Oracle® Communications

Network Analytics Suite Security Guide

Release 23.2.0
F80261-01
June 2023

Oracle Communications Network Analytics Suite Security Guide, Release 23.2.0

F80261-01

Contents

A Network Port Flows

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:

- 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**.
- For Hardware, Networking and Solaris Operating System Support, select **3**.

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 provides information about the acronyms and the terminology used in the document.

Table Acronyms

Acronym	Description
3GPP	3rd Generation Partnership Project
5GC	5G Core Network
5GS	5G System
AF	Application Function
API	Application Programming Interface
AMF	Access and Mobility Management Function
CLI	Command Line Interface
CNC	Cloud Native Core
CNE	Oracle Communications Cloud Native Core, Cloud Native Environment
FQDN	Fully Qualified Domain Name
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
KPI	Key Performance Indicator
HA	High Availability
IMSI	International Mobile Subscriber Identity
K8s	Kubernetes
ME	Monitoring Events
MPS	Messages Per Second
Network Slice	A logical network that provides specific network capabilities and network characteristics.
NEF	Network Exposure Function
NF	Network Function
NRF	Oracle Communications Cloud Native Core, Network Repository Function
NSI	Network Slice Instance. A set of Network Function instances and the required resources (such as compute, storage and networking resources) which form a deployed Network Slice.
NSSF	Oracle Communications Cloud Native Core, Network Slice Selection Function
NWDAF	Network Data Analytics Function
OAM	Operations, Administration, and Maintenance
OHC	Oracle Help Center
OSDC	Oracle Service Delivery Cloud
PLMN	Public Land Mobile Network
REST	Representational State Transfer
SBA	Service Based Architecture
SBI	Service Based Interface

Table (Cont.) Acronyms

Acronym	Description
SCP	Service Communication Proxy
SEPP	Security Edge Protection Proxy
SMF	Session Management Function
SNMP	Simple Network Management Protocol
SVC	Services
SUPI	Subscription Permanent Identifier
UDM	Unified Data Management
UE	User Equipment
URI	Uniform Resource Identifier

What's New in This Guide

This section lists the documentation updates for Network Analytics Suite release *23.2.x* .

Release 23.2.0 - F80261-01, June 2023

- Updated the OCNADD Overview to include UI Router service.
- Added new section, Customizing CSR and Certificate Extensions in the TLS Configuration chapter.
- Updated the section **Configuring SASL for Kafka** in the Kafka Security Configuration chapter.
- Added the procedures for Renewing OCNADD Certificates and Certificate Creation for NFs and Third Party Consumers.
- Added "DF5" data flow information in External Data Flows.

1 Introduction

The Security Guide provides an overview of the security information applicable to the Oracle Communications Network Analytics Suite of products.

This document contains recommendations and step-by-step instructions to assist the customer in hardening the Network Analytics Suite system, it 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 Networks Data Analytics Function User Guide*
- *Oracle Communications Networks Data Analytics Function Solutions Guide*
- *Oracle Communications Networks Data Analytics Function Troubleshooting Guide*
- *Oracle Communications Networks Data Analytics Function Installation and Fault Recovery Guide*
- *Oracle Communications Cloud Native Environment Installation Guide*
- *Oracle Communications Cloud Native Environment Installation, Upgrade, and Fault Recovery Guide*
- *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide*
- *Oracle Communications Network Analytics Data Director Troubleshooting Guide*
- *Oracle Communications Network Analytics Data Director User Guide*

2
Overview

2.2 OCNADD Overview

Deployment Environment

The OCNADD can be deployed in a variety of possible configurations and environments:

Table 2-1 Deployment Environment

Type	Host	CNE	Description
Cloud	Customer Cloud	CNE	The CNE is deployed on the cloud provided by the customer. The OCNADD is deployed on the Kubernetes cluster managed by CNE.
Cloud	Customer CNE	Customer CNE(TANZU)	The customer provides the CNE and deploys the OCNADD directly into the environment. The Oracle provided DBTier is used. The list of all other required software is listed in the install guide *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide* section "*Software Requirements*".

OCNADD Services

Table 2-2 OCNADD Services

Services	Descriptio
Admin Service	The Admin Service administers the Kafka Cluster and it provides an interface to create and delete a Consumer Adapter deployment.
Aggregation Service	The Aggregation Service collects and aggregates network traffic from multiple NFs (For example: SCP 1 and 2 and/or SEPP, NRF, 3rd party NF, and so on).

Table 2-2 (Cont.) OCNADD Services

Services	Descriptio
Alarm Service	The OCNADD Alarm service is used to store the alarms generated by other OCNADD services.
UI Router Service	The OCNADD UI Router service acts as an interface between the CNC Console and the OCNADD services.
Configuration Service	The Configuration Service is used by the OCNADD UI service to configure 3rd party message feed.
Consumer Adapter Service	The Consumer Adapter Service manages the third-party specific connection parameters and provide secure message transport towards the third-party consumer application.
Health Monitoring Service	The Health Monitoring Service monitors the health of each OCNADD service and provides alerts related to the overall health of the OCNADD.
Kafka Broker Service	The Kafka Broker Service stores the incoming network traffic from different NFs. It also stores the intermediate processed data from different microservices.
GUI service	The GUI service is used as an interface between the CNC Console and OCNADD that provides the user interface to configure the Data Feeds.

2.1 OCNWDAF Overview

Deployment Environment

The OCNWDAF can be deployed in a variety of possible configurations and environments:

Table 2-3 Deployment Environment

Type	Host	CNE	Description
Cloud	Customer Cloud	CNE	The CNE is deployed on the cloud provided by the customer. The OCNWDAF is deployed on the Kubernetes cluster managed by CNE.

OCNWDAF 3GPP Standards

Table 2-4 OCNWDAF 3GPP Standards

Network Function	Abbreviation	Description	3GPP Standard
Networks Data Analytics Function	OCNWDAF	Assists in collecting and analyzing data in a 5G network.	• 3GPP TS 23.288 v16 • 3GPP TS 23.288 v17.4.0 • 3GPP TS 29.520 v17.6.0 • 3GPP TS 29.508 v17.5.0 • 3GPP TS 29.518 v17.5.0 • 3GPP TS 23.501 v17.5.0 • 3GPP TS 23.502 v17.4.0 • 3GPP TS 33.521 v17.1.0

3
Secure Development Practices

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

3.1 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 OCNWDAF and OCNADD 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.

