cloud Native Installation and Upgrade Guide
- Security Edge Protection Proxy (SEPP) Cloud Native Installation Guide

Acronyms

Table 1-1 Acronyms

Term	Definition
OSSA	Oracle Software Security Assurance
OC-CNE	Oracle Communications CNE
NF	Network Function. A service providing some function in the 5G Core Network.
NRF	Network Repository Function
SCP	Service Communication Proxy
NSSF	Network Slice Selection Function
SEPP	Security Edge Protection Proxy
PCF	Policy Control Function
BSF	Binding Support Function
cnDRA	Cloud Native Diameter Routing Agent Network
CNE	Cloud Native Environment
5GC	5G Core Network
PKI	Public Key Infrastructure
mTLS	Mutual Transport Layer Security
OWASP	Open Source Foundation for Application Security
UDR	Unified Data Repository
CNCC	Cloud Native Core Console

2
Overview

Deployment Environment

The 4G/5G Cloud Native Core provides a variety of possible configuration and deployment models:

Table 2-1 Deployment Environment

Type	Host	CNE	Description
Bare-Metal	HP Gen 10 Blades / Rack Mount Servers / Cisco Switches	OC-CNE [1]	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.
Cloud	Customer Cloud	OC-CNE	In this environment, all the system elements are hosted in virtualized servers deployed on a customer provided Openstack environment. The OC-CNE is deployed on the openstack infrastructure.
Cloud	Customer CNE	Customer CNE [2]	In this environment, the customer provides the CNE and deploys the 5G NFs directly into the environment. The Oracle provided common services, and DB Tier are not used; equivalent functionality is provided by the customer.

1. Oracle Communications CNE provides basic CNE environment for on-premise deployment.
2. Customer CNE provides CNE environment for running 5G microservices.

> **Note:**
>
> The cloud environment security recommendations and procedures focuses on the OC-CNE reference environment. Customers providing their own CNE must have equivalent security procedures already in place.

3
Cloud Native Core Network Functions

The 4G/5G Network Funtions that are part of this document are following:

Table 3-1 5G Network Functions

Network Functions	Abbreviation	Description
Network Repository Function	NRF	NRF provides registration, discovery and authorization services to all the Network Functions (NF) in the 5G core network.
Service Communication Proxy	SCP	SCP provides a 5G-aware service mesh. The SCP is not a part of the current 3GPP 5G specification, but is expected to be added to a future iteration.
InterWorking Function	IWF	IWF provides 4G/5G inter-working support.
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).
Security Edge Protection Proxy	SEPP	In the roaming architecture, the home and the visited network are connected through Security Protection Proxy (SEPP) for the control plane of the internetwork interconnect.
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).
Unified Data Management/ Authentication Server Function	UDM/AUSF	UDM uses the subscription data stored in UDR and implements the application logic to perform various functionalities such as authentication credential generation, user identification, service and session continuity etc. The Authentication Server Function (AUSF) uses data stored in the UDM to perform authentication.
Network Exposure Function	NEF	Securely exposes network capabilities and events to Application Functions (AF).
Cloud Native Core Policy	CNC Policy	The CNC Policy is a functional element for policy control decision and flows based charging control functionalities.
Binding Support Function	BSF	Provides PCF binding (mapping and selection) for User Equipment (UE).

Table 3-2 4G Network Functions

Network Functions	Abbreviation	Description
Cloud Native 4G Diameter Routing Agent Network	cnDRA	Provides core (subset) 4G DSR capabilities delivered in a CNE microservice.

4
Secure Development Practices

Overview of Secure Development Practices

Oracle Software Security Assurance (OSSA) is Oracle's methodology for building security into the design, build, testing, and maintenance of its products in every phase of the product development life cycle. These products are used on premises by customers, or delivered through Oracle Cloud. Oracle's goal is to ensure that the products help customers meet their security requirements and provide the most cost effective ownership experience.

Secure Development - DevSecOps

Oracle secures the DevOps development process using a variety of techniques:

- Broad developer training to developers for understanding the principles of secure software development.
- Early creation of Trust Models and Risk Assessments to avoid common security pitfalls in the designs.
- Identify and expose sensitive interfaces to targeted testing for reducing or eliminating software vulnerabilities.
- Extensive use of automated security testing to identify vulnerabilities in third party software.
- Check for common OWASP (Open Source Foundation for Application Security) top 10 items and perform fuzz testing on key exposed interfaces.
- Evaluate deployed software configurations using industry best practices.

Vulnerability Handling

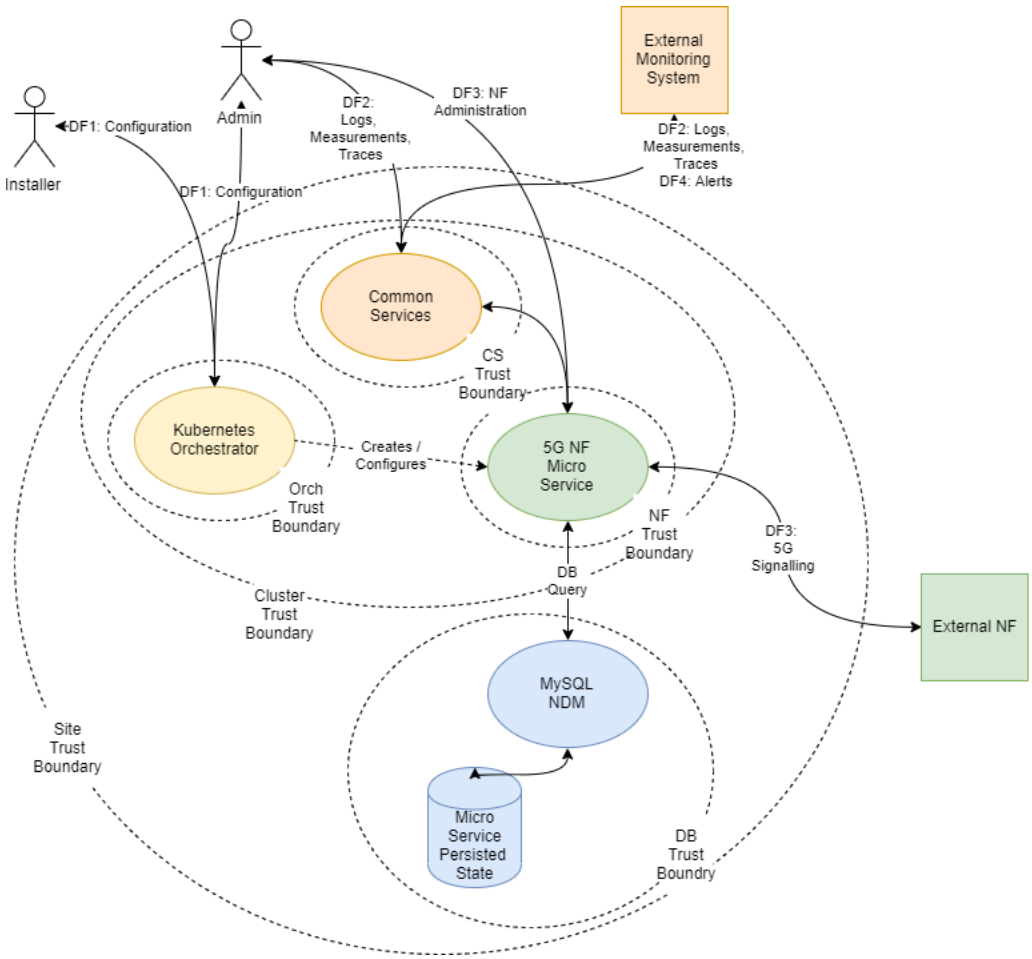
For details about the vulnerability handling, refer to Oracle Critical Patch Update Program. The primary mechanism for the backport of fixes for security vulnerabilities in Oracle products is the quarterly Critical Patch Update (CPU) program.

