Oracle® Communications Converged Charging System

Network Bridge Cloud Native Installation and Administration Guide

Release 2.0
F61229-02
May 2024

ORACLE

Oracle Communications Converged Charging System Network Bridge Cloud Native Installation and Administration Guide, Release 2.0

F61229-02

Contents

9 Monitoring Network Bridge Processes

10 Using Network Bridge Logging

Preface

This guide describes how to install and administer Oracle Communications Network Bridge on a cloud native environment.

Audience

This guide is intended for anyone who installs, configures, administers, customizes, or uses Network Bridge.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Overview of the Network Bridge Cloud Native Deployment

Oracle Communications Network Bridge is a cloud native application in a containerized and orchestrated deployment architecture.

For more information about Network Bridge, see "About Network Bridge" in *Network Bridge User Guide*.

Topics in this document:

About the Network Bridge Cloud Native Deployment

Network Bridge is available as a cloud native deployment, supporting a Kubernetes-orchestrated containerized multiservice architecture to facilitate continuous integration, continuous delivery, and DevOps practices.

Network Bridge cloud native supports the following deployment models:

- On Private Kubernetes Cluster: Network Bridge cloud native can run on a general deployment of Kubernetes.
- On Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE): Network Bridge cloud native can run on Oracle's hosted Kubernetes OKE service.
- On Oracle Cloud Native Environment: Network Bridge cloud native can run on the Oracle Cloud Native Environment.

Network Bridge Cloud Native Architecture

In the Network Bridge cloud native architecture, each component runs as a container and is deployed as a Kubernetes pod, which is the fundamental building block of Kubernetes. Many core Network Bridge components can be deployed and managed as multiple replicas within a Kubernetes replica set.

About Network Bridge Pods

Table 1-1 lists the pods for Network Bridge whose containers are created and services are exposed through them.

Table 1-1 Network Bridge Pods

Pod	Replica Type	Admin Port	Container Port	Service Type
diameter-adapter-d2h	Multiple	8081/TCP	8080/TCP 1408/TCP	ClusterIP
diameter-adapter-h2d	Multiple	8081/TCP	8080/TCP 1408/TCP	ClusterIP
diameter-proxy	Multiple	8081/TCP	8080/TCP 1408/TCP	ClusterIP
diameter-proxy-db-job	Single	8081/TCP	N/A	ClusterIP
egress	Multiple	8081/TCP	8080/TCP 1408/TCP	ClusterIP
mediation	Multiple	8081/TCP	8080/TCP 1408/TCP	ClusterIP
oc-ccs-ndb-mgmd	Single	8081/TCP	1186/TCP	ClusterIP
oc-ccs-ndb-mysqld	Single	8081/TCP	3306/TCP	ClusterIP
oc-ccs-ndb-ndbmtd	Single	8081/TCP	1186/TCP	ClusterIP

Part I
Installing Network Bridge

This part provides information about configuring and deploying Oracle Communications Network Bridge in your cloud native environment. It contains the following chapters:

2

Setting Up Prerequisite Software

You can perform prerequisite tasks, such as installing Kubernetes and Helm, before deploying Oracle Communications Network Bridge on your cloud native environment.

Topics in this document:

> **Caution:**
>
> Oracle does not provide support for any prerequisite third-party software installation or configuration. The customer must handle any installation or configuration issues related to non-Oracle prerequisite software.

Network Bridge Prerequisite Tasks

As part of preparing your environment for Network Bridge cloud native, you choose, install, and set up external applications and services in ways that are best suited for your cloud native environment. The following shows the high-level prerequisite tasks:

1. Ensure you have downloaded the latest supported software that is compatible with Network Bridge cloud native. See "Network Bridge Cloud Native Software Compatibility" in *CCS Compatibility Matrix*.
2. Create a Kubernetes cluster.
3. Install a container platform supported by Kubernetes, such as Docker, Podman, or containerd.
4. Install Helm.
5. Install MySQL NDB Operator.
6. Install an ingress controller.
7. If you plan to trace the flow of API calls through Network Bridge, install and configure Jaeger.

For more information about tracing, see "Tracing the Flow of API Calls".

8. If you plan to autoscale your pods using Kubernetes Horizontal Pod Autoscaler:
   - Install and configure Kubernetes Metrics Server.
   - Install and configure a service mesh, such as Istio.

   For more information about autoscaling, see "Setting up Autoscaling of Network Bridge Pods".
9. If you plan to monitor Network Bridge operations:
   - Install and configure Prometheus Operator.
   - Install and configure Grafana.

   For more information, see "Monitoring Network Bridge Processes".

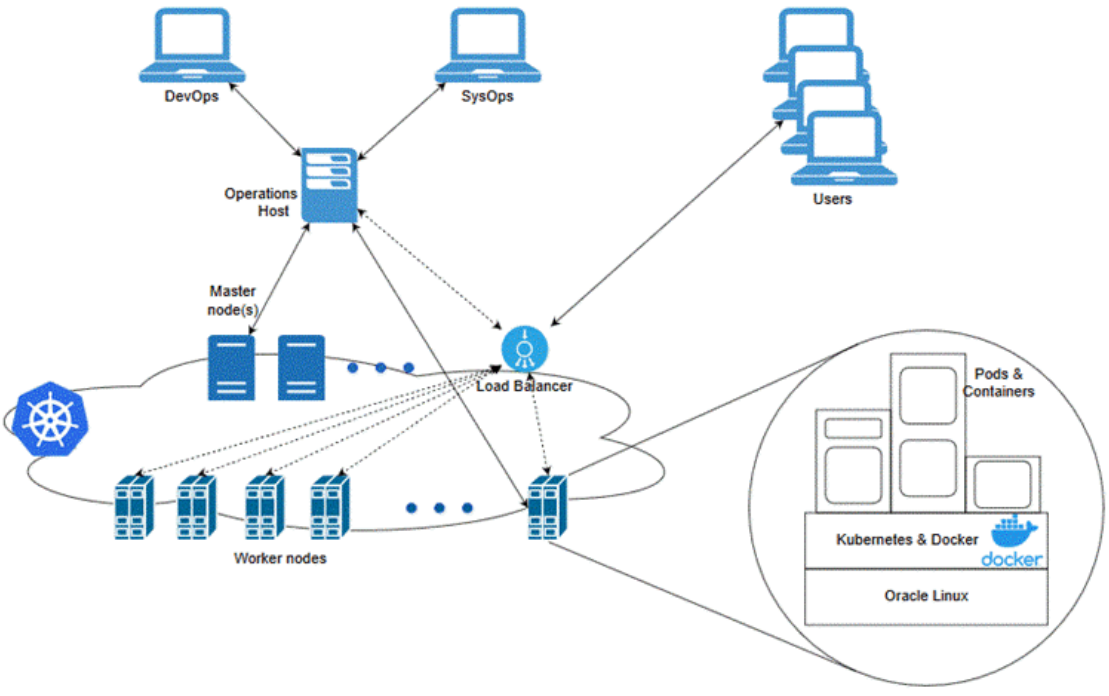
Prepare your environment with these technologies installed, configured, and tuned for performance, networking, security, and high availability. Make sure backup nodes are available in case of system failure in any of the cluster's active nodes.

Creating a Kubernetes Cluster

Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. It groups containers into logical units for easy management and discovery. When you deploy Kubernetes, you get a physical cluster with machines called nodes. A reliable cluster must have multiple worker nodes spread over separate physical infrastructure, and a very reliable cluster must have multiple primary nodes spread over separate physical infrastructure.

Figure 2-1 illustrates the Kubernetes cluster and the components that it interacts with.

Figure 2-1 Overview of the Kubernetes Cluster



Set up a Kubernetes cluster for your Network Bridge cloud native deployment, securing access to the cluster and its objects with the help of service accounts and proper authentication and authorization modules. Also, set up the following in your cluster:

- **Volumes**: Volumes are directories accessible to the containers in a pod and provide a way to share data. The Network Bridge cloud native deployment package uses persistent volumes for sharing data in and out of containers but does not enforce any particular type. You can choose from the volume type options available in Kubernetes.
- **A networking model**: Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. Every pod gets its own IP address, so you do not need to explicitly create a link between pods or map container ports to host ports. Several implementations are available that meet the fundamental requirements of Kubernetes' networking model. Choose the networking model depending on the cluster requirement.

Typically, you don't use Kubernetes nodes directly to run or monitor Kubernetes workloads. Instead, you reserve worker node resources to run the Kubernetes workload. Multiple cluster users (manual and automated) require a point from which to access and operate the cluster. For example, you can use **kubectl** commands or Kubernetes APIs. For this purpose, set aside a separate host or set of hosts. You can restrict operational and administrative access to the Kubernetes cluster to these hosts. To reduce cluster exposure and promote the traceability of actions, give specific users named accounts on these hosts.

Typically, the Continuous Delivery pipeline automation deploys directly on a set of operations hosts or leverages runners deployed on operations hosts. These hosts must run Linux, with all interactive-use packages installed to support tools such as Bash, Wget, cURL, Hostname, Sed, AWK, cut, and grep. An example is the Oracle Linux 7.6 image on Oracle Cloud Infrastructure.

In addition, you need the appropriate tools to connect to your overall environment, including the Kubernetes cluster. For instance, for a Container Engine for Kubernetes (OKE) cluster, you must install and configure the Oracle Cloud Infrastructure Command Line Interface.

Additional integrations may need to include LDAP for users to log in to this host, appropriate NFS mounts for home directories, security lists, firewall configuration for access to the overall environment, and so on.

For more information about Kubernetes, see "Kubernetes Concepts" in the Kubernetes documentation.

Installing Docker

Use the Docker platform to containerize CCS products. Install Docker Engine to use the prebuilt images from the Network Bridge cloud native deployment package.

You can use Docker Engine or any container runtime that supports the Open Container Initiative if it supports the Kubernetes version specified in "Network Bridge Cloud Native Software Compatibility" in *CCS Compatibility Matrix*.

Installing Helm

Helm is a package manager that helps you install and maintain software on a Kubernetes system. In Helm, a package is called a *chart*, consisting of YAML files and templates rendered into Kubernetes manifest files. The Network Bridge cloud native deployment package includes Helm charts that help create Kubernetes objects, such as ConfigMaps, Secrets, controller sets, and pods, with a single command.

The Network Bridge package also includes a **values.yaml** file, which contains the default configuration for a cloud native deployment. You can change the Network Bridge configuration by creating an **override-values.yaml** file and modifying the keys and values you want to change. The settings in this file will override the default values when you deploy Network Bridge or update your Network Bridge release.

Helm leverages **kubeconfig** for users running the **helm** command to access the Kubernetes cluster. By default, this is **$HOME/.kube/config**. Helm inherits the permissions set up for this access into the cluster. If you configure role-based access control (RBAC), ensure you grant sufficient cluster permissions to users running Helm.

To install Helm, see the Helm installation documentation at: https://helm.sh/docs/intro/install/.

Installing MySQL NDB Operator

Network Bridge components use MySQL NDB Operator to store session information. Ensure that you install MySQL NDB Operator before deploying Network Bridge.