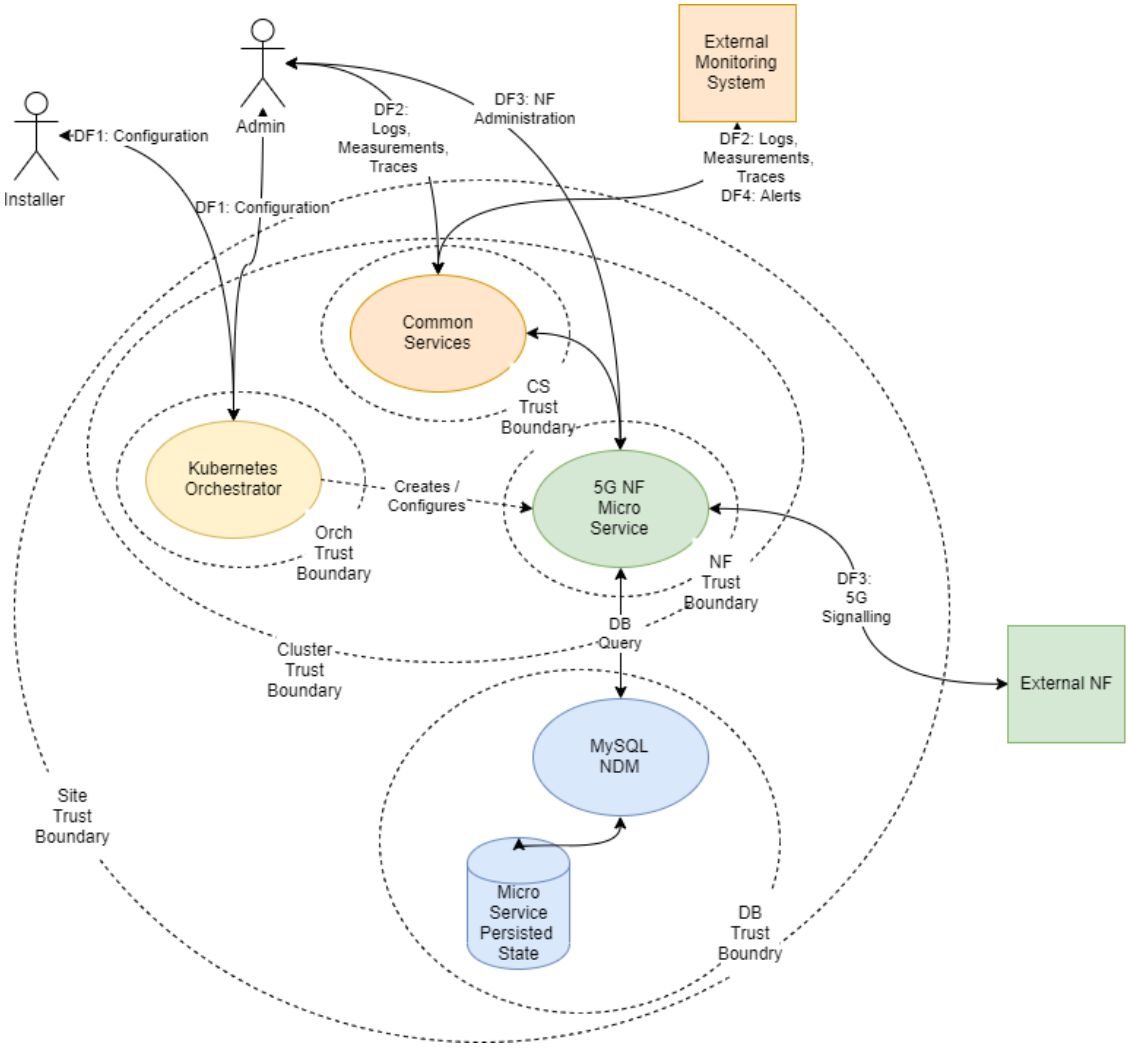
3.2 Trust Model for OCNWDAF

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.

3.2.1 Context diagram

Figure 3-1 Context Diagram



3.2.2 Key Trust Boundaries

Following are the key trust boundaries:

Table 3-1 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 3-1 (Cont.) Key Trust Boundaries

Trust Boundary	Includes	Access Control
Cluster Trust Boundary	All the compute elements for a given cluster	Network Policies control Ingress and Egress traffic. Pod Security Policies manage the workloads allowed in the cluster (For example, no pods requiring privilege escalation).
DB Trust Boundary	All the DBTier elements for a given cluster	Firewall Policies control Ingress and Egress traffic. Database 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.	Some 5G NF microservices provide OAM access through a GUI. 5G NF microservices provide Signaling access through a TLS protected HTTP2 interface. The certificates for these interfaces are managed via the certificate manager.

3.2.3 External Data Flows

The following are external data flows:

Table 3-2 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.

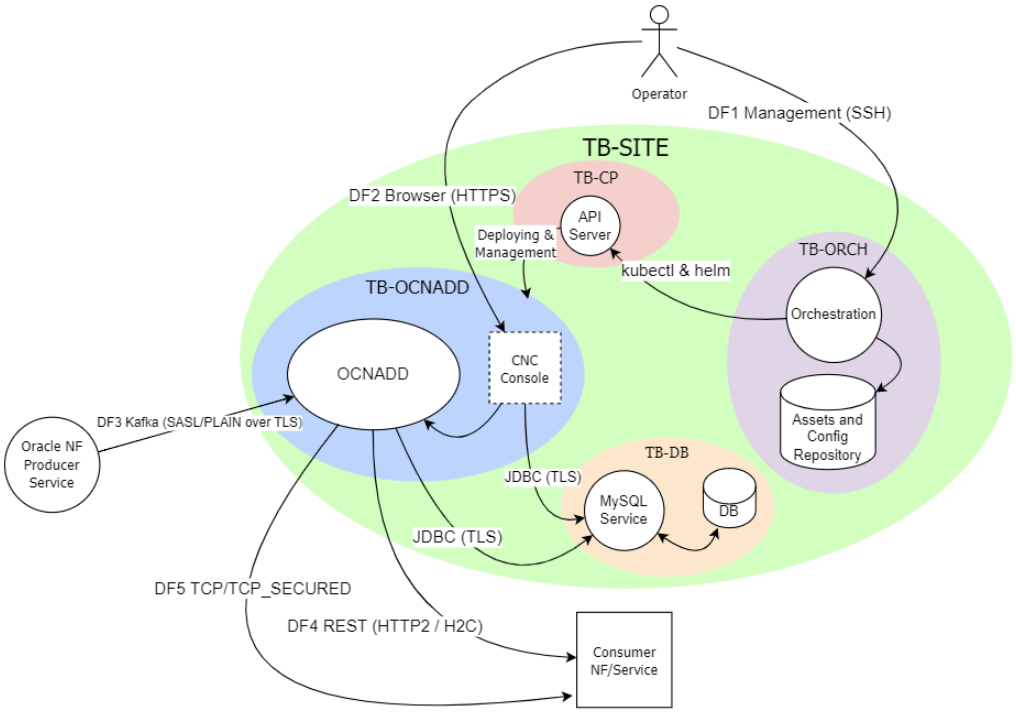
The complete list of network flows including service types and ports are available in Network Port Flows.

3.3 Trust Model for OCNADD

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.

3.3.1 Context Diagram

Figure 3-2 Context Diagram



3.3.2 Key Trust Boundaries

Following are the key trust boundaries:

Table 3-3 Key Trust Boundaries

Trust Boundary	Access Control
OCNADD	Kubernetes Namespace for OCNADD where its internal micro-services are deployed.
Site	Where the Kubernetes cluster is deployed.
Control Plane	This trust boundary delineates the control plane elements of the clusters, that is, API Server, kubelet, containerd and etcd. The configuration database (ETCD service) is isolated so that only control plane services can access it.

Table 3-3 (Cont.) Key Trust Boundaries

Trust Boundary	Access Control
Database	MySQL service deployed in a separate Kubernetes namespace.
CNE Infra	Namespace containing all the infrastructure related services (like Prometheus). Provided by CNE.
Orchestration	Includes the orchestration server and the Code and Image Repository.