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 release provide cumulative patch updates.

5
Trust Model

The following Trust Model depicts the reference trust model (regardless of the target environment). The model describes the key access points and controls site deployment. While the model shows a single 5G NF microservice being deployed, typically many more would be deployed in an individual cluster.

Context diagram



Key Trust Boundaries

Following are the key trust boundaries:

Table 5-1 Key Trust Boundaries

Trust Boundary	Includes	Access Control
Site Trust Boundary	All the NF and other supporting elements for a given site.	Cluster Access Policies are implemented using some kind of Access Control Group (or Security Group) mechanism.
Cluster Trust Boundary	All the Compute Elements for a given cluster	Network Policies controls traffic ingress and egress; Pod Security Policies controls the kinds of workloads allowed in the cluster (Example: no pods requiring privilege escalation).
DB Trust Boundary	All the DB Tier Elements for a given Cluster	Firewall Policies control traffic ingress and egress; DB grants and other permission mechanisms provide authorization for authorized users.
Orchestrator Trust Boundary	The orchestration interface and keys	Firewall Policies control access to a Bastion server which provides orchestration services; access to the Bastion host uses SSH. 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 user interfaces (GUIs) that are currently open. The customer may want to introduce an api-gateway and implement authentication and authorization mechanisms to protect the OAM data. The common services may be configured to use Trasport Layer Security (TLS); when TLS is used, certificates will need to be generated and deployed via the orchestrator.
NF Trust Boundaries	A collection of one (or more) 5G Network Functions deployed as a service.	Some 5G NF microservices provide OAM access via a GUI. 5G NF microservices provide Signaling access via a TLS protected HTTP2 interface. The certificates for these interfaces are managed via the certificate manager.

External Data Flows

The following are external data flows:

Table 5-2 External Data Flows

Data Flow	Protocol	Description
DF1: Configuration	SSH	The installer or administrator accesses the orchestration system, which is hosted on the Bastion Server. The install or administrator must use ssh keys to access the bastion to a special orchestration account (not root); no password access is allowed.
DF2: Logs, Measurements, Traces	HTTP/HTTPS	The administrator or operator interacts with the common services using web interfaces.

Table 5-2 (Cont.) External Data Flows

Data Flow	Protocol	Description
DF3: 5G Signaling	HTTP2 (w/TLS)	All signalling interaction between NFs at a site and NFs at an external site is sent via TLS protected HTTP2.
DF4: Alerts	SNMP (Trap)	All alerting is performed using SNMP traps.

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

6

Common Security Recommendations and Procedures

4G/5G Application Authentication and Authorization

4G/5G NFs use Mutual Transport Layer Security (mTLS) authentication to secure communication. All NFs require a trust relationship to be established with all peers by exchanging and trusting peer root or intermediate certificates. All the peer certificates must be available in the trust store (K8s Secrets) in order to establish secured communication. Ideally, the trust store is populated from the customer Public Key Infrastructure (PKI) using ACME protocols. 4G/5G NFs also support manual importation and a semi-automatic import using the cert-manager external provider.

DB-Tier Authentication and Authorization

The DB-Tier provides a highly available multisite database used to store 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 explains how to change these account passwords. Additionally, communication between the NFs and the MySQL query nodes are protected using TLS.

The procedures are:

- **Modify MySQL NDB Root Password**
- **Configure TLS for MySQL NDB Query Nodes**

Procedure: Modify MySQL NDB Root Password

This procedure is executed by the DB Administrator.

For each of the MySQL Query nodes, perform the following steps :

1. Log into the next query node using ssh:

   ```
   $ ssh admusr@<mysql query node>
   ```

2. Execute the following command to make the node as root:

   ```
   $ sudo su
   ```

3. Invoke mysql using existing DB Root credentials:

   ```
   # mysql -h 127.0.0.1 -uroot -p
   ```

   Enter the password: `<enter existing root password>`

4. Change the DB Root credentials:

```
mysql> ALTER USER'root'@'localhost'IDENTIFIED BY'<NEW_PASSWORD>';
mysql> FLUSH PRIVILEGES;
```

Repeat steps 1 through 4 for each MySQL Query node.

> **Note:**
>
> If you are accessing a DB instance for the first time, the DB Root password is stored in the `/var/occnedb/mysqld_expired.log` file. The system generates a random password at installation time.