> **Note:**
>
> If you attempt to deploy Network Bridge before installing MySQL NDB Operator, you will receive an error message similar to the following:
>
> ```
> Error: INSTALLATION FAILED: unable to build kubernetes objects from release
> manifest: resource mapping not found for name: "ccs-ndb" namespace: "" from
> "": no matches for kind "NdbCluster" in version "mysql.oracle.com/v1"ensure
> CRDs are installed first
> ```

To deploy MySQL NDB Operator on your Network Bridge cloud native environment, run the following command:

```
helm install --repo https://mysql.github.io/mysql-ndb-operator/ ndb-operator
ndb-operator -n NdbOperator --create-namespace \
   --set image=container-registry.oracle.com/mysql/commercial-ndb-
operator:8.0.32
```

If successful, you should see something similar to this:

```
NAME: ndb-operator
LAST DEPLOYED: Fri Oct 28 03:42:38 2023
NAMESPACE: NdbOperator
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

For more information, see the MySQL NDB Operator README on the GitHub website: https://github.com/mysql/mysql-ndb-operator/blob/main/README.md.

Installing an Ingress Controller

Using an ingress controller exposes Network Bridge services outside the Kubernetes cluster and allows clients to communicate with Network Bridge. Ingress controllers monitor ingress

objects and act on the configuration embedded in these objects to expose Network Bridge HTTP and T3 services to the external network.

Adding an external load balancer provides highly reliable single-point access to the services exposed by the Kubernetes cluster. In this case, the ingress controller exposes the services on behalf of the Network Bridge cloud native instance. Using a load balancer removes the need to expose Kubernetes node IPs to the larger user base, insulates users from changes (in terms of nodes appearing or being decommissioned) to the Kubernetes cluster, and enforces access policies.

Add an ingress controller, such as NGINX, Istio, or Traefik, to your Network Bridge cloud native system that has:

- Path-based routing for the Kubernetes Cluster service.
- TLS enabled between the client and the load balancer to secure communications outside of the Kubernetes cluster.

After you install an ingress controller, you must define the rules for directing requests from the following 5G services to the mediation service and mediation port:

- **/nchf-convergedcharging/v3/***
- **/npcf-smpolicycontrol/v1/***

The following shows example rules for mapping requests to the mediation service and the mediation port (**8080**). Although this example is for NGINX, you can use any ingress controller.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-oc-ccs
  namespace: ingressNameSpace
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /nchf-convergedcharging/v3
        pathType: Prefix
        backend:
          service:
            name: mediation
            port:
              number: 8080
      - path: /npcf-smpolicycontrol/v1
        pathType: Prefix
        backend:
          service:
            name: mediation
            port:
              number: 8080
```

Installing Jaeger

The Jaeger tracing tool helps you trace the flow of messages through Network Bridge components, making it easier to troubleshoot issues.

To install Jaeger, see the Jaeger documentation at: https://www.jaegertracing.io/docs/latest/.

Installing Kubernetes Metrics Server

Metrics Server collects resource metrics from kubelets and exposes them through the Metrics API. These metrics are used by Kubernetes Horizontal Pod Autoscaler to automatically adjust the CPU and memory usage in your Network Bridge pods.

To install Metrics Server, see the Kubernetes Metrics Server documentation at: https://kubernetes-sigs.github.io/metrics-server/.

Installing Prometheus Operator

Prometheus Operator in an open-source toolkit that scrapes metric data from Network Bridge and then stores it in a time-series database. You use it to monitor the operation of Network Bridge processes. See "Monitoring Network Bridge Processes" for more information.

To install Prometheus Operator, see the prometheus-operator GitHub website at: https://github.com/prometheus-operator/prometheus-operator.

Installing Grafana

Grafana is an open-source tool for viewing metric data that is stored in Prometheus Operator. You can use the Grafana Dashboards shipped with Network Bridge to view Network Bridge performance data.

To install Grafana, see the Grafana Loki installation documentation at: https://grafana.com/docs/loki/latest/installation/helm/.

3

Preparing Your Network Bridge Cloud Native Environment

You can prepare your cloud native environment by downloading the cloud native deployment package for Oracle Communications Network Bridge, extracting the Helm charts, and loading the Network Bridge component images.

Topics in this document:

Tasks for Preparing Your Cloud Native Environment

As part of preparing your cloud native environment for Network Bridge, you download the Network Bridge package and load the necessary component images into your Kubernetes container system. The following shows the high-level tasks:

1. Download the Network Bridge cloud native deployment package to your cloud native environment.
2. Extract the Network Bridge Helm charts from the downloaded package.
3. Load the Network Bridge component images into your container runtime.
4. (Optional) Create Secrets for your container registry.

Downloading the Network Bridge Cloud Native Deployment Package

To download the Network Bridge cloud native deployment package, go to the Oracle software delivery website (https://edelivery.oracle.com). Search for and then download **Oracle Communications Converged Charging System 2.0.0.1.0**.

Table 3-1 lists the packages in the downloaded archive files. Replace *version* with the release number. For example, for release 2.0 Patch Set 1, the file name for Oracle Communications Network Bridge Diameter Adapter would be **oc-ccs-diameter-adapter-2.0.0.1.0.tar**.

Table 3-1 Network Bridge Component Packages

Component Package Name	File Name
Oracle Communications Network Bridge Database Client	**oc-ccs-db-client-*version*.tar**
Oracle Communications Network Bridge Diameter Adapter	**oc-ccs-diameter-adapter-*version*.tar**

Table 3-1 (Cont.) Network Bridge Component Packages

Component Package Name	File Name
Oracle Communications Network Bridge Diameter Proxy	**oc-ccs-diameter-proxy-***version***.tar**
Oracle Communications Network Bridge Mediation	**oc-ccs-mediation-***version***.tar**
Oracle Communications Network Bridge Helm Chart	**oc-ccs-helm-chart-***version***.tgz**

Extracting the Helm Chart

Extract the Network Bridge Helm chart by running this command:

```
oc-ccs-helm-chart-version.tgz
```

Loading Network Bridge Component Images

The images shipped with the Network Bridge cloud native deployment package are in the form of TAR files. After downloading the TAR files, load them as images into your container runtime.

For example, to load the Mediation image into a Docker system, you would do this:

1. Download the **oc-ccs-mediation-***version***.tar** file to the system where Docker is installed, where *version* is the release number such as **2.0.0.1.0**.
2. Load the Mediation image into Docker by entering this command:

   ```
   docker load --input oc-ccs-mediation-version.tar
   ```

3. Verify that the image is loaded correctly by entering this command:

   ```
   docker images mediation:version
   ```

   The image details should be listed in one row.

If you use an internal registry to access images from different Kubernetes nodes, push the images from the local system to the registry server. For example, if the registry is identified by *RepoHost*:*RepoPort*, you'd push the Mediation image to the registry like this:

1. Tag the image with the registry server by entering this command:

   ```
   docker tag mediation:version RepoHost:RepoPort/oracle/cagbu/ccs/network-
   bridge/mediation:version
   ```

2. Push the image to the registry server by entering this command:

   ```
   docker push RepoHost:RepoPort/oracle/cagbu/ccs/network-bridge/
   mediation:version
   ```

Creating Secrets for Container Registry Authorization

You can automatically pull images from your private container registry by creating an **ImagePullSecrets**, which contains a list of authorization tokens (or Secrets) for accessing a private container registry. You then add references to the **ImagePullSecrets** in your **override-values.yaml** file for the Network Bridge's Helm chart. This allows pods to submit the Secret to the private container registry whenever they want to pull images.

To automatically pull images from a private container registry, create a Secret outside of the Helm chart by entering this command:

```
kubectl create secret docker-registry SecretName --docker-
server=RegistryServer --docker-username=UserName --docker-password=Password -
n NameSpace
```

where:

- *SecretName* is the name of your Kubernetes Secret.
- *RegistryServer* is the fully qualified domain name (FQDN) of your private container registry (*repoHost*:*repoPort*).
- *UserName* and *Password* are the user name and password for your private container registry.
- *NameSpace* is the namespace you will use for installing the Network Bridge Helm chart.

For example:

```
kubectl create secret docker-registry cgbu-docker-registry --docker-
server=mydockerimages.com:0000/ --docker-username=xyz --docker-
password=password -n NetworkBridge
```

4

Configuring Network Bridge for 5G to 4G Payload Transformation

You can configure Oracle Communications Network Bridge to perform protocol translation between a 5G Network Function (NF) and a 4G Network Element (NE). To do so, you configure an **override-values.yaml** file and then deploy Network Bridge on a cloud native environment.

Topics in this document:

About Configuring Network Bridge for 5G-to-4G Payload Conversion

You can use Network Bridge to dynamically transform HTTP usage requests from 5G networks into the Diameter protocol so they can be processed by a 4G LTE online charging system (OCS) or a policy-driven charging system. Network Bridge converts the usage requests according to the rules defined in the Network Bridge rule files.

> **Note:**
>
> The Network Bridge package includes rule files that support the following conversions. You do not need to configure or customize the rule files for 5G NF to 4G NE conversion.
>
> - Interworking between 5GC SMF and EPC OCS – N40 interface to Gy interface
> - Interworking between 5GC SMF and EPC PCRF – N7 interface to Gx interface

Configuring Network Bridge for 5G NF to 4G NE conversion includes these high-level steps:

1. Creating an **override-values.yaml** file for the Network Bridge Helm chart.
2. Configuring the mandatory values for Network Bridge.

   See "Configuring Mandatory Values for 5G-to-4G Conversion".
3. (Optional) Configuring one or more of the Network Bridge components. These components are preconfigured to work for most systems, but you may want to tune their values for a production system.

> **Note:**
>
> This step is not required for simple or demonstration systems.

See "Configuring Network Bridge Components".

4. (Optional) Configuring your components to support autoscaling.

   See "Setting up Autoscaling of Network Bridge Pods".

5. (Optional) Configuring your components for tracing using Jaeger.

   See "Tracing the Flow of API Calls".

6. (Optional) Configuring Prometheus to scrape metric data from your components for display in Grafana Dashboards.

   See "Monitoring Network Bridge Processes".

7. Deploy Network Bridge on your cloud native environment.

   See "Deploying Network Bridge".

Configuring Mandatory Values for 5G-to-4G Conversion

To configure Network Bridge for 5G NF to 4G NE conversion, set the following keys in your **override-values.yaml** file for **oc-ccs-helm-chart-***version*. These keys are required to create a simple or demonstration version of Network Bridge that you can use immediately.

These keys apply to all Network Bridge components, but you can override them for a specific component by setting a different value in the component's key. For example, to use a different pull policy for the Mediation component, set the **network-bridge.mediation.mediation.imagePullPolicy** key.

> **Note:**
>
> The **&keyName** and ***keyName** values are Helm references. If a key in the **values.yaml** file includes one of these references, you must also include the reference in your **override-values.yaml** file.

```
imageRepository: &imageRepository "my.example.com:9999"
imagePullSecrets: &imagePullSecrets Secret
imagePullPolicy: &imagePullPolicy IfNotPresent
protocolTranslationFilePath: &protocolTranslationFilePath "config/
protocolTranslation"
nodeSelector: &nodeSelector {}

partOfLabel: &partOfLabel oc-ccs

loggingPattern: &loggingPattern "%d{ISO8601_OFFSET_DATE_TIME_HHCMM} | %5p |
%X{traceId} | %-20.20thread | %-25.25logger{25} | %m%n"

tracingEnabled: &tracingEnabled true
tracingHost: &tracingHost jaeger-example
```

```
tracingPort: &tracingPort 14250
serviceMonitorEnabled: &serviceMonitorEnabled true

grafanaDashboardsEnabled: &grafanaDashboardsEnabled true
grafanaNamespace: &grafanaNamespace kube-logging

# Component images
mediationImage: &mediationImage "ccs/network-bridge/mediation:2.0.0.1.0"
egressImage: &egressImage "ccs/network-bridge/mediation:2.0.0.1.0"
diameterAdapterImage: &diameterAdapterImage "ccs/network-bridge/diameter-
adapter:2.0.0.1.0"
diameterProxyImage: &diameterProxyImage "ccs/network-bridge/diameter-
proxy:2.0.0.1.0"
dbClientImage: &dbClientImage "ccs/infra/db-client:2.0.0.1.0"
dbClusterImage: &dbClusterImage "container-registry.oracle.com/mysql/
commercial-cluster:8.0.36"
restProxyImage: &restProxyImage "ccs/network-bridge/mediation:2.0.0.1.0"

grafanaDashboards: &grafanaDashboards
  enabled: *grafanaDashboardsEnabled
  grafanaNamespace: *grafanaNamespace
  labels:
    grafana_dashboard: "1"
    release: prometheus
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"

serviceMonitor: &serviceMonitor
  enabled: *serviceMonitorEnabled
  additionalLabels:
    release: prometheus
  namespace: SmNameSpace
```

Table 4-1 describes each key.

Table 4-1 Mandatory Network Bridge Keys

Key	Description
imageRepository	Specifies the registry server where you pushed images, typically in this format: *RepoHost*:*RepoPort*. This value is added as a prefix to all image names when you install or upgrade Helm charts.
imagePullSecrets	Specifies the name of the Secret that contains credentials for accessing images from your private image server. The default is **regcred**. This value is added to each pod to permit it to pull the image from your private registry server. See "Creating Secrets for Container Registry Authorization" for more information.
imagePullPolicy	Specifies when Kubernetes pulls images: • **Always**: It always pulls an image from the repository. • **IfNotPresent**: It pulls an image only when one does not exist on a node. This is the default. • **Never**: It never pulls the image from the repository.