3.3.3 External Data Flows

Table 3-4 External Data Flows

Data Flow	Protocol	Description
DF1 : Management	SSH	Operator will login to the orchestration server through SSH for deploying OCNADD and/or managing the OCNADD Kubernetes deployment using helm.
DF2 : Browser	HTTPS	Operator uses CNC Console to create, manage feed configuration and monitor OCNADD. CNC Console is accessed through the browser and the Operator is authenticated with username and password before access is granted.
DF3 : Kafka	SASL_SSL	Oracle NFs write the 5G SBI data or messages into the Kafka exposed by OCNADD in the respective topics. OCNADD will then process according to the feed configurations. The communications are encrypted using TLS and Oracle NF will authenticate themselves to Kafka through SASL/PLAIN (username and password).
DF4 : Consumer NF	HTTP2(w/TLS)	OCNADD forwards the message feed to respective consumer NF/s as HTTP2 (over TLS) or H2C (HTTP2 clear text) messages according to the feed configurations.
DF5: Consumer NF (Synthetic Packet)	TCP	OCNADD forwards the Synthetic Packet to respective consumer NF/s as TCP or TCP_SECURED messages based on the feed configuration.

4

Implementing Security Recommendations and Guidelines

4.2 Implementing Security Recommendations and Guidelines for OCNADD

This section provides specific recommendations and guidelines for OCNADD security.

4.2.1 TLS Configuration

External TLS Communication

DF2: Browser and **DF3: Kafka communication** is over TLS only. No additional steps apart from certificate creation need to be performed. See **Certificate and Secret Generation** for more information about certificate creation.

DF4: Consumer NF communication can be configured to use clear text or TLS through the OCNADD GUI. For more information, see "Configuring OCNADD Using CNC Console" section in the *Oracle Communications Network Analytics Data Director User Guide*.

DF5: Consumer NF (Synthetic Feed) communication can be configured to use TCP or TCP_SECURED through the OCNADD GUI. For more information, see "Configuring OCNADD Using CNC Console" section in the *Oracle Communications Network Analytics Data Director User Guide*.

See Certificate Creation for NFs and Third Party Consumers before creating certificates for NFs, OCNADD, and Third Party Consumers.

Internal TLS Communication

In addition to the above communication, the customers can also opt to use TLS for internal OCNADD communication. To enable internal TLS, follow the steps below:

- Enable internal TLS in Helm Charts.

  Change the value of *global.ssl.intraTlsEnabled* to *true* in the `ocnadd-custom-values.yaml`. It is set to *false* by default.

  ```
  ssl:
      intraTlsEnabled:true
  ```

- Create TLS certificates for all the internal services.
  - If the **generate_certs** script is used to create the certificates, add entries for internal services in the `ssl_certs/default_values/values` file. For more information see, **Certificate and Secret Generation**.
  - If the following entries are not present in the values file, add them:

In following example, *ocnadd-deploy* is the namespace where OCNADD is deployed and *occne-ocdd* represents the Kubernetes domain name.

```
[ocnadduirouter]
client.commonName=ocnadduirouter-client
server.commonName=ocnadduirouter
DNS.1=*.ocnadduirouter.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnadduirouter

[ocnaddadminservice]
client.commonName=ocnaddadminservice-client
server.commonName=ocnaddadminservice
DNS.1=*.ocnaddadminservice.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddadminservice

[ocnaddalarm]
client.commonName=ocnaddalarm-client
server.commonName=ocnaddalarm
DNS.1=*.ocnaddalarm.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddalarm

[ocnaddconfiguration]
client.commonName=ocnaddconfiguration-client
server.commonName=ocnaddconfiguration
DNS.1=*.ocnaddconfiguration.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddconfiguration

[ocnaddcache]
client.commonName=ocnaddcache-client
server.commonName=ocnaddcache
DNS.1=*.ocnaddcache.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddcache

[ocnaddhealthmonitoring]
client.commonName=ocnaddhealthmonitoring-client
server.commonName=ocnaddhealthmonitoring
DNS.1=*.ocnaddhealthmonitoring.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddhealthmonitoring

[ocnaddscpaggregation]
client.commonName=ocnaddscpaggregation-client
server.commonName=ocnaddscpaggregation
DNS.1=*.ocnaddscpaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddscpaggregation

[ocnaddnrfaggregation]
client.commonName=ocnaddnrfaggregation-client
server.commonName=ocnaddnrfaggregation
DNS.1=*.ocnaddnrfaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddnrfaggregation

[ocnaddseppaggregation]
client.commonName=ocnaddseppaggregation-client
server.commonName=ocnaddseppaggregation
```

```
DNS.1=*.ocnaddseppaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddseppaggregation
```

Certificate and Secret Generation

The OCNADD services can communicate with each other as well as with external interfaces in both secure, encrypted mode as well as in unsecure mode. For establishing encrypted communication between the services, there is a necessity to generate TLS certificates and private keys for each microservice. The service certificate is generated using the provided CA certificate. For more information, refer to the *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide*.

The generated service certificates are stored as the K8s secret.

> **Note**
>
> The default certification creation assumes that internal TLS is enabled and creates the certificates for all the OCNADD services. The customers can choose to delete surplus entries from `ssl_certs/default_values/values` when not using internal TLS. This reduces the number of certificates to be signed by the CA.

Below are sample **values** files with/without internal TLS enabled:

where:

- ocnadd-deploy : is the namespace where OCNADD is deployed
- occne-ocnadd : is the Kubernetes domain name

Without Internal TLS

```
# Do not modify any keys in global section. Please edit only values present
in global section.
# Edit only commonName value for Root CA. Do not modify key
# You can add multiple services in same manner as the sample services are
added. The format should be as follows
#service name, common name for service and list of subject alternate name
#e.g.,
#[<service_name>]
#commonName=your.svc.common.name
#IP.1 = 127.0.0.1
#IP.2 = 10.72.31.4
#DNS.1 = localhost
#DNS.2 = svc.cluster.local
# Make sure to provide a single empty line (without space) after end of every
section
# Do not add comments anywhere in this script to avoid parsing error

[global]
countryName=IN
stateOrProvinceName=KA
localityName=BLR
organizationName=ORACLE
organizationalUnitName=CGBU
defaultDays=365
```

```
##root_ca
commonName=*.ocnadd-deploy.svc.occne-ocdd

[kafka-broker]
client.commonName=kafka-broker-zk
server.commonName=kafka-broker
DNS.1=*.kafka-broker.ocnadd-deploy.svc.occne-ocdd
DNS.2=kafka-broker
DNS.3=*.kafka-broker

[zookeeper]
client.commonName=zookeeper-zk
server.commonName=zookeeper
DNS.1=*.zookeeper.ocnadd-deploy.svc.occne-ocdd
DNS.2=zookeeper

[adapter]
client.commonName=adapter
server.commonName=adapter-server
DNS.1=*adapter.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddconsumeradapter

##end
##end
```