> **Note:**
>
> **Recommendation 1: Separation of Roles**
>
> The roles of DB Administrator and Cluster Administration must be kept separate. The DB Administrator must be responsible for securing and maintaining the DB-Tier MySQL NDM cluster. The Cluster Administrator must be responsible for securing and operating the Bastion Host and K8s 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 Cluster Administrator installs the NF (using helm).
>
> **Recommendation 2: Use Strong Passwords**
>
> The DB Administrator must choose a complex DB Root password as per their organization's security guidelines.

Procedure: Configure TLS for MySQL NDB Query Nodes

The MySQL NDB comes preconfigured to use a self-signed certificate that expires after 365 days. User can replace this certificate using the following procedure:

1. Create private CA and a set of Keys/Certificate pairs for use in securing MySQL :

```
$ my_ssl_rsa_setup
```

2. The available set of PEM files containing CA, server, and client certificates and keys that must be installed on all the MySQL Query Nodes.
3. Using SCP, copy the PEM files to the MySQL Query Node:

```
$ scp *.pem admusr@<mysql query node>
```

4. Login to the MySQL Query Node using ssh:

```
$ ssh admusr@<mysql query node>
```

5. Create a directory to hold the TLS keys and certs, and move them into root:

```
$ sudo mkdir /var/occnedb/opensslcerts
$ sudo chmod 700 /var/occnedb/opensslcerts
$ sudo mv ~admusr/*pem /var/occnedb/opensslcerts
```

6. Mysql Cluster Manager (mcm) is used to configure the TLS configuration from any of the DB nodes
   - Login to any DB node:
     - `$ ssh admusr@<any_db_node>`
       `$ sudo su`
   - Login to the mcm client:
     - `$ mcm`
     - `$ mcm>`
   - Update TLS config for all the SQL nodes using the mcm client:

```
$ mcm> set ssl-ca:mysqld=/var/occnedb/opensslcerts/ca.pem
occnendbclustera;
$ mcm> set ssl-cert:mysqld=/var/occnedb/opensslcerts/server-
cert.pem occnendbclustera;
$ mcm> set ssl-key:mysqld=/var/occnedb/opensslcerts/server-key.pem
occnendbclustera;
$ mcm> set tls_version:mysqld=TLSv1.2 occnendbclustera;
$ mcm> set ssl-cipher:mysqld=DHE-RSA-AES128-GCM-SHA256
occnendbclustera;
$ mcm> set ssl-ca:mysqld=/var/occnedb/opensslcerts/ca.pem
occnendbclustera;
$ mcm> set ssl-cert:mysqld=/var/occnedb/opensslcerts/server-
cert.pem occnendbclustera;
$ mcm> set ssl-key:mysqld=/var/occnedb/opensslcerts/server-key.pem
occnendbclustera;
$ mcm> set tls_version:mysqld=TLSv1.2 occnendbclustera;
$ mcm> set ssl-cipher:mysqld=DHE-RSA-AES128-GCM-SHA256
occnendbclustera;
```

7. Restart SQL nodes from MySQL Cluster Manager (mcm) client tool.

```
$ mcm> stop process 56 occnendbclustera;
$ mcm>start process 56 occnendbclustera;
$ mcm> stop process 57 occnendbclustera;
$ mcm> start process 57 occnendbclustera;
```

Repeat steps 3 through 7 for each MySQL Query node.

> **Note:**
>
> It is possible to integrate into an existing Public Key Infrastructure (PKI) by creating signing requests and having the PKI to generate the needed key/certificate pairs.

7

4G/5G Core Network Function Security Recommendations and Procedures

Network Repository Function (NRF) Security Recommendations and Procedures

This section provides Network Repository Function (NRF) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.
The procedures are:

- **NRF Access Token Secret Configuration**
- **NRF Access Token Secret Update**
- **OCNRF MYSQL Secret configuration**
  - **Kubernetes secret creation for OCNRF privileged database user**
  - **Kubernetes secret update for OCNRF privileged database user**
  - **Kubernetes secret creation for OCNRF application database user**
  - **Kubernetes secret update for OCNRF application database user**

NRF Access Token Secret Configuration

Use the following procedure to create access token secret :

1. Create the following files:
   - ECDSA private key and CA signed certificate of OCNRF (if initialAlgorithm is ES256)
   - RSA private key and CA signed certificate of OCNRF (if initialAlgorithm is RSA256)
   - KeyStore password file

   Note: Creation of private keys, certificates and passwords are at the discretion of user.
2. Login to Bastion Host or server from where kubectl can be executed.
3. 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>
```

Example:

```
$ kubectl create namespace ocnrf
```

4. Create kubernetes secret for Access token by following :
   a. To create kubernetes secret for HTTPS, following files are required:
      - ECDSA private key and CA signed certificate of OCNRF (if initialAlgorithm is ES256)
      - RSA private key and CA signed certificate of OCNRF (if initialAlgorithm is RSA256)
      - KeyStore password file

      Note: Creation process for private keys, certificates and passwords is on discretion of user/operator.
   b. Execute the following command to create secret. The names used below are same as provided in custom values.yaml in OCNRF deployment:

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

   c. Execute the following command to verify secret created:

      ```
      $ kubectl describe secret <ocnrfaccesstoken-secret-name> -n
      <Namespace of OCNRF 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 key and CA signed certificate of OCNRF (if initialAlgorithm is ES256)
   - RSA private key and CA signed certificate of OCNRF (if initialAlgorithm is RSA256)
   - KeyStore password file

   Note: Update of private keys, certificates and passwords are at the discretion of user.
2. Login to Bastion Host or server from where kubectl can be executed
3. Update the secret with new/updated details by following:
   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 will look like:

```
kubectl create secret generic <ocnrfaccesstoken-secret> --
fromfile=<ecdsa_private_key.pem>
 --from-file=<rsa_private_key.pem> --fromfile=<ssl_truststore.txt> --
from-file=<keystore_password.txt>
 --fromfile=<rsa_certificate.crt> --from-file=<ecdsa_certificate.crt> --
dry-run -o yaml -n
 <Namespace of Access Token secret> | kubectl replace -f - -n <Namespace
of Access Token secret>Copy
```

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

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

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

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

OCNRF MYSQL Secret Configuration

This section describes the secret creation for two type of OCNRF users. Different users has different set of permissions.

- OCNRF privileged user : This catogory of user has complete set of permissions. The user can perform DDL and DML operations to perform install/upgrade/rollback or delete operations.
- OCNRF application user : This catogory of user has less set of permissions and will be used by OCNRF application during service operations handling. The user

can insert, update, get, remove the records. This user can't create, alter and drop the database as wells as tables

Kubernetes secret creation for OCNRF privilaged database user

This section explains the steps to create kubernetes secrets for accessing OCNRF database for the privileged user.

1. Login to Bastion Host or server from where kubectl can be executed.
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 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=<OCNRF Privileged Mysql database
      username> --from-literal=dbPassword=<OCNRF Privileged Mysql
      User database passsword> --from-literal=appDbName=<OCNRF Mysql
      database name> --from-literal=networkScopedDbName=<OCNRF Mysql
      Network database name> -n <Namespace of OCNRF 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 -n ocnrf
      ```

b. Verify the secret created using above command:

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

Example:

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

Kubernetes secret update for OCNRF privileged database user

This section explains the steps to update kubernetes secrets for accessing OCNRF database for the privileged user.

1. Login 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=<OCNRF Privileged Mysql database
username> --from-literal=dbPassword=<OCNRF Privileged Mysql
database password> --from-literal=appDbName=<OCNRF Mysql
database name> --from-literal=networkScopedDbName=<OCNRF Mysql
Network database name> -n <Namespace of OCNRF 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=<OCNRF Privileged Mysql database
username> --from-literal=dbPassword=<OCNRF Privileged Mysql
database password> --from-literal=appDbName=<OCNRF Mysql
database name> --from-literal=networkScopedDbName=<OCNRF Mysql
Network database name> --dry-run -o yaml -n <Namespace of OCNRF
deployment> | kubectl replace -f - -n <Namespace of OCNRF
deployment>
```

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

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

Kubernetes secret creation for OCNRF application database user

This section explains the steps to create kubernetes secrets for accessing OCNRF database for the application database user.

1. Login to Bastion Host or server from where kubectl can be executed.
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>
```

Example:

```
$ kubectl create namespace ocnrf
```

3. Create kubernetes secret for OCNRF application database user for configuring records is as follows:

a. Create kubernetes secret for OCNRF application database user:

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

Example:

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

Kubernetes secret update for OCNRF application database user

This section explains the steps to update kubernetes secrets for accessing OCNRF database for the application database user.

1. Login 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=<OCNRF APPLICATION
       User Name> --from-literal=dbPassword=<Password for OCNRF
      APPLICATION User> --from-literal=appDbName=<OCNRF
      Application Database> -n <Namespace of OCNRF 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=<OCNRF APPLICATION
      User Name> --from-literal=dbPassword=<Password for OCNRF
      APPLICATION User> --from-literal=appDbName=<OCNRF
      Application Database> --dry-run -o yaml -n <Namespace of OCNRF
      deployment> | kubectl replace -f - -n <Namespace
      of OCNRF deployment>
      ```

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

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

Cloud Native Core Policy Security Recommendations and Procedures

This section provides Cloud Native Core Policy specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.
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**

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`)

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

2. Login to Bastion Host or server from where kubectl can be executed.
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 create 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. Login to Bastion host or server from where kubectl can be executed.
3. Update the secret with new/updated details:

   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. Login 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. 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 right side of column (:) are examples.

```
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 right side of column (:) are examples.

```
apiVersion: v1
kind: Secret
metadata:
  name: occnp-admin-db-pass
type: Opaque
data:
  mysql-username: b2NjbnBhZG1pbnVzcg==
  mysql-password: b2NjbnBhZG1pbnBhc3N3ZA==
```

> **Note:**
>
> 'name' will be used for the **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. Execute 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 the whether the secret is created by executing the following command:

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

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 right side of column (:) are examples.

```
apiVersion: v1
kind: Secret
metadata:
  name: secretarial
  labels:
    type: ocpm.secret.ldap
type: Opaque
stringData:
  name: "ldap1"
```

```
password: "camiant"
authDn: "uid=PolicyServer,ou=vodafone,c=hu,o=vodafone"
```

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.

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.

Cloud Native Diameter Routing Agent (cnDRA) Security Recommendations and Procedures

This section provides cloud native Diameter Routing Agent (cnDRA) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- **User (OAM) Authentication and Authorization**
- **Authentication and Authorization of Applications**

User (OAM) Authentication and Authorization

- cnDRA supports REST based MMI interface. There is no GUI provided in the current cnDRA release.
- The MMI interface is based on fixed user and password, using which the security token is requested by REST client from cnDRA.
- cnDRA does not allow or support configuration or modify these credentials (user and password).

Authentication and Authorization of Applications

cnDRA currently supports TCP based signaling traffic connection towards the Remote Peer Nodes. These connections are not currently secured via TLS etc mechanism. Currently there is no plan to enable securing of the application/Diameter traffic.

Cloud Native Core Ingress/Egress Gateways Security Recommendations and Procedures

This section provides Ingress/Egress Gateway specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- **Enabling TLS and Ciphers in Ingress/Egress Gateway**
- **Certificate Management and Dynamic reload of certificates in Gateways**

Enabling TLS and Ciphers in Ingress/Egress Gateway

Use the following procedure to enable the TLS and Ciphers :

1. **Helm Configuration to enable TLS:**

   To open Https port in **Ingress gateway:** configure in helm `enableIncomingHttps: true`

   To have a Https client configured in **Egress gateway:** configure in helm `enableOutgoingHttps: true`

2. Create following files:
   a. RSA or ECDSA Private key (Example: `rsa_private_key_pkcs1.pem`)
   b. Trust store password (Example: `trust.txt`)
   c. Key store password(Example: `key.txt`)
   d. Certificate chain for trust store (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 are at the discretion of user.

3. Create secret by executing following command:

```
$ 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 cipher suites:

   # Cipher Suites to be enabled on Server side (Ingress Gateway),

   # Cipher Suites to be enabled on Client side (Egress Gateway),

   cipherSuites:

```
-TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
```

Certificate Management and Dynamic reload of certificates in Gateways

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

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

Command:

```
$ 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 which is implemented in update container will check for change in file state and replace 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 end point to dynamically reload key and truststore. 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.