Table 4-1 (Cont.) Mandatory Network Bridge Keys

Key	Description
protocolTranslationFilePath	Specifies the directory in which the protocol translation files are located. The default is **config/protocolTranslation**.
nodeSelector	Specifies the nodes on which to deploy Network Bridge components. Labels appear underneath **nodeSelector** in a set of key-value pairs.
partOfLabel	Specifies the value to assign to the component's **app.kubernetes.io/part-of label**. This label identifies the application that the resource belongs to. The default is **oc-ccs**.
loggingPattern	Specifies the format and syntax of log files.
tracingEnabled	Specifies whether tracing through Jaeger is enabled. The default is **false**.
tracingHost	Specifies the host name for the Jaeger server. The default is **jaeger-all-in-one.jaeger.svc.cluster.local**. **Note:** This key applies only if tracing is enabled.
tracingPort	Specifies the port number for Jaeger. The default is **14250**. **Note:** This key applies only if tracing is enabled.
serviceMonitorEnabled	Specifies whether to enable Kubernetes ServiceMonitor, which is used to monitor a group of services. The default is **false**.
grafanaDashboardsEnabled	Specifies whether to enable Grafana Dashboards for Grafana Operator. The default is **false**.
grafanaNamespace	Specifies the namespace on which Grafana is deployed. The default is **kube-logging**. **Note:** This key applies only if Grafana Dashboards are enabled.
mediationImage	Specifies the name of the Mediation image. The default is **ccs/network-bridge/mediation:2.0.0.1.0**.
egressImage	Specifies the name of the Egress image. The default is **ccs/network-bridge/mediation:2.0.0.1.0**.
diameterAdapterImage	Specifies the name of the Diameter Adapter image. The default is **ccs/network-bridge/diameter-adapter:2.0.0.1.0**.
diameterProxyImage	Specifies the name of the Diameter Proxy image. The default is **ccs/network-bridge/diameter-proxy:2.0.0.1.0**.
dbClientImage	Specifies the name of the Database Client image. The default is **ccs/infra/db-client:2.0.0.1.0**.
dbClusterImage	Specifies the name of the Database Cluster image. The default is **container-registry.oracle.com/mysql/commercial-cluster:8.0.36**.
restProxyImage	Specifies the name of the REST Proxy image. The default is **ccs/network-bridge/mediation:2.0.0.1.0**.

Table 4-1 (Cont.) Mandatory Network Bridge Keys

Key	Description
grafanaDashboards.enabled	Specifies whether to enable Grafana Dashboards for Grafana Operator. The default is ***grafanaDashboardsEnabled**, which specifies to use the value from the **grafanaDashboardsEnabled** key.
grafanaDashboards.grafanaNamespace	Specifies the namespace on which Grafana is deployed. The default is ***grafanaNamespace**, which specifies to use the value from the **grafanaNamespace** key.
grafanaDashboards.labels.grafana_dashboard	Specifies the labels to add to Grafana CRDs. This helps Grafana discover the dashboards. The default is **1**. **Note:** This key applies only if Grafana Dashboards are enabled.
grafanaDashboards.labels.release	Specifies the release name on which Grafana Operator is deployed. The default is **prometheus**. **Note:** This key applies only if Grafana Dashboards are enabled.
grafanaDashboards.annotations.k8s-sidecar-target-directory	Specifies the directory in which the Grafana Dashboards are deployed. The default is **/tmp/dashboards/ccs**. **Note:** This key applies only if Grafana Dashboards are enabled.
serviceMonitor.enabled	Specifies whether to enable Kubernetes ServiceMonitor, which is used to monitor a group of services. The default is ***serviceMonitorEnabled**, which specifies to use the value from the **serviceMonitorEnabled** key.
serviceMonitor.additionalLabels.release	Specifies the release name on which Service Monitor is deployed. The default is **prometheus**.
serviceMonitor.namespace	Specifies the namespace in which the Service Monitor is deployed. **Note:** This key applies only if Service Monitor is enabled.

Configuring Network Bridge Components

The following Network Bridge components contain default values that will work for most test and demonstration systems, but you can configure them according to your business requirements.

- MySQL NDB component. For information about this component's keys, see https://github.com/mysql/mysql-ndb-operator/blob/release-8.0.36-1.0.5/docs/NdbCluster-CRD.md.
- Mediation component. For information about this component's keys, view the **oc-ccs/charts/oc-network-bridge-2.0.2.tgz/charts/oc-nb-mediation/values.schema.json** file in a Markdown viewer.
- Egress component. For information about this component's keys, view the **oc-ccs/charts/oc-network-bridge-2.0.2.tgz/charts/oc-nb-mediation/values.schema.json** file in a Markdown viewer.
- HTTP to Diameter Adapter component. For information about this component's keys, view the **oc-ccs/charts/oc-network-bridge-2.0.2.tgz/charts/oc-nb-diameter-adapter/values.schema.json** file in a Markdown viewer.
- Diameter to HTTP Adapter component. For information about this component's keys, view the **oc-ccs/charts/oc-network-bridge-2.0.2.tgz/charts/oc-nb-diameter-adapter/values.schema.json** file in a Markdown viewer.

- Diameter Proxy component. For information about this component's keys, view the **oc-ccs/charts/oc-network-bridge-2.0.2.tgz/charts/oc-nb-diameter-proxy/values.schema.json** file in a Markdown viewer.

5

Configuring Network Bridge for 5G-to-5G Payload Conversion

You can configure Oracle Communications Network Bridge to mediate payloads between two 5G Network Functions (NFs). To do so, you create a mutation rules file, configure an **override-values.yaml** file, and then deploy Network Bridge on a cloud native environment.

Topics in this document:

About Configuring Network Bridge for 5G-to-5G Payload Mediation

You can use Network Bridge to interconnect two 5G NFs that support different payload formats. In this case, Network Bridge acts as an N40 proxy, dynamically transforming HTTP message payloads according to your defined rules.

To configure Network Bridge to mediate payloads between 5G NFs:

1. Create an **override-values.yaml** file for the Network Bridge Helm chart.
2. Configure the mandatory values for 5G-to-5G payload conversion.

   See "Configuring Mandatory Values for 5G-to-5G Payload Conversion".
3. Configure the Mediation component for 5G-to-5G payload conversion.

   See "Configuring the Mediation Component for 5G-to-5G Payload Conversion".
4. Define how to mutate message payloads.

   See "Defining Mutation Rules for Payload Conversion".
5. (Optional) Define any business rules for modifying messages.

   See "Defining Business Rules in DRL Files".
6. (Optional) Configure the Mediation component to support autoscaling.

   See "Setting up Autoscaling of Network Bridge Pods".
7. (Optional) Configure your Mediation component for tracing using Jaeger.

   See "Tracing the Flow of API Calls".
8. (Optional) Configure Prometheus to scrape metric data from your Mediation component for display in Grafana Dashboards.

See "Monitoring Network Bridge Processes".

9. Deploy Network Bridge on your cloud native environment.

   See "Deploying Network Bridge".

Configuring Mandatory Values for 5G-to-5G Payload Conversion

To configure Network Bridge for 5G NF to 5G NF payload conversion, set the following keys in your **override-values.yaml** file for **oc-ccs-helm-chart-***version*. These keys are required to create a simple or demonstration version of Network Bridge that you can use immediately.

These keys apply to all Network Bridge components, but you can override a key's value for a specific component. For example, to use a different pull policy for the Mediation component, set the **network-bridge-restproxy.rest-proxy.imagePullPolicy** key to another value.

> **Note:**
>
> The **&keyName** and ***keyName** values are Helm references. If a key in the **values.yaml** file includes one of these references, you must also include the reference in your **override-values.yaml** file.

```
imageRepository: &imageRepository "repository.example.com:7840"
imagePullSecrets: &imagePullSecrets regcred
imagePullPolicy: &imagePullPolicy IfNotPresent
protocolTranslationFilePath: &protocolTranslationFilePath "config/
protocolTranslation"
nodeSelector: &nodeSelector

partOfLabel: &partOfLabel oc-ccs

loggingPattern: &loggingPattern "%d{ISO8601_OFFSET_DATE_TIME_HHCMM} | %5p |
%X{traceId} | %-20.20thread | %-25.25logger{25} | %m%n"

tracingEnabled: &tracingEnabled true
tracingHost: &tracingHost jaeger-all-in-one.jaeger.svc.cluster.local
tracingPort: &tracingPort 14268
serviceMonitorEnabled: &serviceMonitorEnabled true

grafanaDashboardsEnabled: &grafanaDashboardsEnabled true
grafanaNamespace: &grafanaNamespace kube-logging

grafanaDashboards:
  enabled: *grafanaDashboardsEnabled
  grafanaNamespace: *grafanaNamespace
  labels:
    grafana_dashboard: "1"
    release: prometheus
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"
serviceMonitor:
  enabled: *serviceMonitorEnabled
  namespace:
```

Table 5-1 describes each key.

Table 5-1 Mandatory Keys for 5G-to-5G Payload Conversion

Key	Description
imageRepository	Specifies the registry server where you pushed images, typically in this format: *RepoHost*:*RepoPort*. This value is added as a prefix to all image names when you install or upgrade Helm charts.
imagePullSecrets	Specifies the name of the Secret that contains credentials for accessing images from your private image server. The default is **regcred**. The Secret is added to each pod to permit it to pull the image from your private registry server. See "Creating Secrets for Container Registry Authorization" for more information.
imagePullPolicy	Specifies when Kubernetes pulls images: • **Always**: It always pulls an image from the repository. • **IfNotPresent**: It pulls an image only when one does not exist on a node. This is the default. • **Never**: It never pulls the image from the repository.
partOfLabel	Specifies the value to assign to the component's **app.kubernetes.io/part-of label**. This label identifies the application that the resource belongs to.
nodeSelector	Specifies the nodes on which to deploy Network Bridge components. Labels appear underneath **nodeSelector** in a set of key-value pairs.
loggingPattern	Specifies the format and syntax of logs.
tracingEnabled	Specifies whether tracing through Jaeger is enabled. The default is **false**.
tracingHost	Specifies the host name for the Jaeger server. The default is **jaeger-all-in-one.jaeger.svc.cluster.local**. This key only applies if tracing is enabled.
tracingPort	Specifies the port number for Jaeger. The default is **14268**. This key only applies if tracing is enabled.
serviceMonitorEnabled	Specifies whether to enable a Kubernetes Service Monitor, which can monitor a group of services. The default is **false**.
grafanaDashboardsEnabled	Specifies whether to enable Grafana Dashboards for Grafana Operator. The default is **false**.
grafanaNamespace	Specifies the namespace on which Grafana is deployed. The default is **kube-logging**. This key only applies if Grafana Dashboards are enabled.
grafanaDashboards.labels.grafana_dashboard	Specifies the labels to add to Grafana CRDs. Labels help Grafana discover dashboards. The default is **1**. This key applies only if Grafana Dashboards are enabled.
grafanaDashboards.labels.release	Specifies the release name on which Grafana Operator is deployed. The default is **prometheus**. This key only applies if Grafana Dashboards are enabled.
grafanaDashboards.annotations.k8s-sidecar-target-directory	Specifies the directory in which the Grafana Dashboards are deployed. The default is **/tmp/dashboards/ccs**. This key only applies if Grafana Dashboards are enabled.

Table 5-1 (Cont.) Mandatory Keys for 5G-to-5G Payload Conversion

Key	Description
serviceMonitor.namespace	Specifies the namespace in which the Service Monitor is deployed. This key only applies if Service Monitor is enabled.