With Internal TLS

```
# Do not modify any keys in global section. Please edit only values present
in global section.
# Edit only commonName value for Root CA. Do not modify key
# You can add multiple services in same manner as the sample services are
added. The format should be as follows
#service name, common name for service and list of subject alternate name
#e.g.,
#[<service_name>]
#commonName=your.svc.common.name
#IP.1 = 127.0.0.1
#IP.2 = 10.72.31.4
#DNS.1 = localhost
#DNS.2 = svc.cluster.local
# Make sure to provide a single empty line (without space) after end of every
section
# Do not add comments anywhere in this script to avoid parsing error

[global]
countryName=IN
stateOrProvinceName=KA
localityName=BLR
organizationName=ORACLE
organizationalUnitName=CGBU
defaultDays=365

##root_ca
commonName=*.ocnadd-deploy.svc.occne-ocdd
```

```
[kafka-broker]
client.commonName=kafka-broker-zk
server.commonName=kafka-broker
DNS.1=kafka-broker.ocnadd-deploy.svc.occne-ocdd
DNS.2=*.kafka-broker.ocnadd-deploy.svc.occne-ocdd
DNS.3=kafka-broker
DNS.4=*.kafka-broker

[zookeeper]
client.commonName=zookeeper-zk
server.commonName=zookeeper
DNS.1=*.zookeeper.ocnadd-deploy.svc.occne-ocdd
DNS.2=zookeeper

[ocnaddthirdpartyconsumer]
client.commonName=ocnaddthirdpartyconsumer-client
server.commonName=ocnaddthirdpartyconsumer
DNS.1=*.ocnaddthirdpartyconsumer.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddthirdpartyconsumer

[oraclenfproducer]
client.commonName=oraclenfproducer
server.commonName=oraclenfproducer-server
DNS.1=*.oraclenfproducer.ocnadd-deploy.svc.occne-ocdd
DNS.2=oraclenfproducer

[ocnadduirouter]
client.commonName=ocnadduirouter-client
server.commonName=ocnadduirouter
DNS.1=*.ocnadduirouter.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnadduirouter

[ocnaddadminservice]
client.commonName=ocnaddadminservice-client
server.commonName=ocnaddadminservice
DNS.1=*.ocnaddadminservice.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddadminservice

[ocnaddalarm]
client.commonName=ocnaddalarm-client
server.commonName=ocnaddalarm
DNS.1=*.ocnaddalarm.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddalarm

[ocnaddconfiguration]
client.commonName=ocnaddconfiguration-client
server.commonName=ocnaddconfiguration
DNS.1=*.ocnaddconfiguration.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddconfiguration

[ocnaddhealthmonitoring]
client.commonName=ocnaddhealthmonitoring-client
server.commonName=ocnaddhealthmonitoring
DNS.1=*.ocnaddhealthmonitoring.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddhealthmonitoring
```

```
[ocnaddscpaggregation]
client.commonName=ocnaddscpaggregation-client
server.commonName=ocnaddscpaggregation
DNS.1=*.ocnaddscpaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddscpaggregation

[ocnaddnrfaggregation]
client.commonName=ocnaddnrfaggregation-client
server.commonName=ocnaddnrfaggregation
DNS.1=*.ocnaddnrfaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddnrfaggregation

[ocnaddseppaggregation]
client.commonName=ocnaddseppaggregation-client
server.commonName=ocnaddseppaggregation
DNS.1=*.ocnaddseppaggregation.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddseppaggregation

[adapter]
client.commonName=adapter
server.commonName=adapter-server
DNS.1=*adapter.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddconsumeradapter

[ocnaddcache]
client.commonName=ocnaddcache-client
server.commonName=ocnaddcache-server
DNS.1=*ocnaddcache.ocnadd-deploy.svc.occne-ocdd
DNS.2=ocnaddcache
DNS.3=ocnaddcache-wka
DNS.4=ocnaddcache-wka.ocnadd-deploy.svc.occne-ocdd

##end
```

MTLS Configuration

The customers can opt to use MTLS for internal OCNADD communication. To enable the internal MTLS, run the following steps:

- Enable internal MTLS in Helm Charts.

  Change the value of global.ssl.mTLS to "true" in the ocnadd-custom-values.yaml. By default it is set to false.

  ```
  ssl:
      intraTlsEnabled: true
      mTLS: true
  ```

  Change the value of *ssl.client-auth* to "required" in `<chart-path>/charts/ocnaddkafka/template/scripts-config.yaml`

  ```
  ssl.client.auth=required
  ```

- The certificate creation step remains the same as mentioned in the "**TLS Configuration**" section.

> **Note**
>
> – It is mandatory to create certificates and secrets for MTLS to work.

Customizing CSR and Certificate Extensions

OpenSSL creates the Certificate Signing Requests (CSRs) and certificates in the `generate_certs.sh` script of OCNADD. OpenSSL configuration file templates `ssl_certs/templates/services_client_openssl.cnf` and `ssl_certs/templates/services_server_openssl.cnf` are used during CSR or certificate creation.

Modify the files based on customer requirements. For more information, see OpenSSL x509 v3 Certificate and CSR Configuration.

Listed below are some sample scenarios where file modifications are required:

Subject Alternative Name (SAN) Requirement for Client Certificate Signing Request (CSR)

When external Certificate Authorities (CAs) are used to issue certificates based on the CSR generated by the `generate_certs.sh` script, the CA may require that the CSR contains Subject Alternative Name (SAN) in addition to Common Name (CN). SAN is added in the Server CSR but omitted in the Client CSR.

To add SAN extension for client CSR, append the following text at the bottom of `ssl_certs/templates/services_client_openssl.cnf`:

```
[ req_ext ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

subjectAltName = @alt_names # Add this for SAN support

[ alt_names ]
```

Modify the `[ req ]` section in `ssl_certs/templates/services_client_openssl.cnf` as follows:

```
[ req ]
default_bits        = 2048
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes      = req_attributes
x509_extensions = usr_cert  # The extensions to add to the self signed cert
req_extensions = req_ext # Add extensions required for CSR
```

Key Usage Requirement for Client and Server CSR

External CA may require specific key usage and external key usage to be mentioned in the CSR. Add or modify the `[ req_ext ]` section in both `ssl_certs/templates/`

`services_client_openssl.cnf` and `ssl_certs/templates/services_server_openssl.cnf` files as follows:

```
[ req_ext ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage=critical,digitalSignature,keyEncipherment # Modify Key Usage
extendedKeyUsage=serverAuth,clientAuth # Add Extended Key Usage
```

Also add reference to `req_ext` in `[ req ]` section as follows:

```
[ req ]
default_bits        = 2048
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes      = req_attributes
x509_extensions = usr_cert  # The extensions to add to the self signed cert
req_extensions = req_ext # Add extensions required for CSR
```

The above code snippets add the standard key and extended key usage required by the CA to sign the certificate. The user can similarly add other specific usage requirements.

4.2.2 Network Policy

The OCNADD is also shipped with Network Policies for ingress connections which are disabled by default.