Service Communication Proxy (SCP) Security Recommendations And Procedures

This section provides Service Communication Proxy Function (SCP) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- **OCSCP MYSQL Secret configuration**
- **OCSCP MYSQL Secret Updates for Password of DB User**

OCSCP MYSQL Secret configuration

Use the following procedure to create Mysql kubernetes secret :

1. Login 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>
   ```

   Example:

   ```
   $ kubectl create namespace ocscp
   ```

3. Execute

```
$ kubectl
        create secret generic <secretName>  --
fromliteral=DB_USERNAME=<userName>
        --fromliteral=DB_PASSWORD=<password>  --
fromliteral=DBNAME=<dbName> -n
      <SCPNamespace>
```

to create the secret for Mysql.
Example:

```
$ kubectl create secret generic cred
      --fromliteral=DB_USERNAME=root --
fromliteral=DB_PASSWORD=lLn94uba5p
      --fromliteral=DB_NAME=ocscpdb -n scpsvc
```

Where

<secretName> is Secret name and must be same value present for 'dbSecretName' in ocscp_values.yaml file.
<SCPNamespace> must be the name of namespace where SCP will be deployed.

4. Verify the whether the secret is created

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

Example:

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

.

OCSCP MYSQL Secret Updates for Password of DB User

Use the following procedure to update Mysql secret:

1. Login to Bastion Host or server from where kubectl can be executed.
2. Update the kubernetes secret for Mysql by executing the following command:

   Delete the secret:

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

   Create the secret with updated details:

```
$ kubectl create secret generic <secretName> --
fromliteral=DB_USERNAME=<userName>
--fromliteral=DB_PASSWORD=<password> --fromliteral=DBNAME=<dbName> -
n <SCPNamespace>
```

Network Slice Selection Function (NSSF) Security Recommendations and Procedures

This section provides Network Slice Selection Function (NSSF) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- **OCNSSF Access Token Secret Configuration**
- **OCNSSF Access Token Secret Update**
- **OCNSSF MYSQL Secret Configuration**
- **OCNSSF MYSQL Secret Update**

OCNSSF 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 OCNSSF certificate (Example: `ecdsa_ocnssf_certificate.crt`)
   - CA signed RSA OCNSSF certificate (Example: `rsa_ocnssf_certificate.crt`)

   Note: Creation of private keys, certificates and passwords are at the discretion of user.
2. Login to Bastion Host or server from where kubectl can be executed.
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
```

OCNSSF 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 OCNSSF certificate (Example: `ecdsa_ocnssf_certificate.crt`)
   - CA signed RSA OCNSSF certificate (Example: `rsa_ocnssf_certificate.crt`)

   Note: Update of private keys, certificates and passwords are at the discretion of user.
2. Login to Bastion Host or server from where kubectl can be executed.
3. Update the secret with new/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
```

OCNSSF MYSQL Secret Configuration

Use the following procedure to create Mysql kubernetes secret:

1. Login 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 ocnssf
```

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

> **Note:**
>
> The values mentioned in the right side of column (:) are examples.

```
apiVersion: v1
 kind: Secret
 metadata:
```

```
  name: <secret-name>
  type: Opaque
data:
  mysql-username: cGNmdXNy
  mysql-password: cGNmcGFzc3dk
```

1. **Note:** The values for "mysql-username" and "mysql-password" must be base64 encoded.
2. Execute

   ```
   kubectl create -f <yaml_file_name> -n <namespace>
   ```

   to create the secret.
3. Verify whether the secret is created by executing the following command:

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

OCNSSF MYSQL Secret Update

Use the following procedure to update Mysql kubernetes secret:

1. Login to Bastion Host or server from where kubectl can be executed.
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. Execute `kubectl create -f <yaml_file_name> -n <namespace>` to create the secret.
5. Verify whether the secret is created by executing the following command:
   `$ kubectl describe secret <secret-name> -n <namespace>`

Security Edge Protection Proxy (SEPP) Security Recommendations and Procedures

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

The procedures are:

- **SEPP Access Token Secret Configuration**
- **OCSEPP Access Token Secret Update**

SEPP Access Token Secret Configuration

Use the following procedure to create access token secret :

1. Login to Bastion Host or server from where kubectl can be executed.
2. Create the following files:
   - ECDSA private key with P-256 curve

     Example:

     ```
     ecdsa_private_key_pkcs8.pem
     ```

   Note: Creation of private keys, certificates and passwords are at the discretion of user.
3. Create namespace for the secret by executing the following command:

   ```
   $ kubectl create namespace seppsvc
   ```
4. Create kubernetes secret for Access token by executing the following command:

   ```
   $ kubectl create secret generic ocsepp-ipx-secret
   --from-file=ecdsa_private_key_pkcs8.pem -n seppsvc
   ```

OCSEPP Access Token Secret Update

Use the following procedure to update access token secret:

1. Login to Bastion Host or server from where kubectl can be executed.
2. Update the following files:
   - ECDSA private key with P-256 curve

     Example:

     ```
     ecdsa_private_key_pkcs8.pem
     ```

   Note: Update of private keys, certificates and passwords are at the discretion of user.
3. Update the secret with new/updated details.
   Delete the secret:

   ```
   $ kubectl delete secret ocsepp-ipx-secret -n seppsvc
   ```

   Create the secret again with updated details:

   ```
   $ kubectl create secret generic ocsepp-ipx-secret
   --from-file=ecdsa_private_key_pkcs8.pem -n seppsvc
   ```

Unified Data Repository (UDR) / Unstructured Data Storage Function (UDSF) Security Recommendations and Procedures

This section provides Unified Data Repository (UDR) / Unstructured Data Storage Function (UDSF) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedure is:

- **OCUDR MYSQL kubernetes secret for storing database username and password**

OCUDR MYSQL kubernetes secret for storing database username and password

Use the following procedure to create Mysql kubernetes secret:

1. Login 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. Create a yaml file with the username and password with the syntax shown below:

> **Note:**
>
> The values mentioned in the right side of column (:) are examples.

```
apiVersion: v1
 kind: Secret
 metadata:
   name: <secret-name>
   type: Opaque
 data:
    dbname: dWRyZGI=
    dsusername: dWRydXNlcg==
    dspassword: dWRycGFzc3dk
    encryptionkey: TXkgc2VjcmV0IHBhc3NwaHJhc2U=
```

Note: The values for "dbname", "dsusername", "dspassword" and "encryptionkey" must be base64 encoded.