Configuring the Mediation Component for 5G-to-5G Payload Conversion

To configure the Mediation component for 5G-to-5G payload conversion, set the following keys in your **override-values.yaml** file:

```
network-bridge-restproxy:
  enabled: true
  # Network Bridge Top Level
  name: network-bridge
  fullname: "network-bridge-restproxy"

  rest-proxy:
    enabled: true
    imageRepository: *imageRepository
    imagePullSecrets: *imagePullSecrets
    imagePullPolicy: *imagePullPolicy
    partOfLabel: *partOfLabel
    mediation:
      name: rest-proxy
      fullname: "rest-proxy"
      additionalLabels: { }
      annotations: { }
      nodeSelector: *nodeSelector
      client:
        connect-timeout-millis: 2000
        read-timeout-millis: 2000
        follow-redirects: true
        max-redirects: 5
        additionalAttributes: { }
      kieRulesConfigMapOverride: MyKieRules
      mutatorRulesConfigMapOverride: MyMutationRules

      restServices:
        - name: "all_sbi"
          config:
            path: "/{:npcf-smpolicycontrol\/v1\/sm-policies($|\/[^/]+\/
(update|delete)($|\/$))|nchf-convergedcharging\/v3\/chargingdata($|\/[^/]+\/
(update|release)($|\/$))|nchf-convergedcharging\/v3\/notify($|\?
notifyUri=.*)}"
          processors:
            - name: "kie_processor"
              type: KIE_PROCESSOR
              config:
                rulesPath: "/app/config/rules"
      grpcServices: [ ]
```

```
      container:
        image: *restProxyImage
      terminationGracePeriodSeconds: 60
      resources:
        memoryRequest: "256Mi"
        cpuRequest: "250m"
        memoryLimit: "2048Mi"
        cpuLimit: "500m"
      replicas: 1
      jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -
XX:InitialRAMPercentage=80 -XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -
Dlog4j.configurationFile=config/log4j2.yaml"
      restartCount: 0
      hpa:
        enabled: true
        minReplicas: 1
        maxReplicas: 8
        metrics:
          cpuAverageUtilization: 65
        scaleDown:
          selectPolicy: Max
          stabilizationWindowSeconds: 300
          periodSeconds: 180
        scaleUp:
          selectPolicy: Max
          stabilizationWindowSeconds: 5
          periodSeconds: 20
      service:
        type: ClusterIP
        additionalLabels: { }
        additionalAnnotations: { }
      adminService:
        type: ClusterIP
        additionalLabels: { }
        additionalAnnotations: { }
      logging:
        format:
          type: TEXT
          pattern: *loggingPattern
        rootLevel: INFO
        packageLogging:
          - name: com.oracle.cagbu
            level: INFO
          - name: io.helidon
            level: INFO
          - name: org.jboss
            level: INFO
          - name: io.jaegertracing.internal.reporters
            level: WARN
      tracing:
        enabled: *tracingEnabled
        service: mediation
        maxQueueSize: 42
        flushIntervalMs: 10001
        host: *tracingHost
        port: *tracingPort
```

```
      samplerType: ratio
      samplerParam: 0.01
      samplerManager:
      logSpans: true
      webServer:
        spans:
          httpRequest:
            contentWrite: false
            contentRead: false

  grafanaDashboards: *grafanaDashboards
  serviceMonitor: *grafanaDashboards

  ingress:
    ingressClassName:
```

Table 5-2 describes each key.

> **Note:**
>
> You can also view key descriptions by opening the **oc-ccs/charts/oc-network-bridge-0.2.0.tgz/charts/oc-nb-mediation/values.schema.json** file in a Markdown viewer.

Table 5-2 Mediation Component Keys

Key	Path in values.yaml	Description
enabled	**network-bridge-restproxy**	Specifies whether Network Bridge as a REST proxy is enabled. The default is **true**.
name	**network-bridge-restproxy**	Sets the name of the application. This is used to identify the application in the cluster. The default is **network-bridge**.
fullname	**network-bridge-restproxy**	Sets the full name of the application. The default is **network-bridge-restproxy**.
enabled	**network-bridge-restproxy.rest-proxy**	Enables the Mediation component as a REST Proxy. The default is **true**.
imageRepository	**network-bridge-restproxy.rest-proxy**	Specifies the registry server where you pushed images, typically in this format: *RepoHost*:*RepoPort*. **Note:** The component uses the global **imageRepository** setting by default. Set this key only if you want to use a different registry server.
imagePullSecrets	**network-bridge-restproxy.rest-proxy**	Specifies the name of the Secret that contains credentials for accessing images from your private image server. **Note:** The component uses the global **imagePullSecrets** setting by default. Set this key only if you want to use a different Secret.

Table 5-2 (Cont.) Mediation Component Keys

Key	Path in values.yaml	Description
imagePullPolicy	**network-bridge-restproxy.rest-proxy**	Sets when Kubernetes pulls images: **Always**, **IfNotPresent**, or **Never**. **Note:** The component uses the global **imagePullPolicy** setting by default. Set this key only if you want to use a different setting.
partOfLabel	**network-bridge-restproxy.rest-proxy**	Sets the value to assign to the component's **app.kubernetes.io/part-of** label. This label identifies the application that the resource belongs to. **Note:** The component uses the global **partOfLabel** setting by default. Set this key only if you want to use a different setting.
name	**network-bridge-restproxy.rest-proxy.mediation**	Sets the name of the component. This is used to identify the component in the cluster. The default is **rest-proxy**.
fullname	**network-bridge-restproxy.rest-proxy.mediation**	Sets the full name of the component. This name will be used in place of the *ReleaseName-Name* pattern. The default is **rest-proxy**.
additionalLabels	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the additional labels to add to a Kubernetes custom resource definition (CRD).
annotations	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the additional annotations to add to a Kubernetes CRD.
nodeSelector	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the nodes on which to deploy the mediation component. **Note:** The component uses the global **nodeSelector** setting by default. Set this key only if you want to use a different setting.
client.*	**network-bridge-restproxy.rest-proxy.mediation**	The details for configuring the client: • **connect-timeout-millis**: Sets the time to wait in milliseconds for a connection to be established with the service. The default is **2000**. • **read-timeout-millis**: Sets the time to wait in milliseconds to read a response from the service. The default is **2000**. • **follow-redirects**: Enables GET methods to follow redirects provided in the response. The default is **true**. • **max-redirects**: Sets the maximum number of chained redirects that are allowed. The default is **5**. • **additionalAttributes**: Specifies the additional attributes for configuring the web client.

Table 5-2 (Cont.) Mediation Component Keys

Key	Path in values.yaml	Description
kieRulesConfigMapOverride	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the name of the ConfigMap that contains your KIE rule file. If not specified, Network Bridge uses the rules defined in the default KIE rule file. This value applies to **KIE_PROCESSOR** type processors only. See "Defining Business Rules in DRL Files" for more information.
mutatorRulesConfigMapOverride	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the name of the ConfigMap that contains your mutation rule file. If not specified, Network Bridge uses the rules defined in the default mutation rule file. This value applies to **MUTATOR_PROCESSOR** type processors only. See "Defining Mutation Rules for Payload Conversion" for more information.
restServices.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies how to process REST service messages. See "REST Services Configuration for 5G-to-5G Payload Conversion" for more information.
grpcServices.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies how to process gRPC service messages.
image	**network-bridge-restproxy.rest-proxy.mediation.container**	Specifies the name of the REST Proxy image. **Note:** The component uses the global **restProxyImage** setting by default. Set this key only if you want to use a different setting.
terminationGracePeriodSeconds	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the amount of time, in seconds, the Mediation pod needs to shut down gracefully. The value must be a non-negative integer. A value of 0 specifies to terminate immediately, and a nil value specifies to use the default grace period. The grace period is the duration in seconds after which the processes running in the pod are sent a termination signal. The processes are forcibly halted with a kill signal. Set this value longer than the expected cleanup time for your process. The default is **60**.
resources.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the minimum and maximum amount of memory and CPU that can be used. See "Setting up Autoscaling of Network Bridge Pods" for more information.
replicas	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the desired number of pod replicas. The default is **1**. This key is ignored if HPA is enabled.
jvmOpts	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the JVM options to use when starting the Mediation component.

Table 5-2 (Cont.) Mediation Component Keys

Key	Path in values.yaml	Description
restartCount	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the number of times the service has been restarted. Increment this value by 1 and run the Helm upgrade command to force a rolling restart of the Mediation pods. The default is **0**.
hpa.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the minimum and maximum number of pod replicas that can be deployed, the scale-up rules, and the scale-down rules. See "Setting up Autoscaling of Network Bridge Pods" for more information.
type	**network-bridge-restproxy.rest-proxy.mediation.service**	Specifies how the service is exposed: • **NodePort**: Builds on ClusterIP and allocates a port on every node that routes to ClusterIP. • **ClusterIP**: Allocates a cluster-internal IP address for load balancing to endpoints. The default is **ClusterIP**.
type	**network-bridge-restproxy.rest-proxy.mediation.adminService**	Specifies how the admin service is exposed: • **NodePort**: Builds on ClusterIP and allocates a port on every node that routes to ClusterIP. • **ClusterIP**: Allocates a cluster-internal IP address for load balancing to endpoints. The default is **ClusterIP**.
logging.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies the logging levels for the Mediation component. See "Using Network Bridge Logging" for more information.
tracing.*	**network-bridge-restproxy.rest-proxy.mediation**	Specifies how to set up tracing for the Mediation component. See "Tracing the Flow of API Calls" for more information.
grafanaDashboards	**network-bridge-restproxy.rest-proxy**	Specifies your Grafana Dashboard settings. **Note:** The component uses the global **grafanaDashboards** keys and settings by default. Set this key only if you want to use different settings.
serviceMonitor	**network-bridge-restproxy.rest-proxy**	**Note:** The component uses the global **serviceMonitor** keys and settings by default. Set this key only if you want to use different settings.
ingressClassName	**network-bridge-restproxy.rest-proxy.ingress**	Specifies the Class Name for the Ingress Controller.

REST Services Configuration for 5G-to-5G Payload Conversion

You can configure the Mediation component to listen for REST services from a specific URL and then call one or more processors. To do so, add a **restServices** section under **network-bridge-restproxy.rest-proxy.mediation**. The **restServices** section uses this syntax:

```
restServices:
  - name: "Example Name"
    config:
      path: "*"
    processors:
      - ProcessorN
      - ProcessorN
      - ProcessorN
      ...
```

Replace *ProcessorN* with the keys for one or more of these processor types:

- **KIE processor**. See "Configuring a KIE Processor".
- **Mutation processor**. See "Configuring a Mutation Processor".
- **REST processor**. See "Configuring a REST Processor".
- **gRPC processor**. See "Configuring a GRPC Processor".

For an example of how to configure this section, see "Example Processor Configuration for REST Services".

Configuring a KIE Processor

You use a KIE processor to convert REST messages according to the rules defined in a DRL file. You configure the KIE processor by using these keys in the **restServices.processors** section of your **override-values.yaml** file:

```
- name: "kie_processor"
  type: KIE_PROCESSOR
  config:
    rulesPath:"/app/config/rules"
```

where:

- **name**: Specifies the name of the KIE processor instance.
- **config.rulesPath**: Specifies the system path to your DRL business rule files.

Configuring a Mutation Processor

You use a Mutation processor to transform REST request and response messages according to the rules defined in a mutation rule file. You configure the processor by using these keys in the **restServices.processors** section of your **override-values.yaml** file:

```
- name: mutation-response
  type: MUTATOR_PROCESSOR
  config:
    rulesPath: "src/test/resources/mutation/rules"
```

```
    ruleNames:
     - reWriteResponse
```

where:

- **name**: Specifies the name of the mutation processor instance.
- **config.rulesPath**: Specifies the file system path that contains your mutation rule file.
- **config.ruleNames**: Specifies the names of one or more rules to run against incoming REST messages. The listed names must match the rule names in your mutation rule file.

Configuring a REST Processor

You use a REST processor to optionally add a header to REST messages before sending them to a specified URI.

You can also add fault tolerance to the REST processor, which is achieved through a combination of retries and a circuit breaker. Retries automatically attempt to run the processor multiple times if it fails to respond. For example, if a REST processor request fails due to a temporary error, the processor can retry the call after a specified interval to check if the error has been resolved. Retries can increase the chances of successfully calling the service and reduce the overall impact of temporary errors.

The circuit breaker prevents cascading processor failures by automatically failing fast when a processor fails too often. The circuit breaker monitors the health of a processor by tracking the number of failures the processor encounters. If the number of failures exceeds a threshold, the circuit breaker trips, and all subsequent calls to the processor fail automatically without actually trying to run the processor. This prevents further failures and allows the system to recover more quickly. After a specified period of time, the circuit breaker is reset, and processor calls are allowed to go through.