The Network Policies can be categorized into three categories:

- **Internal service Connections:** The service connections that are internal to the OCNADD namespace. To enable network policies, change the value of global.network.policy.enable to *true* in `ocnadd-custom-values.yaml` file. By default, it is *false*.

  ```
  network:
      # enable network policies
      policy:
        enable: true
  ```

- **External service Connections:** The service connections that are connecting to the OCNADD namespace. The policy gets applied on connections from OCCNCC, OCSCP, OCNRF and so on which are deployed outside the OCNADD namespace. To allow connections from specific namespaces (when network policies are enabled), add the namespace to the namespaces list in global.network.policy.ingress.namespaces of `ocnadd-custom-values.yaml` file as below:

  ```
  #allow ingress network connections from below namespaces
  namespaces:
    -occne-infra
    -occncc
  ```

In the above example, the connections from all pods of occne-infra (prometheus, grafana, etc.) and occncc namespace are allowed. Similarly add the SCP, and NRF namespace to this list.

- **External service Connectionfrom specific IP address**: This can be enabled (disabled by default) to allow connections from specific IP address ranges when Network Policy is enabled. In the below example, the connections from the 10.0.0.0/8 subnet are allowed. In other words, all connections from IP address 10.0.0.0 - 10.255.255.255 are allowed.

```
# to allow external connections from postman, curl, etc.
# only needed for development/debugging purposes
external:
   enable:true
   cidrs:
     -10.0.0.0/8
```

4.2.3 Kafka Security Configuration

The basic configuration for Kafka security in order to be compliant with product delivery is described below.

Prerequisites

- SSL certificates & keys (generated using ssl scripts which are provided as part of OCNADD release)

Configuring SASL for Kafka

Kafka uses the Java Authentication and Authorization Service (JAAS) for SASL configuration. In OCNADD, by default, all Kafka communication (inter-broker, Kafka-client, and Kafka-broker and between Kafka and Zookeeper) happens with a common user (ocnadd) configured in `<chart-path>/charts/ocnaddkafka/config/kafka_server_jaas.conf`

Customers can add more users in `<chart-path>/charts/ocnaddkafka/config/kafka_server_jaas.conf` file. The "ocnadd" user is the default user for all internal Kafka communication and should not be modified.

Below are the sample changes to add NRF, SCP, and SEPP users in Kafka:

```
KafkaServer {
org.apache.kafka.common.security.plain.PlainLoginModule required
username="ocnadd"
password="<change this>"
user_ocnadd="<change this>";
user_NRF="NRF-secret";
user_SCP="SCP-secret";
user_SEPP="SEPP-secret";
};
Client {
org.apache.zookeeper.server.auth.DigestLoginModule required
username="ocnadd"
password="<change this>";
};
```

In the above code snippet `user_NRF="NRF-secret"` is added for NRF user, `user_SEPP="SEPP-secret"` is added for SEPP user, and `user_SCP="SCP-secret"`is added for SCP user. Where "NRF-secret", "SEPP-secret" and "SCP-secret" are the passwords.

After making the above changes, continue with the installation steps as mentioned in the *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide*.

> **Note**
>
> - Field marked as `<change this>` needs to be updated by the user to set the Kafka admin password.
> - User addition is one time operation and can be done only during installation.
> - It is recommended to use a strong password in **kafka_server_jass.conf**.

4.2.4 MySQL Configuration

The OCNADD uses cnDBtier service for the database, which provides secure database connectivity. For configuration details, see *Oracle Communications Network Analytics Data Director Installation Guide, Upgrade, and Fault Recovery Guide*.

4.2.5 Certificate Creation for NFs and Third Party Consumers

The following sections provide information about certificate creation rules to be considered by the NFs and Third Party Consumers.

Certification Creation for NFs

NFs must ensure the following conditions are met prior to sending data to OCNADD over TLS:

- NFs must trust the Certificate Authority(CA) that issues the certificate to the OCNADD. The CA certificate for OCNADD's Certificate Authority must be included in the trust store, *cacert* file, or the *ca-bundle* of the NF.
- When **mTLS** is enabled in OCNADD and an NF in the same Kubernetes cluster wishes to send data to the SSL endpoint instead of SASL_SSL endpoint, the TLS certificate of the NF should be issued by the same CA issuing certificate to the OCNADD or any one of the trusted CAs.

Certification Creation for Third Party Consumers

When TLS encrypted feeds (HTTP2 or TCP_SECURED) are configured, it is essential that the third party endpoint given in the feed configuration supports TLS. In addition to supporting TLS, consider the following while creating certificates for the consumer:

- The certificate should be signed by any of the trusted CAs or by the CA issuing certificate for the OCNADD.
- The certificates Common Name (CN) or Subject Alternative Name (SAN) contains the host name or IP address provided as part of the feed configuration.
  For example, if Feed A is configured to send data to Consumer A (*https://thirdpartyconsumer/*) and Feed B is configured to send data to Consumer B (*https://10.10.10.10*), the certificate for Consumer A must contain the *thirdpartyconsumer* in CN or

SAN of its certificate and similarly Consumer B's certificate must contain 10.10.10.10 in the SAN field of its certificate.

- If **mTLS** is enabled by the Third Party Consumer, it must trust the CA issuing certificate for OCNADD. The CA certificate for OCNADD's CA must be included in the trust store, *cacert* file or *ca-bundle* of the Third Party Consumer.

CA Certificate Expiry

Suppose the CA certificate expires before the OCNADD certificate expiry. In that case, the customer can renew the CA (if self-managed) and reuse the certificate provided the CA certificate's hash, private key, public key, and so on remain the same.

Renewing the OCNADD certificate and performing a rollout restart after renewing the CA certificate will ensure that OCNADD services are using a valid, non-expired CA certificate. For more information, see, Renewing OCNADD Certificates.

4.2.6 Renewing OCNADD Certificates

Follow the steps in the sections below to renew the OCNADD certificates.

> **Note**
>
> Steps to renew the certificate vary based on the method used to create the certificates.

Certificate Generated Using CACert and CAKey

If the certificate is generated using CACert and CAKey:

1. Add the services whose certificates must be renewed in the `ssl_certs/default_values/renew_cert_files` file.
   The following code snippet displays the services to be added if Internal TLS is enabled:

```
# This files contain the list of services for which certificate needs to
be renewed
# The service name should be exactly same for which the certificates has
been initially generated
# defaultDays is number of days upto which certificate should be renewed.
Certificate for all listed
# service will be updated with this value.

defaultDays=365

kafka-broker
zookeeper
egw
ocnaddthirdpartyconsumer
oralcenfproducer
ocnadduirouter
ocnaddadminservice
ocnaddalarm
ocnaddconfiguration
ocnaddhealthmonitoring
ocnaddscpaggregation
ocnaddnrfaggregation
```

```
ocnaddsepaggreagation
adapter
ocnaddcache
```

2. To generate the certificate, run the following command:

   ```
   ./generate_certs.sh -cacert <path to>/cacert.pem -cakey <path to>/
   cakey.pem --renew
   ```

3. Provide the password to your CA key at the "*Enter passphrase for CA Key file:*" prompt.
4. Old certificates are revoked, and new certificates are created.
5. At the prompt "*Would you like to choose any above namespace for creating secrets (y/n)*", enter "*y*".
6. At the prompt "*Enter Kubernetes Namespace:* ", enter the OCNADD namespace. Secrets containing the certificates are replaced.
7. Enter the OCNADD namespace at the prompt "*Enter namespace:* ", a rolling restart of all the OCNADD services is triggered.
8. Wait for the rollout restart completion and for all pods to return to the "Running" state.