4. Execute

```
kubectl create -f <yaml_file_name> -n <namespace>
```

to create the secret.

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

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

InterWorking and Mediation Function (IWF) Security Recommendations and Procedures

This section provides InterWorking and Mediation Function (IWF) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- **OCIWF MYSQL Secret configuration**
- **OCIWF MYSQL Secret updates for password of DB user**

OCIWF MYSQL Secret configuration

Use the following procedure to configure Mysql secret:

1. Login to Bastion Host or server from where kubectl can be executed.
2. Create namespace for the mysql secret. If it is already created, skip this step.

```
$ kubectl create namespace <IwfNamespace>
Example: $ kubectl create namespace iwfsvc
```

3. Create kubernetes secret for Mysql

```
$ kubectl create secret generic <secretName> --
fromliteral=DB_USERNAME=<userName>
      --fromliteral=DB_PASSWORD=<password> --
fromliteral=DBNAME=<dbName> -n <IwfNamespace>
Example: $ kubectl create secret generic cred --
fromliteral=DB_USERNAME=root
      --fromliteral=DB_PASSWORD=lLn94uba5p --
fromliteral=DB_NAME=ociwfdb -n
    iwfsvc
```

> **Note:**
>
> a. <secretName> → Secret name must be same to value present for 'dbSecretName' in ociwf_values.yaml file.
>
> b. <IwfNamespace> -> Must be the name of namespace where IWF will be deployed.

4. Verify that secret is create successfully

```
$ kubectl describe secret database-secret -n <IwfNamespace>
Example: $ kubectl describe secret database-secret -n iwfsvc
```

OCIWF MYSQL Secret updates for password of DB user

Use the following procedure to update MYSQL Secret for password of DB user:

1. Login to Bastion Host or server from where kubectl can be executed
2. Update the kubernetes secret for Mysql

```
Delete the secret:
         $ kubectl delete secret database-secret -n <IwfNamespace>
         Create the secret with updated details:
         $ kubectl create secret generic <secretName> --
fromliteral=DB_USERNAME=<userName>
      --fromliteral=DB_PASSWORD=<password> --
fromliteral=DBNAME=<dbName> -n
    <IwfNamespace>
```

Binding Support Function (BSF) Security Recommendations and Procedures

This section provides Binding Support Function (BSF) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedure is:

- **Creating BSF MYSQL Kubernetes Secret for Storing Database Username and Password**

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. Login to Bastion Host or server from where kubectl can be executed.
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:

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

> **Note:**
>
> The values mentioned in the right side of column (:) are examples.

```
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 right side of column (:) are examples.

```
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. Execute 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>
   ```

8

Cloud Native Core Console (CNCC) Security Recommendations and Procedures

This section provides Cloud Native Core Console (CNCC) specific security recommendations and procedures. Recommendations common to all 5G/4G are available in the Common Procedures Section.

The procedures are:

- CNCC IAM MYSQL Secret Configuration
- CNCC IAM Default User (Admin) Secret Configuration
- CNCC IAM LDAP Configuration
- CNCC TLS Secret configuration
- CNCC Core Secret Configuration to Enable HTTPS
- CNCC IAM SAML Configuration

CNCC IAM MYSQL Secret Configuration

Use the following procedure to create Mysql kubernetes secret:

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

   ```
   $ kubectl get namespaces
   ```

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

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

   ```
   $ kubectl create namespace cncc
   ```

3. Execute 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 passsword> -n <Namespace of MYSQL secret
   ```

   Execute the following command to verify the secret creation:
   ```
   $ kubectl describe secret <database secret name> -n <Namespace of
   MYSQL secret>
   ```

Example:

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

CNCC IAM Default User (Admin) Secret Configuration

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

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

   ```
   $ kubectl get namespaces
   ```

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

   ```
   $ kubectl create namespace <required namespace>


   $ kubectl create namespace cncc
   ```

3. Execute 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>
   ```

   Execute the following command to verify the secret creation:

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

   Example:

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

CNCC IAM LDAP Configuration

Use the following procedure to configure CNCC IAM LDAP :

1. Setting up User Federation with CNCC IAM by executing following steps:
   a. Login to CNCC IAM application.
   b. Select Cncc Realms and 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) appears next to the Save and Cancel.

e. If a user has to be import 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. Login to CNCC IAM application.

   b. Select Cncc Realms and 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 drop down 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 appears.

   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 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.

CNCC TLS Secret configuration

Use the following procedure to configure CNCC TLS Secret:

1. To create kubernetes secret for HTTPS, following files are required:
   - ECDSA private key and CA signed certificate of CNCC (if initialAlgorithm is ES256)
   - RSA private key and CA signed certificate of CNCC (if initialAlgorithm is RSA256)
   - TrustStore password file
   - KeyStore password file
   - CA certificate
2. Create a secret by executing 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 executing the above command, the following message will be displayed:
secret/cncc-iam-ingress-secret created

Execute 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 executing 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 executing the above command, the following message will be displayed:
secret/cncc-iam-ingress-secret replaced

CNCC Core Secret Configuration to Enable HTTPS

Use the following procedure to configure CNCC Core Secret to Enable HTTPS:

1. To create kubernetes secret for HTTPS, following files are required:
   - ECDSA private key and CA signed certificate of CNCC (if initialAlgorithm is ES256)
   - RSA private key and CA signed certificate of CNCC (if initialAlgorithm is RSA256)
   - TrustStore password file
   - KeyStore password file
   - CA certificate
2. Create a secret by executing 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
       cncc
   ```

   On successfully executing the above command, the following message will be displayed:
   secret/cncc-core-ingress-secret created

   Execute 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 executing 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 executing the above command, the following message will be displayed:
secret/cncc-core-ingress-secret replaced

CNCC IAM SAML Configuration

Use the following procedure to configure CNCC IAM SAML:

1. To configure SAML identity provider **(IdP)** in CNCC IAM, login to CNCC IAM Console using admin credentials provided during installation of CNCC 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 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.