You define the REST processor by using these keys in the **restServices.processors** section of your **override-values.yaml** file:

```
- name: sample-rest-processor
  type: REST_PROCESSOR
  config:
    uri: "http://sample.example.com"
    destUriHeader: dest_uri
  faultTolerance:
    retry:
      delay: 5
      calls: 3
      delayFactor: 1.0
    circuitBreaker:
      delay: 500
      errorRatio: 20
      successThreshold: 10
      volume: 2
```

where:

- **name**: Specifies the name of the REST processor instance.
- **config.uri**: Specifies the URI to send messages to.

- **config.destUriHeader**: Sets the metadata parameter that indicates the host to send REST requests to.
- **faultTolerance.retry.delay**: Sets the duration, in milliseconds, to wait before retrying the process in case of failure.
- **faultTolerance.retry.calls**: Sets the maximum number of times to retry the process in case of failure.
- **faultTolerance.retry.delayFactor**: Sets the multiplication factor of delay between two consecutive retries.
- **faultTolerance.circuitBreaker.delay**: Sets the duration, in milliseconds, to wait before retrying the process in case of failure.
- **faultTolerance.circuitBreaker.errorRatio**: Specifies the percentage of failures that will open the circuit breaker.
- **faultTolerance.circuitBreaker.successThreshold**: Specifies the number of successful calls that will close a half-open breaker.
- **faultTolerance.circuitBreaker.volume**: Specifies the size of the processing window.

Configuring a GRPC Processor

You use a gRPC processor to convert REST messages into an internal gRPC format for intercomponent communication.

> **Note:**
>
> This processor is not typically needed for 5G-to-5G payload conversion.

You configure the gRPC processor by using these keys in the **restServices.processors** section of your **override-values.yaml** file:

```
- name: "grpc"
  type: GRPC_PROCESSOR
  config:
    host: "egress"
    port: 1408
```

where:

- **name**: Specifies the name of the gRPC processor instance.
- **config.host**: Specifies the host name of the server to send messages to.
- **config.port**: Specifies the port to send messages to.

Example Processor Configuration for REST Services

The following example processor configuration for REST services specifies to do the following:

1. Listen for REST messages from a 5G SBI path.
2. Call **kie_processor** to convert incoming REST messages according to the DRL files in the **/app/config/rules** file path.

3. Call **mutation-request** to mutate incoming REST messages by applying the **addChargingInfo** and **addHeaders** rules from the mutation rule file stored in **src/test/resources/mutation/rules**.
4. Call **sample-rest-processor** to add the **destination:dest_uri** header to messages before sending them to the **http://sample.example.com** server. If a call to the server fails, **sample-rest-processor** will retry the call up to three times, five milliseconds apart.
5. When processing response messages from the 5G NF, call **mutation-response** to mutate REST messages by applying the **reWriteResponse** rule from the mutation rule file stored in **src/test/resources/mutation/rules**.

```
restServices:
  - name: "all_sbi"
    config:
      path: "/{:npcf-smpolicycontrol\/v1\/sm-policies($|\/[^/]+\/(update|
delete)($|\/$))|nchf-convergedcharging\/v3\/chargingdata($|\/[^/]+\/(update|
release)($|\/$))|nchf-convergedcharging\/v3\/notify($|\?notifyUri=.*)}"
    processors:
      - name: "kie_processor"
        type: KIE_PROCESSOR
        config:
          rulesPath:"/app/config/rules"
     - name: mutation-request
       type: MUTATOR_PROCESSOR
       config:
          rulesPath: "src/test/resources/mutation/rules"
          ruleNames:
           - addChargingInfo
           - addHeaders
      - name: sample-rest-processor
        type: REST_PROCESSOR
        config:
          uri: "http://sample.example.com"
          destUriHeader: dest_uri
        faultTolerance:
          retry:
            delay: 5
            calls: 3
            delayFactor: 1.0
      - name: mutation-response
        type: MUTATOR_PROCESSOR
        config:
          rulesPath: "src/test/resources/mutation/rules"
          ruleNames:
           - reWriteResponse
```

Defining Mutation Rules for Payload Conversion

For 5G-to-5G payload conversion, you create a mutation rule file that defines:

- The criteria that the HTTP message must meet to be transformed. If you include multiple requirements, the message must meet them all to be transformed.
- The transformation to perform on the HTTP message, such as changing a field in the request header or JSON body.

> **Note:**
>
> To configure Network Bridge as an N40 proxy, you must add three rules with the following names to the file: **addChargingInfo**, **addHeaders**, and **reWriteResponse**.

To define the rules for 5G payload conversion:

1. Create a mutation rule file in YAML format and open it in a text editor.

   Alternatively, you can start with the example **oc-ccs/charts/oc-network-bridge-0.2.0.tgz/charts/oc-nb-mediation/data/mutation-rules/exampleRule.yaml** file.

2. Add a rule name and a brief description. For example:

   ```
   ruleName: "addChargingInfo"
   ruleDescription: "My new rule adds a field to the JSON body."
   ```

3. Add a **criterion** section that defines one or more criteria that an HTTP message must meet. To specify that all HTTP messages meet the requirements, leave the **criterion** section empty.

   For example, this shows a simple **criterion** section that requires a message to contain a POST HTTP method:

   ```
   criterion:
      value:
         method:
            criteriaType: EQUAL
            value: POST
   ```

   For more information, see "Configuring the Criterion Section".

4. Add a **mutation** section that specifies how to transform the HTTP message.

   For example, this shows a simple **mutation** section that modifies the JSON body by changing the **ratingGroup** value to **15**.

   ```
   mutation:
     json:
       body:
       - mutationType: JSON_PATCH
         jsonPatch:
         - op: replace
           path: /multipleUnitUsage/ratingGroup
           value: "15"
   ```

   For more information, see "Configuring the Mutation Section".

5. Save and close the file.

   For more information about configuring the mutation rule file, see "Example Mutation Rule File".

   You can also view descriptions of each key in the mutation rule file by opening the **oc-ccs/charts/oc-network-bridge-0.2.0.tgz/charts/oc-nb-mediation/mutator.rules.schema.json** file in a Markdown viewer.

6. Create a ConfigMap from your mutation rule file by running this command:

   ```
   kubectl create configmap ConfigMapName --from-file=Path
   ```

   where *ConfigMapName* is your desired name for the ConfigMap, and *Path* is the directory that contains the mutation rule file.

7. Configure Network Bridge to use your mutation rule file by doing the following in your **override-values.yaml** file:

   - Set the **network-bridge-restproxy.rest-proxy.mediation.mutatorRulesConfigMapOverride** key to *ConfigMapName*.

     This configures Network Bridge to use your mutation rule file for all **MUTATOR_PROCESSOR** type processors.

   - Ensure you add a section to your **override-values.yaml** file for calling a **MUTATOR_PROCESSOR** type processor when processing REST services. For more information, see "Configuring a Mutation Processor".

Configuring the Criterion Section

The **criterion** section specifies the fields and values the HTTP message must contain to qualify for transformation. The criteria can include a required value for a single field, a required key-value pair for a field, or a value for a JSON field.

The **criterion** section uses the following syntax:

```
criterion:
   GroupType
   GroupType
   ...
```

Replace *GroupType* with one or more of the following group types:

- **value**: Requires a URI, method, or status code to match a single string value. For example, requiring a message to contain a URI of **https://***hostname***:***httpsPort***/nchf-convergedcharging/v3/chargingdata**. The **value** group type uses this syntax:

  ```
  value:
     ValueObject:
        criteriaType: CriteriaType
        value: Value
  ```

  where:

  – *ValueObject* is one of these: **uri**, **method**, or **statusCode**.

  – *CriteriaType* specifies the criteria type. Supported values are **EQUALS**, **EQUALS_IGNORE_CASE**, **NOT_EQUALS**, **NOT_EQUALS_IGNORE_CASE**, **IS_NOT_NULL**, **IS_NULL**, and **REGEX**.

  – *Value* specifies the required value.

- **keyValue**: Requires a header or query parameter to contain a specified key-value pair, such as a message must include **Accept-Language:en-US** in its header. The **keyValue** group type uses this syntax:

```
keyValue:
   MapName:
   - criteriaType: CriteriaType
     key: KeyName
     value: Value
```

  where:

  – *MapName* is either **headers** or **queryParams**.

  – *CriteriaType* specifies the criteria type. Supported values are **EQUALS**, **EQUALS_IGNORE_CASE**, **NOT_EQUALS**, **NOT_EQUALS_IGNORE_CASE**, **IS_NOT_NULL**, **IS_NULL**, and **REGEX**.

  – *KeyName* specifies the required key.

  – *Value* specifies the required value.

- **json**: Requires a JSON path to match a string value, such as the message body must contain a JSON path of **$.SequenceNumber**. The **json** group type uses this syntax:

```
json:
   body:
   - criteriaType: CriteriaType
     value: MatchValue
```

  where:

  – *CriteriaType* specifies the criteria type. Supported values are **REGEX** and **JSON_PATH**.

  – *Value* specifies the required JSON string value.

Table 5-3 describes each criteria type.

Table 5-3 Criteria Types

Criteria Type	Description	Supported Group Types
EQUALS	The value must match, and the value is case-sensitive.	**value** **keyValue**
EQUALS_IGNORE_CASE	The value must match, and the value is *not* case-sensitive.	**value** **keyValue**
NOT_EQUALS	The value must *not* match, and the value is case-sensitive.	**value** **keyValue**
NOT_EQUALS_IGNORE_CASE	The value must not match, and the value is not case-sensitive.	**value** **keyValue**
IS_NOT_NULL	The value is not null.	**value** **keyValue**
IS_NULL	The value is null.	**value** **keyValue**

Table 5-3 (Cont.) Criteria Types

Criteria Type	Description	Supported Group Types
REGEX	The value must match the specified regular expression.	**value** **keyValue** **json**
JSON_PATH	The JSON string must contain the specified JSON_PATH.	**json**

Configuring the Mutation Section

The **mutation** section specifies how to transform an HTTP message, such as by adding, replacing, or removing fields in the request URL and body. It uses the following syntax:

```
mutation:
   GroupType
   GroupType
   ...
```

Replace *GroupType* with one or more of the following:

- **value**: Modifies a single string value in a URI, method, or status code. For example, by changing the HTTP method to PUT. The **value** group type uses this syntax:

  ```
  value:
    ValueObject:
      mutationType: MutationType
      value: Value
  ```

  where:

  – *ValueObject* is one of these: **uri**, **method**, or **statusCode**.

  – *MutationType* specifies the transformation to make. Supported values are **REPLACE** and **REGEX_REPLACE**.

  – *Value* specifies the new value.

- **keyValue**: Modifies a key-value pair in a header or query parameter, such as by adding **Authority: example.com:1534** to the header. The **keyValue** group type uses this syntax:

  ```
  keyValue:
     MapName:
     - mutationType: MutationType
       key: KeyName
       value: Value
  ```

  where:

  – *MapName* is either **headers** or **queryParams**.

  – *MutationType* specifies the transformation to make. Supported values are **REPLACE**, **ADD**, **COPY**, and **DELETE**.

  – *KeyName* specifies the key on which to act.

  - *Value* specifies a new or modified value.
- **json**: Modifies a specified string in the JSON body, such as by adding or changing a field.

```
json:
   body:
   - mutationType: MutationType
     MutationSpecificParameters
```

  where:

  - *MutationType* specifies the transformation to make. Supported values are **REGEX_REPLACE**, **DELETE**, **JSON_PATCH**, **JSON_PATH**, and **REPLACE_JSON**.
  - *MutationSpecificParameters* specifies more information about the transformation. The fields in this section are dependent on the mutation type.

Table 5-4 describes each mutation type.