Only CSR is Generated

If only the CSR is generated:

1. Navigate to `ssl_certs/demoCA/services/<service-name>/client` or `ssl_certs/demoCA/services/<service-name>/server ssl_certs/` for each *<service-name>* given in the *default_values/values* file.
   The following example displays how to navigate to client folder of the adapter service:

   ```
   [ssl_certs]$ ls
   default_values  demoCA  generate_certs.sh  generate_secrets.sh  logs
   template
   [ssl_certs]$ cd demoCA/services/adapter/client/
   [client]$ ls
   adapter-clientcert.pem  adapter-client.csr  adapter-clientprivatekey.pem
   adapter_client_ssl.cnf  client_rsa_certificate.crt
   ```

2. Delete the old CSR request.
   The following example displays how to delete the old CSR request for the adapter service:

   ```
   [client]$ ls
   adapter-clientcert.pem  adapter-client.csr  adapter-clientprivatekey.pem
   adapter_client_ssl.cnf  client_rsa_certificate.crt
   [client]$ rm adapter-client.csr
   [client]$ ls
   adapter-clientcert.pem  adapter-clientprivatekey.pem
   adapter_client_ssl.cnf  client_rsa_certificate.crt
   ```

3. To create a new CSR (client.csr) from the old certificate (clientcert.pem) and private key (clientprivatekey.pem), run the following command:

   ```
   openssl x509 -x509toreq -in clientcert.pem -signkey clientprivatekey.pem -
   out client.csr
   ```

The following example displays how to create a new client CSR for the adapter service:

```
[client]$ ls
adapter-clientcert.pem  adapter-clientprivatekey.pem
adapter_client_ssl.cnf  client_rsa_certificate.crt
[client]$ openssl x509 -x509toreq -in adapter-clientcert.pem -signkey
adapter-clientprivatekey.pem -out adapter-client.csr
Getting request Private Key
Generating certificate request
[client]$ ls
adapter-clientcert.pem  adapter-client.csr  adapter-clientprivatekey.pem
adapter_client_ssl.cnf  client_rsa_certificate.crt
[client]$
```

4. Repeat the steps 1 up to 3 for all other services.
5. Create new certificates from the newly created CSR and follow the "Generate Certificate Signing Request (CSR)" procedure from step 5 in the *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide.*
6. Run the following commands to perform rollout restart of all OCNADD services:

```
kubectl rollout restart -n ocnadd-deploy deployment
```

```
kubectl rollout restart -n ocnadd-deploy statefulset
```

Where, ocnadd-deploy is the namespace where OCNADD is deployed.

Only Secrets are Generated

If only the secrets are generated:

1. Generate new certificates and follow step 3 of the "Generate Certificate and Private Keys" procedure in the *Oracle Communications Network Analytics Data Director Installation, Upgrade, and Fault Recovery Guide* to recreate the secrets.
2. Run the following commands to perform rollout restart of all OCNADD services:

```
kubectl rollout restart -n ocnadd-deploy deployment
```

```
kubectl rollout restart -n ocnadd-deploy statefulset
```

Where, ocnadd-deploy is the namespace where OCNADD is deployed.

4.1 Implementing Security Recommendations and Guidelines for OCNWDAF

This section provides specific recommendations and guidelines for Oracle Communications Network Data Analytics Function (OCNWDAF) security.

4.1.1 Common Security Recommendations and Procedures

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

4.1.1.1 4G and 5G Application Authentication and Authorization

Cloud Native Core NFs support integration with platform service meshes and 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 establishing 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.

4.1.1.2 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.

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 :

Table 4-1 Modify MySQL NDB Root Password

Step	Description	Est Time
1.	Log into the next query node using ssh: `$ ssh admusr@<mysql query node>`	1m
2.	Execute the following command to make the node as root: `$ sudo su`	1m
3.	Invoke mysql using existing DB Root credentials: `# mysql -h 127.0.0.1 -uroot -p` Enter password: `<enter existing root password>`	1m
4.	Change the DB Root credentials: `mysql> ALTER USER'root'@'localhost'IDENTIFIED BY'<NEW_PASSWORD>';` `mysql> FLUSH PRIVILEGES;`	1m
5	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:

Table 4-2 Configure TLS for MySQL NDB Query Nodes

Step	Description	Est Time
1.	Create private CA and a set of Keys/Certificate pairs for use in securing MySQL : `$ my_ssl_rsa_setup`	1m
2.	The available set of PEM files containing CA, server, and client certificates and keys 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>`	1m
4.	Login to the MySQL Query Node using ssh: `$ ssh admusr@<mysql query node>`	1m

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`
		1m

Table 4-2 (Cont.) Configure TLS for MySQL NDB Query Nodes

Step	Description	Est Time

6	Configure MySQL to use the new TLS certs and keys by executing the following command: `$ sudo vi /etc/my.cnf`
	Update the [mysqld] section as follows:
	
	`[mysqld]`
	`ssl-ca=/var/occnedb/opensslcerts/ca.pem`
	`ssl-cert=/var/occnedb/opensslcerts/server-cert.pem`
	`ssl-key=/var/occnedb/opensslcerts/server-key.pem`
	`tls_version=TLSv1.2`
	`ssl-cipher=DHE-RSA-AES128-GCM-SHA256`
		5m

7	Restart the mysql daemon: `$ sudo /etc/init.d/mysql.server restart`	1m
	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.

4.1.1.3 Cloud Native Core Ingress and Egress Gateways Specific Security Recommendations and Guidelines

This section provides Ingress and Egress Gateways specific security recommendations and guidelines.

> **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 and Egress Gateway**
- **Certificate Management and Dynamic Reload of Certificates in Gateways**

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_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 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.

4.1.2 Oracle Communications Networks Data Analytics Function (OCNWDAF) Specific Security Recommendations and Guidelines

This section provides specific recommendations and guidelines for Oracle Communications Networks Data Analytics Function (OCNWDAF) security.

> **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 *Oracle Communications Networks Data Analytics Function Installation and Fault Recovery Guide*.

> **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.

The procedures are:

- Oauth Token Validation Configuration
  - Rest Configuration
  - Public key Update for Changed Access Token
  - Disabling the Signature Validation for Oauth
- Create HTTPS Certificates
- Obtain List of Certificate Providers Allowed
- Configuring Kafka Security
- MySQL Configuration
- OCNWDAF Secret Configuration to Enable HTTPS

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 ocnwdaf 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 ocnwdaf
   ```

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 ocnwdaf
   ```