9

Cloud Native Environment (CNE) Security Recommendations and Procedures

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

Credential Name	Deployment	Type	Associated Resource	Initial Setting	Credential Rotation
TOR Switch	Bare Metal Only	username and password	Cisco Top or Rack Switch	username and password from PreFlight Checklist	Reset post-install
Enclosure Switch	Bare Metal Only	username and password	HP Enclosure Switch	username and password from PreFlight Checklist	Reset post-install
OA Admin	Bare Metal Only	username and password	On-board Administrator Console	username and password from PreFlight Checklist	Reset post-install
ILO Admin	Bare Metal Only	username and password	HP Integrated Lights Out Manger	username and password from PreFlight Checklist	Reset post-install
Server Super User (root)	All	username and password	Server Super User	Set to well-known Oracle default during server installation	Reset post-install
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, it must be changed prior to placing the system into operation. The customer must store these credentials in a safe and secure way offsite. It is recommended that the customer must plan a regular schedule for updating (rotating) these credentials. Specific procedures and recommendations for OC-CNE credential management are provided below.

The procedures are:

1.1.Network Security Recommendations and Procedures

Recommendation: Review and Follow TOR installation procedures

The OC-CNE on-premise installation guide provides detailed procedures on how to configure the TOR switches and configure them for remote monitoring. Deviations from the standard installation time configurations are not recommended.

Credential Management Procedures

Procedure 1: Setting Top Of Rack Switch Credentials

This procedure is used to set the credentials on the Cisco TOR switch as deployed with the bare metal deployment option. Steps for creating and deleting accounts and for setting account passwords is given below.

1. Login to the TOR switch (from the bastion host):

   `$ ssh <username>@<switchIP address>` User Access Verification

   `Password: <password>`

   Cisco Nexus Operating System (NX-OS) Software

   TAC support: www.cisco.com/tac

   `<switchname>#`

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 desired):

   # configureEnter 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 <switchIP address> closed.

   ```
   $ ssh <newusername>@<switchIP address>
   ```

   User Access Verification `Password: <newpassword>`

   Cisco Nexus Operating System (NX-OS) SoftwareTAC support: www.cisco.com/tac

   <switchname>#

5. Delete an unrequired user account:

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

Procedure 2: Setting Enclosure Switch Credentials
This procedure is used to set the credentials on the HP enclosure switch as deployed with the bare metal 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.

Setting Enclosure Switch Credentials

1. Login to the HP enclosure switch (from the bastion host): `$ ssh <username>@<switchIP address>`

   ```
   Copyright (c)2010-2017Hewlett Packard Enterprise Development LP **
   Without the owner's prior written consent,
   ** no decompiling or reverse-engineering shall be allowed.
   <switchname>
   <switchname>
   sysSystem View:returnto User View with Ctrl+Z.
   ```

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:

   ```
   [switchname]local-user <newusername>class[manage|network]
   ```

   New local user added

   ```
   [switchname-luser-manage-<newusername>]password simple <newpassword>
   [switchname-luser-manage-<newusername>]quit
   ```

4. Delete the user account that is not required:

   ```
   [switchname]undo local-user <username>class <currentclass>
   ```

5. Delete the user account that is not required:

   ```
   [switchname]undo local-user <username>class <currentclass>
   ```

6. Save the configuration changes:

   ```
   [switchname]save
   The current configuration will be written to the device. Are you
   sure? [Y/N]: y
   Please input the file name(*.cfg)[flash:/<filename>]
   (To leave the existing filename unchanged, press the enter key):
   flash:/<filename> exists, overwrite? [Y/N]: yValidating file.
   Please wait...
   Saved the current configuration to mainboard device successfully.
   Slot1:
   Save next configuration file successfully.
   [switchname]
   [switchname]save
   The current configuration will be written to the device. Are you
   sure? [Y/N]: y
   Please input the file name(*.cfg)[flash:/<filename>]
   (To leave the existing filename unchanged, press the enter key):
   flash:/<filename> exists, overwrite? [Y/N]: yValidating file.
   Please wait...
   ```

```
Saved the current configuration to mainboard device successfully.
Slot1:
Save next configuration file successfully.
[switchname]
```

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

1.2 Hosting Environment Security Recommendations and Procedures

The best way to keep your CNE environment secure is to keep it up-to-date. New OC-CNE releases are typically released every 2 months. The OC-CNE upgrade is not service affecting and will typically install newer versions of:

- Host OSs
- Kubernetes and associated containers
- DB-Tier binaries
- Common service containers

The upgrade process ensures that the uplifts do not affect active service. Refer Cloud Native Environment (OC-CNE) Upgrade Guide for more details.

Repository Management Recommendations

System Update (YUM) Recommendations
Recommendation: **Keep central yum repositories up to date**.

Keep central repositories up-to date with latest yum packages; yum updates are performed on-site whenever a fresh install or upgrade is performed. An up-to date yum repository will help ensure that fixes for all publish vulnerabilities are applied.

Docker Repository Recommendations
Recommendation: **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 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.

1.3 Credential Management Procedures

Procedure 1: Setting HP Onboard Administrator (OA) Credentials.

This procedure is used to set the credentials on the HP Onboard Administrator as deployed with the bare metal deployment option. Steps for creating and deleting accounts and for setting account passwords is shown. For additional information, please refer to: HP commands to configure OA username and password.

1. Login to the OA:

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

WARNING: This is a private system. Do not attempt to login unless you are anauthorized 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
(sOA-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 login 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:

```
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.
```

Procedure 2: Setting HP Integrated Lights Out Manger (ILO) Credentials

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

1. Login 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 an unneeded account:

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

Procedure 3: Setting Root Passwords for All Cluster Nodes

The procedure to reset the root account requires that the administrator login to each and every server.

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

1. Login 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 steps 1 - 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 users 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.