Table 5-4 Mutation Types

Mutation Type	Description	Supported Group Types
REPLACE	Replaces an existing key's value with a new value.	**value** **keyValue**
REGEX_REPLACE	Replaces a value based on the specified regular expression.	**value** **json**
ADD	Adds a key-value pair to the header or query parameter.	**keyValue**
COPY	Copies one key to another key with the same value.	**keyValue**
DELETE	Removes a key-value pair or JSON parameter.	**keyValue** **json**
JSON_PATCH	Performs the specified operation on the JSON body. It includes these parameters: • **op**: The operation to perform on the JSON field: **add**, **copy**, **test**, **remove**, **replace**, or **move**. • **path**: Specifies the location in the JSON body to change. • **value**: Specifies the value. For more information, see RFC 6902.	**json**
JSON_PATH	Performs a JSON path mutation. For more information, see JsonPath on the GitHub website: https://github.com/json-path/JsonPath.	**json**
REPLACE_JSON	Replaces the entire JSON body with the new values.	**json**

Example Mutation Rule File

The following shows a sample mutation rule file for updating the request header and JSON body in an incoming HTTP message:

```
ruleName: "Sample Rule"
ruleDescription: "This sample rule updates the request header and adds a
field to the JSON body"
criterion:
  json:
```

```
    body:
      - criteriaType: "JSON_PATH"
        value: "$.invocationSequenceNumber"
mutation:
  keyValue:
    headerMap:
      - mutationType: REPLACE
        key: location
        value: https://new.example.com
      - mutationType: ADD
        key: HTTP2-Settings
        value: token64
  json:
    body:
      - mutationType: JSON_PATCH
        jsonPatch:
          - op: add
            path: /multipleUnitInformation/ratingStamp
            value: "Gold"
```

In this example, Network Bridge would check if an incoming HTTP message includes an **invocationSequenceNumber** field or array in its JSON body. If an HTTP message meets the criteria, Network Bridge would do the following:

- Replace the value of the **location** header field with **https://new.example.com**.
- Add the **HTTP2-Settings** header field with the value set to **token64**.
- In the JSON body, add the **ratingStamp** field set to **Gold** in the **multipleUnitInformation** array. If the body doesn't contain the **multipleUnitInformation** array, it would be added.

Table 5-5 shows a sample HTTP message before and after it has been mutated by Network Bridge using the sample rules. The fields that have been changed or added are shown in bold.

Table 5-5 Sample HTTP Message Mutation

HTTP Message Part	Before Mutation	After Mutation
Header	connection: keep-alive content-length: 154 content-type: application/json **location: https://original.example.com** date: Tue, 12 Jul 2024 08:21:13 +0530	connection: keep-alive content-length: 154 content-type: application/json **location: https://new.example.com** **HTTP2-Settings: token64** date: Tue, 12 Jul 2024 08:21:13 +0530

Table 5-5 (Cont.) Sample HTTP Message Mutation

HTTP Message Part	Before Mutation	After Mutation

JSON Body	<pre>{
	"invocationTimeStamp":
	"2024-7-11T13:00:33.882+05:30",
	"invocationSequenceNumber": 1,
	"multipleUnitInformation": [
	{
	"resultCode": "SUCCESS",
	"ratingGroup": 10,
	"grantedUnit": {},
	"validityTime": 3600
	}
	]
	}</pre>
		<pre>{
		"invocationTimeStamp":
		"2024-7-11T13:00:33.882+05:30",
		"invocationSequenceNumber": 1,
		"multipleUnitInformation": [
		{
		"resultCode": "SUCCESS",
		"ratingGroup": 10,
		"grantedUnit": {},
		"validityTime": 3600,
		"ratingStamp": "Gold"
		}
		]
		}</pre>

Defining Business Rules in DRL Files

Creating DRL files allows you to customize how Network Bridge transforms 5G message payloads.

To define business rules for 5G payload conversion:

1. Create a DRL file that applies rules on HTTP messages.

   For information about creating DRL files, see the Drools documentation at: https://docs.drools.org/8.44.0.Final/drools-docs/drools/introduction/index.html.

2. In your file, add rules for converting HTTP message payloads based on your business logic.

3. Create a ConfigMap from your DRL file by running this command:

   ```
   kubectl create configmap ConfigMapName --from-file=Path
   ```

   where *ConfigMapName* is your desired name for the ConfigMap, and *FilePath* is the directory that contains the DRL file.

4. In your **override-values.yaml** file, set the **network-bridge-restproxy.rest-proxy.mediation.kieRulesConfigMapOverride** key to *ConfigMapName*.

   This configures Network Bridge to use your DRL file for all **KIE_PROCESSOR** type processors.

Sample N40 Proxy Configuration

The following shows sample **override-values.yaml** entries for configuring Network Bridge as an N40 proxy, with tracing, autoscaling, and monitoring through Prometheus and Grafana all enabled.

> **Note:**
>
> The **&keyName** and ***keyName** values are Helm references. If a key in the **values.yaml** file includes one of these references, you must also include it in your **override-values.yaml** file.

```
imageRepository: &imageRepository repository.example.com:7840
imagePullSecrets: &imagePullSecrets regcred
imagePullPolicy: &imagePullPolicy IfNotPresent

partOfLabel: &partOfLabel network-bridge-restproxy
nodeSelector: &nodeSelector
#Sample:
#   key: value

loggingPattern: &loggingPattern "%d{ISO8601_OFFSET_DATE_TIME_HHCMM} | %5p |
%X{traceId} | %-20.20thread | %-25.25logger{25} | %m%n"

tracingEnabled: &tracingEnabled true
tracingHost: &tracingHost jaeger-all-in-one.jaeger.svc.cluster.local
tracingPort: &tracingPort 14268
serviceMonitorEnabled: &serviceMonitorEnabled true

grafanaDashboardsEnabled: &grafanaDashboardsEnabled true
grafanaNamespace: &grafanaNamespace kube-logging

grafanaDashboards:
  enabled: *grafanaDashboardsEnabled
  grafanaNamespace: *grafanaNamespace
  labels:
    grafana_dashboard: "1"
    release: prometheus
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"
serviceMonitor:
  enabled: *serviceMonitorEnabled
  namespace: SmNameSpace

network-bridge-restproxy:
  enabled: false
  restProxyImage: &restProxyImage "ccs/network-bridge/mediation:2.0.0"
  # Network Bridge Top Level
  name: network-bridge
  fullname: "network-bridge-restproxy"

  rest-proxy:
    enabled: true
    imageRepository: *imageRepository
    imagePullSecrets: *imagePullSecrets
    imagePullPolicy: *imagePullPolicy
    partOfLabel: *partOfLabel
    mediation:
      name: rest-proxy
```

```
      fullname: "rest-proxy"
      additionalLabels: { }
      annotations: { }
      nodeSelector: *nodeSelector
      client:
        connect-timeout-millis: 2000
        read-timeout-millis: 2000
        follow-redirects: true
        max-redirects: 5
        additionalAttributes: { }
      kieRulesConfigMapOverride: MyKieRules
      mutatorRulesConfigMapOverride: MyMutationRules
      restServices:
        - name: "all_sbi"
          config:
            path: "/{:npcf-smpolicycontrol\/v1\/sm-policies($|\/[^/]+\/
(update|delete)($|\/$))|nchf-convergedcharging\/v3\/chargingdata($|\/[^/]+\/
(update|release)($|\/$))|nchf-convergedcharging\/v3\/notify($|\?
notifyUri=.*)}"
          processors:
            - name: "kie_processor"
              type: KIE_PROCESSOR
              config:
                rulesPath:"/app/config/rules"
            - name: mutation-request
              type: MUTATOR_PROCESSOR
              config:
                rulesPath: "src/test/resources/mutation/rules"
                ruleNames:
                  - addChargingInfo
                  - addHeaders
            - name: sample-rest-processor
              type: REST_PROCESSOR
              config:
                uri: "http://sample.example.com"
                destUriHeader: dest_uri
              faultTolerance:
                retry:
                delay: 5
                calls: 3
                delayFactor: 1.0
            - name: mutation-response
              type: MUTATOR_PROCESSOR
              config:
                rulesPath: "src/test/resources/mutation/rules"
                ruleNames:
                  - reWriteResponse
      grpcServices: [ ]
      container:
        image: *restProxyImage
      terminationGracePeriodSeconds: 60
      resources:
        memoryRequest: "256Mi"
        cpuRequest: "250m"
        memoryLimit: "2048Mi"
        cpuLimit: "500m"
```

```
      replicas: 1
      jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -
XX:InitialRAMPercentage=80 -XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -
Dlog4j.configurationFile=config/log4j2.yaml"
      restartCount: 0
      hpa:
        enabled: true
        minReplicas: 1
        maxReplicas: 8
        metrics:
          cpuAverageUtilization: 65
        scaleDown:
          selectPolicy: Max
          stabilizationWindowSeconds: 300
          periodSeconds: 180
        scaleUp:
          selectPolicy: Max
          stabilizationWindowSeconds: 5
          periodSeconds: 20
      service:
        type: ClusterIP
        additionalLabels: { }
        additionalAnnotations: { }
      adminService:
        type: ClusterIP
        additionalLabels: { }
        additionalAnnotations: { }
      logging:
        format:
          type: TEXT
          pattern: *loggingPattern
        rootLevel: INFO
        packageLogging:
          - name: com.oracle.cagbu
            level: INFO
          - name: io.helidon
            level: INFO
          - name: org.jboss
            level: INFO
          - name: io.jaegertracing.internal.reporters
            level: WARN
      tracing:
        enabled: *tracingEnabled
        service: mediation
        maxQueueSize: 42
        flushIntervalMs: 10001
        host: *tracingHost
        port: *tracingPort
        samplerType: ratelimiting
        samplerParam: 1
        samplerManager:
        logSpans: true
        webServer:
          spans:
            httpRequest:
              contentWrite: false
```

```
          contentRead: false

grafanaDashboards:
  enabled: *grafanaDashboardsEnabled
  grafanaNamespace: *grafanaNamespace
  labels:
    grafana_dashboard: '1'
    release: prometheus
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"

serviceMonitor:
  enabled: *serviceMonitorEnabled
  namespace:

ingress:
  ingressClassName:
```

6
Deploying Network Bridge

You can deploy Oracle Communications Network Bridge on a cloud native environment by running the Helm install command.

Topics in this document:

- Deploying Network Bridge Cloud Native

Deploying Network Bridge Cloud Native

To deploy Network Bridge on your cloud native environment:

1. Create a namespace for the Network Bridge Helm chart:

   kubectl create namespace *NBNameSpace*

   where *NBNameSpace* is the namespace in which to create Kubernetes objects for the Network Bridge Helm chart.

2. Validate the content of your Helm chart by running the **Helm lint** command:

   helm lint --strict oc-ccs-helm-chart-*version* **--values oc-ccs-helm-chart-***version***/values.yaml --values** *OverrideValuesFile*

   You'll see this if the command completes successfully:

   ```
   1 chart(s) linted, no failures
   ```

3. Validate the values in your **override-values.yaml** file by using the **helm template** command:

   helm template *NBReleaseName* **oc-ccs-helm-chart-***version*

   where *NBReleaseName* is the release name for **oc-ccs-helm-chart-***version* and is used to track this installation instance.

4. Deploy Network Bridge by running this command:

   helm install *NBReleaseName* **oc-ccs-helm-chart-***version* **--namespace** *NBNameSpace* **--values override-values.yaml**

5. Verify that the pods were deployed successfully by running this command:

   kubectl -n *NbNameSpace* **get pods**

   You should see something similar to this:

   ```
   NAME                                    READY   STATUS      RESTARTS   AGE
   diameter-adapter-d2h-6f79d95887-lp7qs   1/1     Running     0
   6d17h
   diameter-adapter-h2d-5496bf8d94-vjgn7   1/1     Running     0
   ```

```
6d17h
diameter-proxy-d5ccf6dbd-l968b          1/1     Running     0
6d17h
diameter-proxy-db-job-22bsg             0/1     Completed   0          7d5h
egress-7b974f4488-wx9qz                 1/1     Running     0          7d4h
mediation-75f9dcd99d-vrrwg              1/1     Running     0          7d4h
oc-ccs-ndb-mgmd-0                       1/1     Running     0          7d5h
oc-ccs-ndb-mgmd-1                       1/1     Running     0          7d5h
oc-ccs-ndb-mysqld-0                     1/1     Running     0          7d5h
oc-ccs-ndb-mysqld-1                     1/1     Running     0          7d5h
oc-ccs-ndb-ndbmtd-0                     1/1     Running     0          7d5h
oc-ccs-ndb-ndbmtd-1                     1/1     Running     0          7d5h
```

Part II
Administering Network Bridge

This part describes how to perform administration tasks on your cloud native deployment of Oracle Communications Network Bridge. It contains the following chapters:

- Managing Network Bridge Pods
- Tracing the Flow of API Calls
- Monitoring Network Bridge Processes
- Using Network Bridge Logging

7

Managing Network Bridge Pods

You can manage the pods in your Oracle Communications Network Bridge cloud native deployment by setting up autoscaling.