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

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

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:

```
“/nwdaf/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 ocnwdaf

   # 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.

Create HTTPS Certificates

Access to the OCNWDAF subscription endpoint is enabled through the gateway service. To access the endpoint, use a HTTPS version 2 **POST** request.

Endpoint address: `<path>nnwdaf-eventssubscription/v1/subscriptions`

1. To configure HTTPS version 2, enable the property in the configuration file and generate a SSL certificate. Configure the property in the project using the `.properties` file. See below example:
   Example:

   ```
   server.http2.enabled=true
   server.ssl.enabled=true
   server.ssl.key-store-type=PKCS12
   server.ssl.key-store=classpath:localhost.p12
   server.ssl.key-store-password=<password>
   ```

2. To create the certificate, run the following commands:

   ```
   openssl req -x509 -out localhost.crt -keyout localhost.key \
   -newkey rsa:2048 -nodes -sha256 \
   -subj '/CN=localhost' -extensions EXT -config <( \
   printf "[dn]\nCN=localhost\n[req]\ndistinguished_name = dn\n[EXT]
   \nsubjectAltName=DNS:localhost\nkeyUsage=digitalSignature\nextendedKeyUsage
   =serverAuth")
   ```

   Two files are generated - a `.crt` file and a `.key` file.

3. Use openssl to create a PKCS truststore file that contains both the certificate and key, see below example:
   Example:

   ```
   openssl pkcs12 -export -in localhost.crt -inkey localhost.key -name
   localhost -out localhost.p12
   ```

4. Configure the properties, see below example:
   Example:

   ```
   server.ssl.enabled=true
   server.ssl.key-store-type=PKCS12
   server.ssl.key-store=classpath:localhost.p12
   server.ssl.key-store-password=<password>
   ```

Obtain List of Certificate Providers Allowed

The following command lists the SSL certificate providers allowed:

```
cd JAVA_HOME/jre/lib/security
keytool -list -keystore cacerts
```

SSL Providers allow `jdk version 17.0.2`.

- actalisauthenticationrootca
- affirmtrustcommercialca
- affirmtrustnetworkingca
- affirmtrustpremiumca

- affirmtrustpremiumeccca
- amazonrootca1
- amazonrootca2
- amazonrootca3
- amazonrootca4
- baltimorecybertrustca
- buypassclass2ca
- buypassclass3ca
- camerfirmachambersca
- camerfirmachamberscommerceca
- camerfirmachambersignca
- cert_100_emsign_root_ca___g1100
- cert_101_emsign_ecc_root_ca___g3101
- cert_102_emsign_root_ca___c1102
- cert_103_emsign_ecc_root_ca___c3103
- cert_104_hongkong_post_root_ca_3104
- cert_106_microsoft_ecc_root_certificate_authority_2017106
- cert_107_microsoft_rsa_root_certificate_authority_2017107
- cert_108_e_szigno_root_ca_2017108
- cert_109_certsign_root_ca_g2109
- cert_110_trustwave_global_certification_authority110
- cert_111_trustwave_global_ecc_p256_certification_authority111
- cert_112_trustwave_global_ecc_p384_certification_authority112
- cert_113_naver_global_root_certification_authority113
- cert_114_ac_raiz_fnmt_rcm_servidores_seguros114
- cert_115_globalsign_root_r46115
- cert_116_globalsign_root_e46116
- cert_117_globaltrust_2020117
- cert_118_anf_secure_server_root_ca118
- cert_119_certum_ec_384_ca119
- cert_120_certum_trusted_root_ca120
- cert_121_tuntrust_root_ca121
- cert_122_harica_tls_rsa_root_ca_2021122
- cert_123_harica_tls_ecc_root_ca_2021123
- cert_124_autoridad_de_certificacion_firmaprofesional_cif_a62634068124
- cert_125_vtrus_ecc_root_ca125
- cert_126_vtrus_root_ca126
- cert_127_isrg_root_x2127

- cert_128_hipki_root_ca___g1128
- cert_129_globalsign_ecc_root_ca___r4129
- cert_130_gts_root_r1130
- cert_131_gts_root_r2131
- cert_132_gts_root_r3132
- cert_18_secure_global_ca18
- cert_19_comodo_certification_authority19
- cert_20_network_solutions_certificate_authority20
- cert_22_certigna22
- cert_24_certsign_root_ca24
- cert_25_netlock_arany__class_gold__f__tan__s__tv__ny25
- cert_26_hongkong_post_root_ca_126
- cert_27_securesign_rootca1127
- cert_28_microsec_e_szigno_root_ca_200928
- cert_30_autoridad_de_certificacion_firmaprofesional_cif_a6263406830
- cert_31_izenpe_com31
- cert_40_twca_root_certification_authority40
- cert_42_ec_acc42
- cert_43_hellenic_academic_and_research_institutions_rootca_201143
- cert_50_ca_disig_root_r250
- cert_51_accvraiz151
- cert_52_twca_global_root_ca52
- cert_54_e_tugra_certification_authority54
- cert_56_atos_trustedroot_201156
- cert_69_staat_der_nederlanden_ev_root_ca69
- cert_74_cfca_ev_root74, Feb 22, 2022, trustedCertEntry
- cert_75_oiste_wisekey_global_root_gb_ca75
- cert_76_szafir_root_ca276
- cert_77_certum_trusted_network_ca_277
- cert_81_ac_raiz_fnmt_rcm81
- cert_86_tubitak_kamu_sm_ssl_kok_sertifikasi___surum_186
- cert_87_gdca_trustauth_r5_root87
- cert_88_trustcor_rootcert_ca_188
- cert_89_trustcor_rootcert_ca_289
- cert_90_trustcor_eca_190
- cert_94_ssl_com_ev_root_certification_authority_ecc94
- cert_96_oiste_wisekey_global_root_gc_ca96
- cert_97_uca_global_g2_root97

- cert_98_uca_extended_validation_root98
- cert_99_certigna_root_ca99
- certumca
- certumtrustednetworkca
- chunghwaepkirootca
- comodoaaaca
- comodoeccca
- comodorsaca
- digicertassuredidg2
- digicertassuredidg3
- digicertassuredidrootca
- digicertglobalrootca
- digicertglobalrootg2
- digicertglobalrootg3
- digicerthighassuranceevrootca
- digicerttrustedrootg4
- dtrustclass3ca2
- dtrustclass3ca2ev
- entrust2048ca
- entrustevca
- entrustrootcaec1
- entrustrootcag2
- entrustrootcag4
- geotrustglobalca
- geotrustprimaryca
- geotrustprimarycag2
- geotrustprimarycag3
- geotrustuniversalca
- globalsignca
- globalsigneccrootcar4
- globalsigneccrootcar5
- globalsignr3ca
- globalsignrootcar6
- godaddyclass2ca
- godaddyrootg2ca
- haricaeccrootca2015
- haricarootca2015
- identrustcommercial

- identrustpublicca
- letsencryptisrgx1
- luxtrustglobalroot2ca
- quovadisrootca1g3
- quovadisrootca2
- quovadisrootca2g3
- quovadisrootca3
- quovadisrootca3g3
- secomscrootca1
- secomscrootca2
- securetrustca
- sslrooteccca
- sslrootevrsaca
- sslrootrsaca
- starfieldclass2ca
- starfieldrootg2ca
- starfieldservicesrootg2ca
- swisssigngoldg2ca
- swisssignplatinumg2ca
- swisssignsilverg2ca
- teliasonerarootcav1
- thawteprimaryrootca
- thawteprimaryrootcag2
- thawteprimaryrootcag3
- ttelesecglobalrootclass2ca
- ttelesecglobalrootclass3ca
- usertrusteccca
- usertrustrsaca
- verisignclass3g3ca
- verisignclass3g4ca
- verisignclass3g5ca
- verisignuniversalrootca
- xrampglobalca Kafka security configuration

Configuring Kafka Security

Use the following procedure to configure basic Kafka security:

Prerequisite: SSL certificates and keys.

Configuring Kafka Brokers

1. Configure Kafka to use TLS or SSL encryption, edit the configuration file `server.properties`. This file is usually stored in the Kafka config directory.

   > **Note**
   >
   > Login as *su* to ensure the ownership of the file is not changed.

   ```
   kafka config cmd