Procedure 4: 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 bastion host, and the key used to access the cluster servers. 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 bastion host key pair is typically provided to administrators who will manage the cluster. The key pair used to access the cluster servers should kept local to the cluster:

Table 9-1 Updating admusr SSH Keys

Key Pair Name	Public Key Distribution	Private Key Distribution
Bastion Host	Place copy in the authorized_keys file on the bastion host	**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.	**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:

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

Update the bastion host keys

1. Login to the bastion host and generate a new key pair using the ssh-keygen command show above.

   ```
   $ ssh-keygen -b 4096 -t rsa -C "New
               Bastion Key" -f ~/.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 (e.g., putty or openssh)
3. Add the new public key to the authorized key file on the bastion host:

   ```
   $ 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 login.
6. Assuming that you were able to login 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.

General Security Administration Recommendations and Procedures

> **Note:**
>
> **Recommendation:** Record configuration changes
>
> Note that 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, etc). This can be done using normal linux user administration procedures.

SSHD Policy Administration Procedures

Customers may want to create augment the standard sshd configuration to perform additional hardening; this can be done using normal linux ssh administration procedures. Note that 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:**
>
> **Recommendation**: 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 seperate file in the `/etc/audit/rules.d` directory; do not modify any of the other existing audit configuration files

A

Cloud Native Core Network Port Flows

Network Port Flows

- Cluster IP addresses are reachable outside of the cluster and are typically assigned via a Network Load Balancer
- Node IP addresses are reachable from the bastion host (and may be exposed outside of the cluster)

OC-CNE Port Flows

Name	Sever/ 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; no root / key only.
Repository	Bastion Host	80/TCP, 443/ TCP, 5000/TCP	Y		Repository Access	Access repositories (YUM, Docker, Helm, etc.)
RPC Bind	All	111/ TCP, UDP	N		RPCBind	Used for installation; pxe booting of NFS mounted images
BGP	K8s 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

Name	Sever/ Container	Ingress Port ext[:int]/Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
MySQL Data	MySQL Data Node	50501/ 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 / Management	This interface is used to manage the frame; it provided low level management for all of the frame HW assets
MetalLB	Metallb Speaker	7472	N		Load Balancer	This port is used to share service locations
Metrics Server	K8s Metrics Server	8443	N		Monitoring	Used for scaling
ETCD Client	K8s Master Nodes	2379/TCP	Y		Client Access	Keystore DB used by K8s
ETCD Peer	K8s Master Nodes	2380/TCP	Y		Peer Access	ETCD Server Communication
Kube API Server	K8s Master Nodes	6443/TCP	Y		K8s Orchestration	The Kube API Server provides an orchestration API for the creation of K8s resources.
Kubelet cAdvisor	K8s Nodes	4149/TCP	Y		Container Metrics	Default cAdvisor port used to query container metrics
Kubelet API	K8s Nodes	10250/ TCP	Y		Control Plane Node Access	API which allows full node access
Kube-scheduler	K8s Nodes	10251/ TCP	N		Scheduler Access	Serve HTTP insecurely
Kube-controller	K8s Nodes	10252/ TCP	N		Controller Access	Serve HTTP insecurely
Kube-proxy	K8s Nodes	10256/ TCP	N		Health Check	Health check server for Kube Proxy
Kube-proxy	K8s Nodes	30000-32767	N		Service Access	The default service node port range
Kube-controller	K8s Nodes	10257/ TCP	Y		Controller Access	HTTPS Access
Kube-Scheduler	K8s Node	10259/ TCP	Y		Scheduler Access	HTTPS Access

Name	Sever/ Container	Ingress Port ext[:int]/Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
Kibana	K8s Nodes	80:5601/TPC	N	GUI		Logging Visualization
ElasticSearch	K8s Nodes	9200/TCP	N	GUI		Search API access
ElasticSearch	K8s Nodes	9300/TCP	N		Logging	Internal Logging
Jaeger Agent	K8s Nodes	6831/UDP	N		Agent	Accept jaeger.thrift over *compact* thrift protocol
Jaeger Agent	K8s Nodes	6832/UDP	N		Agent	Accept jaeger.thrift over binary thrift protocol
Jaeger Agent	K8s Nodes	5778/TCP	N		Agent	Serve Configs
Jaeger Query	K8s Nodes	80:16686/TCP	N	GUI		Service Frontend
Jaeger Collector	K8s Nodes	14268/TCP	N		Collector	Accept jaeger.thrift directly from clients
Jaeger Collector	K8s Nodes	9411/TCP	N		Collector	Zipkin compatable endpoint (optional)
Prometheus Server	K8s Nodes	80:9090/TCP	N	GUI		Prometheus Server
Prometheus Push Gateway	K8s Nodes	9091/TCP	N		Push Gateway	Prometheus Push Gateway
Alertmanager	K8s Nodes	80:9093/TCP	N	GUI		Alertmanager
Alertmanager clustering	K8s Nodes	9094/TCP	N		Amertmanger Clustering	Alertmanager Clustering
Prometheus Exporters	K8s Nodes	9100-9551/TCP 24231/TCP (fluent) 9099/TCP (snmp)	N		Prometheus Exporters	Prometheus Exporters
Grafana	K8s Nodes	80:3000/TCP	N	GUI		Grafana

NF Port Flows

Table A-1 NF Port Flows

Name	Sever / Container	Ingress Port [external]:internal	TLS ?	Cluster IP (Service IP)	Node IP	Notes
5G NRF	K8s Nodes / NRF Service	80/TCP 443/TCP	Y	NfConfiguration IngressGateway	NfRegistration NfSubscription NfDiscovery NfAccessToken EgressGateway	5G NRF Refer OCNRF microservices to port mapping for more details.
5G SPF	K8s Nodes / SPF Worker	8000/TCP	N		5G Proxy	5G SCP (SPF) Proxy
5G SPF	K8s Nodes / Soothsayer	8082/TCP	N	Proxy Configuration		5G SCP (SPF) Proxy Configuration.
5G SPF	K8s Nodes / Istio	/TCP	N		Mesh State Sharing	5G SCP (SPF) Mesh Management.
5G NSSF	K8s Nodes / NSSF Service	80/TCP 443/TCP	Y	NSSF configuration IngressGateway	NS-selection, NS-availability, NS-subscription EgressGateway NRF-Client	5G NSSF Refer to NSSF microservices to port mapping for further details.
5G UDR/ UDSF	K8s Nodes / UDR Service	80/TCP	N		Nudr-dr/ Nudr-prov	5G UDR: Signalling network can be used for management API exposed.