Topics in this document:

Setting up Autoscaling of Network Bridge Pods

You can use the Kubernetes Horizontal Pod Autoscaler to automatically scale up or scale down the number of pod replicas based on a pod's CPU or memory utilization. In Network Bridge cloud native deployments, the Horizontal Pod Autoscaler can monitor and scale these pods:

- mediation
- egress
- diameter-adapter-h2d
- diameter-adapter-d2h
- diameter-proxy

Changing the number of replicas in a Network Bridge autoscalable ReplicaSet rebalances the in-memory cache distribution across the replicas. This rebalancing activity consumes incremental CPU and memory resources and can take multiple seconds to complete. Therefore, your autoscaling design should strike a balance between optimizing infrastructure resource usage and minimizing changes to the number of replicas in a ReplicaSet due to autoscaling.

To set up and enable autoscaling for Network Bridge pods:

1. Ensure that your Network Bridge cluster is set up and the system is in the **UsageProcessing** state.
2. Open your **override-values.yaml** file.
3. Enable the Horizontal Pod Autoscaler in all of the pods by setting these keys to **true**:
    - For the mediation pod: **network-bridge.mediation.mediation.hpa.enabled**
    - For the mediation pod in a REST proxy: **network-bridge-restproxy.rest-proxy.mediation.hpa.enabled**
    - For the egress pod: **network-bridge.egress.mediation.hpa.enabled**
    - For the diameter-adapter-h2d pod: **network-bridge.diameter-adapter-h2d.protocolTransform.hpa.enabled**
    - For the diameter-adapter-d2h pod: **network-bridge.diameter-adapter-d2h.protocolTransform.hpa.enabled**
    - For the diameter-proxy pod: **network-bridge.diameter-proxy.diameterProxy.hpaEnabled**

4. For each pod, specify the minimum and maximum amount of memory and CPU that can be used.

   Set these keys under the **resources** section of each pod. For example, under **diameter-adapter-h2d.protocolTransform.resources**.

   - **memoryRequest**: Set this to the minimum amount of memory required for a Kubernetes node to deploy a pod.

     If the minimum amount is not available, the pod's status is set to **Pending**.

   - **cpuRequest**: Set this to the minimum CPU amount, in millicores, that must be available in a Kubernetes node to deploy a pod. For example, enter 1000m for 1 CPU core.

     If the minimum CPU amount is not available, the pod's status is set to **Pending**.

   - **memoryLimit**: Set this to the maximum amount of memory that a pod can utilize.
   - **cpuLimit**: Set this to the maximum amount of CPU that a pod can utilize.

5. For each pod, specify the minimum and maximum number of pod replicas that can be deployed.

   Set these keys under the **hpa** section of each pod. For example, under the **network-bridge.egress.mediation.hpa** section.

   - **minReplicas**: Set this to the minimum number of pod replicas to deploy when scale down is triggered.

     If a pod's average utilization goes below **cpuAverageUtilization**, the Horizontal pod Autoscaler decreases the number of pod replicas down to this minimum count.

   - **maxReplicas**: Set this to the maximum number of pod replicas to deploy when scale up is triggered.

     If a pod's average utilization goes above **cpuAverageUtilization**, the Horizontal pod Autoscaler increases the number of pod replicas up to this maximum count.

   - **metrics.cpuAverageUtilization**: Set this as a target or threshold for average CPU usage across all of the pod's replicas with the same entry point. For example, if a cluster has three mediation pod replicas, the average will be the sum of CPU usage divided by three. The default is 65%.

     The autoscaler increases or decreases the number of pod replicas to maintain the average CPU utilization you specified across all pods.

6. For each pod, specify the rules for scaling down pods.

   Set these keys under the **hpa.scaleDown** section of each pod. For example, under the **network-bridge.egress.mediation.hpa.scaleDown** section.

   - **selectPolicy**: Specifies **Min**, **Max**, or **Disabled**.
     - **Min** selects the policy with the smallest change in the replica count.
     - **Max** selects the policy with the largest change in the replica count.
     - **Disabled** prevents autoscaling in the scale down direction.
   - **stabilizationWindowSeconds**: Specifies the duration, in seconds, of the stabilization window when scaling down pods.
   - **periodSeconds**: Specifies the number of seconds for which metrics should be collected before scaling.

7. For each pod, specify the rules for scaling up pods.

Set these keys under the **hpa.scaleUp** section of each pod. For example, under the **network-bridge.egress.mediation.hpa.scaleUp** section.

- **selectPolicy**: Specifies **Min**, **Max**, or **Disabled**.
  - **Min** selects the policy with the smallest change in the replica count.
  - **Max** selects the policy with the largest change in the replica count.
  - **Disabled** prevents autoscaling in the scale down direction.
- **stabilizationWindowSeconds**: Specifies the duration, in seconds, of the stabilization window when scaling up pods.
- **periodSeconds**: Specifies the number of seconds for which metrics should be collected before scaling.

8. To lower the heap memory used by the pods, set the appropriate JVM parameters in the **jvmOpts** key.

   Memory-based scale down occurs only if the amount of pod memory decreases. You can decrease pod memory by using JVM garbage collection (GC).

9. Save and close your **override-values.yaml** file.

10. Do one of the following:

    - Deploy Network Bridge (if you have not already deployed Network Bridge). See "Deploying Network Bridge".
    - If you have already deployed Network Bridge, update your Network Bridge release:

      ```
      helm upgrade NBReleaseName oc-ccs-helm-chart-version --values override-values.yaml -n NBNameSpace
      ```

      where:

      - *NBReleaseName* is the release name for the Network Bridge deployment.
      - *version* is the Network Bridge release number, such as **2.0.0**.
      - *NBNameSpace* is the namespace in which to create Kubernetes objects for the Network Bridge Helm chart.

8

Tracing the Flow of API Calls

You can trace the flow of API calls made to Network Bridge through Jaeger in your Oracle Communications Network Bridge cloud native system.

Topics in this document:

About Tracing

You can trace the flow of messages through Network Bridge by using the Jaeger tracer tool integrated with the Helidon framework. You use this tool to understand any request failures and to troubleshoot performance issues in Network Bridge.

Helidon is a collection of Java libraries used by Network Bridge, and Jaeger is an open-source tracing system that is used with Helidon. For more information about Helidon and Jaeger, see:

- "MP - Jaeger Tracing" in the Helidon documentation
- "Introduction" in the Jaeger documentation

In a Network Bridge cloud native deployment, you can trace these components:

- Mediation component
- Egress component
- HTTP to Diameter Adapter component
- Diameter to HTTP Adapter component
- Diameter Proxy component

To set up tracing in Network Bridge cloud native:

1. Install Jaeger Operator. See the Jaeger Operator for Kubernetes documentation: https://www.jaegertracing.io/docs/1.41/operator/.
2. Enable Jaeger tracing in Network Bridge cloud native. See "Enabling Tracing in Network Bridge".

Afterward, you can use the Jaeger UI to start visualizing and analyzing the trace data.

Enabling Tracing in Network Bridge

By default, tracing is disabled in Network Bridge cloud native.

To enable tracing with Jaeger:

1. Create an **override-values.yaml** file for **oc-ccs-helm-chart-***version*.
2. Enable tracing across all Network Bridge components by setting these keys:
   - **tracingEnabled**: Set this to **true**.

- **tracingHost**: Set to this to the host name of the server on which tracing is running. For example: **opentelemetry.telemetry.svc.cluster.local**.
- **tracingPort**: Set this to the trace server port.

3. (Optional) Customize how Network Bridge samples messages. Set these keys under the **tracing** section of each Network Bridge component:
   - **service**: Set this to the name of the service.
   - **maxQueueSize**: Set this to the maximum queue size for Jaeger reporters. The default is **42** spans.
   - **flushIntervalMs**: Specify the amount of time, in milliseconds, that events are held in the queue before sending a batch. The default is **10002** (about 10 seconds).
   - **samplerType**: Specify the sampling strategy to use: **const** or **ratio**. The default is **ratio**.
     - **const**: The sampler performs the same action, defined in the **samplerParam** key, on all traces.

       Set the **samplerParam** key to **1** to sample all traces or to **0** to sample none of the traces.
     - **ratio**: The sampler ensures that a specified percentage of traces are sampled.

       Set the **samplerParam** key to the percentage of traces to sample. For example, **0.5** specifies to sample 50% of all traces, and **1.0** specifies to sample all traces.
   - **logSpans**: Specify whether to log spans for diagnostic purposes. A span is a logical unit of work that has an operation name, a start time, and a duration. The default is **true**.
4. Save and close your **override-values.yaml** file.
5. Deploy or redeploy the Network Bridge Helm release by running the **helm install** command:

   helm install *NbReleaseName* **oc-ccs-helm-chart-***version* **--values override-values.yaml -n** *NbNameSpace*

   where:
   - *NbReleaseName* is the release name for the Network Bridge Helm chart and is used to track this installation instance.
   - *NbNameSpace* is the namespace in which to create Kubernetes objects for the Network Bridge Helm chart.

The following sample **override-values.yaml** file configures tracing for the Mediation component. It specifies to create a queue the size of 42 spans, to hold events in the queue for 10 seconds, to sample all traces, and to log spans.

```
tracing:
  service: mediation
  maxQueueSize: 42
  flushIntervalMs: 1000
  samplerType: const
  samplerParam: 1
  samplerManager:
  logSpans: true
```

9

Monitoring Network Bridge Processes

You can monitor system processes, such as memory and thread usage, in your Oracle Communications Network Bridge components in a cloud native environment.

Topics in this document:

About Monitoring Network Bridge Cloud Native

You can set up monitoring of your Network Bridge cloud native components using a Kubernetes Service Monitor, Prometheus Operator, and Grafana Dashboards. Service Monitor exposes JVM and application metric data through a single endpoint in an OpenMetrics/ Prometheus exposition format. Prometheus then scrapes the metrics and stores them for analysis and monitoring through the Grafana Dashboards.

Network Bridge cloud native exposes metric data for the following components:

- Mediation component
- Egress component
- HTTP to Diameter Adapter component
- Diameter to HTTP Adapter component
- Diameter Proxy component

Setting Up Monitoring of Network Bridge Components

Setting up monitoring of Network Bridge cloud native components involves the following high-level tasks:

1. Deploying the following prerequisite software on your Network Bridge cloud native environment:
   - Deploying Prometheus Operator on your Network Bridge cloud native environment. See "Installing Prometheus Operator".
   - Deploying Grafana on your Network Bridge cloud native environment. See "Installing Grafana".

   For the list of compatible versions, see "Network Bridge Cloud Native Software Compatibility" in *CCS Compatibility Matrix*.
2. Enabling the Network Bridge Service Monitor and Grafana Dashboards.

Enabling the Network Bridge Service Monitor

By default, the Network Bridge Service Monitor and Grafana Dashboards are disabled. You must enable both if you want to monitor your system's processes using Prometheus and Grafana.

To enable the Service Monitor and Grafana Dashboards:

1. Create an **override-values.yaml** file for the **oc-ccs-helm-chart-***version* Helm chart.
2. Enable the Network Bridge Service Monitor and Grafana Dashboards by setting these keys:
   - **serviceMonitorEnabled**: Set this to **true**.
   - **grafanaDashboardsEnabled**: Set this to **true**.
   - **grafanaNamespace**: Set this to the namespace in which to deploy the Grafana Dashboards.
   - **serviceMonitor.namespace**: Set this to the namespace in which to deploy the Service Monitor CRD.
3. Optionally, modify the default settings for Grafana:
   - **granfanaDashboards.labels.grafana_dashboard**: Specify the labels to add to the Grafana CRDs. This helps Grafana discover the dashboards.
   - **granfanaDashboards.labels.release**: Set this to the release name in which Grafana is deployed.
   - **granfanaDashboards.annotations.k8s-sidecar-target-directory**: Set this to the directory for the Grafana sidecar.
4. Save and close your **override-values.yaml** file.
5. Run the **helm upgrade** command to update your Network Bridge Helm release:

   ```
   helm upgrade NbReleaseName oc-ccs-helm-chart-version --namespace
   NbNameSpace --values override-values.yaml
   ```

   where *NbReleaseName* is the release name for Network Bridge, and *NbNameSpace* is the namespace in which to create Kubernetes objects for the Network Bridge Helm chart.