   /opt/kafka/config# su -s /bin/bash kafka
   ```

2. Add the following properties for each broker configuration, see below example:
   Example:

   ```
   listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
   advertised.listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
   ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks

   ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks

   ssl.truststore.password=<secret_value>

   ssl.keystore.password=<secret_value>

   ssl.key.password=<secret_value>

   security.inter.broker.protocol=SSL
   ```

   In the above sample configurations, PLAINTEXT and SSL protocols are used for the SSL enabled brokers.

3. Complete the above process for each broker, then restart the Kafka Cluster. Run the following scripts to reset the Kafka Cluster:

   ```
   kafka-server-stop.sh
   kafka-server-start.sh
   ```

4. Enable Client Authentication (two way authentication), run the following command:

   ```
   ssl.client.auth=required
   ```

Configuring Kafka Client

1. On the client side both certificates and keys are required. Configure all the clients communicating with SSL enabled Kafka Cluster. See below example:
   Example:

   ```
   bootstrap.servers=kafka-broker-host-name:9093

   security.protocol=SSL
   ```

```
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks

ssl.truststore.password=<secret_value>

ssl.keystore.location=/var/private/ssl/kafka.client.keystore.jks

ssl.keystore.password=<secret_value>

ssl.key.password=<secret_value>
```

Where `kafka-broker-host-name` is the FQDN of the broker.

ACL authorization

1. To enable and use the `AclAuthorizer`, set its full class name for your broker configuration in file `server.properties`.
2. Run the following command:

   ```
   authorizer.class.name=kafka.security.authorizer.AclAuthorizer
   ```

3. Kafka ACLs control the principals that perform operations on Kafka resources. Kafka brokers can use ZooKeeper ACLs by enabling broker configuration in the properties file.

   ```
   zookeeper.set.acl=true
   ```

4. Create ACLs for every topic, see below example:
   Example:

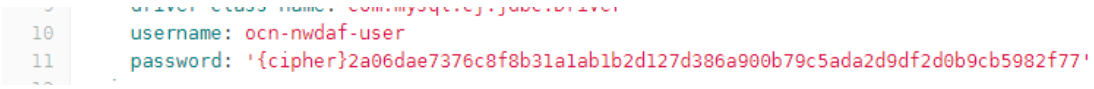
   ```
   kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
   configs.conf --add \
   --allow-principal User:* --operation All --topic mesa.report.location
   ```

MySQL Configuration

DBTier is used for DB connections. This ensures security issues are avoided and the data is encrypted.

OCNWDAF database passwords are encrypted using cipher.

Figure 4-1 OCNWDAF database password



OCNWDAF Secret Configuration to Enable HTTPS

Use the following procedure to create Kubernetes secret for HTTPS:

Prerequisites:

- ECDSA private key and CA signed certificate of CNCC (if initial algorithm is ES256)
- RSA private key and CA signed certificate of CNCC (if initial algorithm is RSA256)

- TrustStore password file
- KeyStore password file
- CA certificate

> **Note**
>
> Private Keys, certificates, passwords are created at the discretion of users.

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

3. Create `nwdaf-tls secret` based on TLS certificates:

   ```
   K8_NAMESPACE=<insert-your-namespace>
   CERT_PATH=<certificate-path>
   kubectl create secret generic nwdaf-tls \
   --save-config \
   --dry-run=client \
   --from-file=${CERT_PATH}/nwdaf.crt --from-file=${CERT_PATH}/nwdaf.key -o
   yaml | \
   kubectl -n ${K8_NAMESPACE} apply -f -
   ```

   > **Note**
   >
   > `.crt` and `.key` files should be named `nwdaf.crt` and `nwdaf.key`

4. After successful secret creation, the following message is displayed:

   ```
   secret/nwdaf-tls created
   ```

A
Network Port Flows

This section describes network port flows for CNE and OCNWDAF.

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	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			Repository Access	Access repositories (YUM, Docker, Helm, and so on.)
MySQL Query	MySQL SQL Node	3306/TCP	N		Microservice SQL Access	The SQL Query interfaces are used for NWDAF to access the database.
ETCD Client	Kubernetes Master Nodes	2379/TCP	Y		Client Access	Keystore DB used by Kubernetes
Kubelet API	Kubernetes Nodes	10250/TCP	Y		Control Plane Node Access	API which allows full node access.
Kubelet 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

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	5775/UDP	N		Agent	Accepts zipkin.thrift in compact Thrift protocol (deprecated; only used by very old Jaeger clients, circa 2016).
Jaeger Agent	Kubernetes Nodes	5778/TCP	N		Agent	Serves SDK configs, namely sampling strategies at /sampling.
Jaeger Agent	Kubernetes Nodes	6831/UDP	N		Agent	UDP Accepts jaeger.thrift in compact Thrift protocol used by most current Jaeger clients.
Jaeger Agent	Kubernetes Nodes	6831/UDP	N		Agent	UDP Accepts jaeger.thrift in binary Thrift protocol used by Node.js Jaeger client (because thriftrw npm package does not support compact protocol).
Jaeger Agent	Kubernetes Nodes	14271/TCP	N		Agent	Admin port: health check at / and metrics at /metrics.
Jaeger Collector	Kubernetes Nodes	9411/TCP	N		Collector	Accepts Zipkin spans in Thrift, JSON and Proto (disabled by default).
Jaeger Collector	Kubernetes Nodes	14250/TCP	N		Collector	Used by jaeger-agent to send spans in model.proto format.
Jaeger Collector	Kubernetes Nodes	14268/TCP	N		Collector	Accepts spans directly from clients in jaeger.thrift format with binary thrift protocol (POST to /api/traces). Also serves sampling policies at /api/sampling, similar to Agent's port 5778.
Jaeger Collector	Kubernetes Nodes	14269/TCP	N		Collector	Admin port: health check at / and metrics at /metrics.
Jaeger-Query	Kubernetes Nodes	80/TCP	N	GUI		Service frontend
Prometheus Server	Kubernetes Nodes	80/TCP	N	GUI		Prometheus Server

Table A-1 (Cont.) CNE Port Flows

Name	Server/ Container	Ingress Port ext[:int]/ Proto	TLS	Cluster IP (Service IP)	Node IP	Notes
Prometheus-Exporter	Kubernetes Nodes	9100/TCP	N		Prometheus Exporter	Prometheus Exporter
Alertmanager	Kubernetes Nodes	80/TCP	N	GUI		The Alertmanager handles alerts sent by client applications such as the Prometheus server.

OCNWDAF Port Flows

Table A-2 OCNWDAF Port Flows

Name	Server / Container	Ingress Port [external]:internal	TLS	Cluster IP (Service IP)	Node IP	Notes
NWDAF	Kubernetes Nodes/ NWDAF Service	8080/TCP	Y	Ingress Gateway	ocn-nwdaf-gateway-service	NWDAF
NWDAF Portal	Kubernetes Nodes/ NWDAF Service	80/TCP	Y	GUI	nwdaf-portal	NWDAF GUI web.