Network Bridge Cloud Native Metrics

Network Bridge cloud native collects metrics in the following groups to produce data for monitoring your components:

- Mediation and REST Proxy Metrics
- Diameter Adapter Metrics
- Diameter Proxy Metrics

Mediation and REST Proxy Metrics

The Mediation and REST Proxy group contains standard metrics about the central processing unit (CPU) and memory utilization of JVMs, which are members of the Network Bridge grid. It also contains metrics for tracking the processing performance of requests to and responses

from the Mediation, Egress, and REST Proxy components. Table 9-1 lists the metrics in this group.

Table 9-1 Mediation and REST Proxy Metrics

Metric Name	Type	Description
jvm_buffer_count_buffers	Gauge	Contains the estimated number buffers in the JVM memory pool.
jvm_buffer_memory_used_bytes	Gauge	Contains an estimated amount of memory the JVM is using for the buffer pool.
jvm_buffer_total_capacity_bytes	Gauge	Contains the estimated total capacity of the buffers in this pool.
jvm_gc_live_data_size_bytes	Gauge	Contains the size, in bytes, of the long-lived heap memory pool after reclamation.
jvm_gc_max_data_size_bytes	Gauge	Contains the maximum size of the long-lived heap memory pool.
jvm_gc_memory_allocated_bytes_total	Counter	Tracks the total size of increases to the young heap memory after one GC to before the next one.
jvm_gc_memory_promoted_bytes_total	Counter	Tracks the total size of incremental increases to the old generation memory pool from before GC to after GC.
jvm_gc_pause_seconds	Summary	Contains information about the time spent in GC pause.
jvm_gc_pause_seconds_max	Gauge	Contains the maximum amount of time spent in GC pause.
jvm_memory_committed_bytes	Gauge	Contains the amount of memory, in bytes, that is committed for the JVM to use.
jvm_memory_max_bytes	Gauge	Contains the maximum amount of memory, in bytes, that can be used for memory management.
jvm_memory_used_bytes	Gauge	Contains the amount of memory used, in bytes.
jvm_threads_daemon_threads	Gauge	Contains the current number of live daemon threads.
jvm_threads_live_threads	Gauge	Contains the current number of live threads, including both daemon and non-daemon threads.
jvm_threads_peak_threads	Gauge	Contains the peak live thread count since the JVM started or the peak was reset.
jvm_threads_states_threads	Gauge	Contains the current number of threads in the NEW state.
log4j2_events_total	Counter	Tracks the total number of fatal-level log events.
nb_service_mutation_execution_seconds	Histogram	Contains the time taken to perform a mutation.
nb_service_mutation_execution_seconds_max	Gauge	Contains the time taken to perform a mutation.
nb_service_processor_circuit_breaker_state	Gauge	Contains the current state of the circuit breaker: CLOSED (**0.0**), HALF_OPEN (**1.0**), or OPEN (**2.0**).

Table 9-1 (Cont.) Mediation and REST Proxy Metrics

Metric Name	Type	Description
ccs_service_processor_failed_total	Counter	Tracks the total number of failed events.
nb_service_processor_retries_total	Counter	Tracks the total number of retries.
ccs_service_processor_seconds	Histogram	Contains information about the time taken to process the request.
ccs_service_processor_seconds_max	Gauge	Contains the time taken to process the request.
nb_service_request_failed_total	Counter	Tracks the total number of failed mediation requests.
nb_service_request_seconds	Histogram	Provides information about the total amount of time to process the request.
nb_service_request_seconds_max	Gauge	Contains the total amount of time to process the request.
nb_service_rule_matches_total	Counter	Tracks the number of times a rule has been matched.
process_cpu_usage	Gauge	Contains the recent CPU usage for the JVM process.
process_files_max_files	Gauge	Contains the maximum number of file descriptors.
process_files_open_files	Gauge	Contains the number of open file descriptors.
process_start_time_seconds	Gauge	Contains the start time of the process since the UNIX epoch time.
process_uptime_seconds	Gauge	Contains the JVM's total amount of uptime.
system_cpu_count	Gauge	Contains the number of processors available to the JVM.
system_cpu_usage	Gauge	Contains the recent CPU usage for the entire system.
system_load_average_1m	Gauge	Contains the total number of runnable entities queued to available processors, and the number of runnable entities running on the available processors averaged over a period of time.

Diameter Adapter Metrics

The Diameter Adapter Metrics group contains standard metrics about the CPU and memory utilization of JVMs and the processing performance of requests to and responses from the HTTP-to-Diameter Adapter and Diameter-to-HTTP Adapter components. Table 9-2 lists the metrics in this group.

Table 9-2 Diameter Adapter Metrics

Metric	Type	Description
jvm_buffer_count_buffers	Gauge	Contains the estimated number of buffers in the JMV memory pool.

Table 9-2 (Cont.) Diameter Adapter Metrics

Metric	Type	Description
jvm_buffer_memory_used_bytes	Gauge	Contains an estimated amount of memory the JVM is using for the buffer pool.
jvm_buffer_total_capacity_bytes	Gauge	Contains the estimated total capacity of the buffers in this pool.
jvm_gc_live_data_size_bytes	Gauge	Contains the size, in bytes, of the long-lived heap memory pool after reclamation.
jvm_gc_max_data_size_bytes	Gauge	Contains the maximum size of the long-lived heap memory pool.
jvm_gc_memory_allocated_bytes_total	Counter	Tracks the total size of increases to the young heap memory after one GC to before the next one.
jvm_gc_memory_promoted_bytes_total	Counter	Tracks the total size of incremental increases to the old generation memory pool from before GC to after GC.
jvm_gc_pause_seconds	Summary	Contains information about the time spent in GC pause.
jvm_gc_pause_seconds_max	Gauge	Contains the maximum amount of time spent in GC pause.
jvm_memory_committed_bytes	Gauge	Contains the amount of memory, in bytes, that is committed for the JVM to use.
jvm_memory_max_bytes	Gauge	Contains the maximum amount of memory, in bytes, that can be used for memory management.
jvm_memory_used_bytes	Gauge	Contains the amount of memory used, in bytes.
jvm_threads_daemon_threads	Gauge	Contains the current number of live daemon threads.
jvm_threads_live_threads	Gauge	Contains the current number of live threads, including both daemon and non-daemon threads.
jvm_threads_peak_threads	Gauge	Contains the peak live thread count since the JVM started or the peak was reset.
jvm_threads_states_threads	Gauge	Contains the current number of threads in the NEW state.
log4j2_events_total	Counter	Tracks the total number of fatal-level log events.
nb_service_processor_failed_total	Counter	Tracks the total number of failed events.
nb_service_processor_retries_total	Counter	Tracks the total number of processor retries.
nb_service_processor_seconds	Histogram	Provides information about the time taken to process the request.
nb_service_processor_seconds_max	Gauge	Contains the time taken to process the request.
nb_service_request_failed_total	Counter	Tracks the total number of failed events.
nb_service_request_seconds	Histogram	Tracks the delta between the Usage UsageDate and the time the service received the event.

Table 9-2 (Cont.) Diameter Adapter Metrics

Metric	Type	Description
nb_service_request_seconds_max	Gauge	Tracks the delta between the Usage UsageDate and the time the service received the event.
process_cpu_usage	Gauge	Contains the recent CPU usage for the JVM process.
process_files_max_files	Gauge	Contains the maximum number of file descriptors.
process_files_open_files	Gauge	Contains the number of open file descriptors.
process_start_time_seconds	Gauge	Contains the start time of the process since the UNIX epoch time.
process_uptime_seconds	Gauge	Contains the JVM's total amount of uptime.
system_cpu_count	Gauge	Contains the number of processors available to the JVM.
system_cpu_usage	Gauge	Contains the recent CPU usage for the entire system.
system_load_average_1m	Gauge	Contains the total number of runnable entities queued to available processors, and the number of runnable entities running on the available processors averaged over a period of time.

Diameter Proxy Metrics

The Diameter Proxy group contains standard metrics about the CPU and memory utilization of JVMs and the processing performance of requests to and responses from the Diameter Proxy component. Table 9-3 lists the metrics in this group.

Table 9-3 Diameter Proxy Metrics

Metric	Type	Description
jvm_buffer_count_buffers	Gauge	Contains the estimated number of buffers in the JVM memory pool.
jvm_buffer_memory_used_bytes	Gauge	Contains an estimated amount of memory the JVM is using for the buffer pool.
jvm_buffer_total_capacity_bytes	Gauge	Contains the estimated total capacity of the buffers in this pool.
jvm_gc_live_data_size_bytes	Gauge	Contains the size, in bytes, of the long-lived heap memory pool after reclamation.
jvm_gc_max_data_size_bytes	Gauge	Contains the maximum size of the long-lived heap memory pool.
jvm_gc_memory_allocated_bytes_total	Counter	Tracks the total size of incremental increases to the young heap memory after one GC to before the next one.
jvm_gc_memory_promoted_bytes_total	Counter	Tracks the total size of incremental increases to the old generation memory pool from before GC to after GC.

Table 9-3 (Cont.) Diameter Proxy Metrics

Metric	Type	Description
jvm_gc_pause_seconds	Summary	Contains information about the time spent in GC pause.
jvm_gc_pause_seconds_max	Gauge	Contains the maximum amount of time spent in GC pause.
jvm_memory_committed_bytes	Gauge	Contains the amount of memory, in bytes, that is committed for the JVM to use.
jvm_memory_max_bytes	Gauge	Contains the maximum amount of memory, in bytes, that can be used for memory management.
jvm_memory_used_bytes	Gauge	Contains the amount of memory used, in bytes.
jvm_threads_daemon_threads	Gauge	Contains the current number of live daemon threads.
jvm_threads_live_threads	Gauge	Contains the current number of live threads, including both daemon and non-daemon threads.
jvm_threads_peak_threads	Gauge	Contains the peak live thread count since the JVM started or the peak was reset.
jvm_threads_states_threads	Gauge	Contains the current number of threads in the NEW state.
log4j2_events_total	Counter	Tracks the total number of fatal-level log events.
nb_service_message_seconds	Histogram	Timer information about for recording messages to and from the Diameter Proxy.
nb_service_message_seconds_max	Gauge	Timer for recording messages to and from the Diameter Proxy.
nb_service_open_connections	Gauge	Tracks the number of open Diameter connections.
nb_service_peer_event_total	Counter	Tracks the total number of Diameter Peer events.
nb_service_processor_failed_total	Counter	Tracks the total number of failed events.
nb_service_processor_retries_total	Counter	Tracks the total number of processor retries.
nb_service_processor_seconds	Histogram	Contains the time taken to process the request.
nb_service_processor_seconds_max	Gauge	Contains the time taken to process the request.
nb_service_request_failed_total	Counter	Tracks the total number of failed Diameter Proxy requests.
nb_service_request_seconds	Histogram	Contains information about the total amount of time to process a request.
nb_service_request_seconds_max	Gauge	Contains the total amount of time to process the request.
process_cpu_usage	Gauge	Contains the recent CPU usage for the JVM process.

Table 9-3 (Cont.) Diameter Proxy Metrics

Metric	Type	Description
process_files_max_files	Gauge	Contains the maximum number of file descriptors.
process_files_open_files	Gauge	Contains the number of open file descriptors.
process_start_time_seconds	Gauge	Contains the start time of the process since the UNIX epoch time.
process_uptime_seconds	Gauge	Contains the JVM's total amount of uptime.
system_cpu_count	Gauge	Contains the number of processors available to the JVM.
system_cpu_usage	Gauge	Contains the recent CPU usage for the entire system.
system_load_average_1m	Gauge	Contains the total number of runnable entities queued to available processors, and the number of runnable entities running on the available processors averaged over a period of time.

10

Using Network Bridge Logging

You can review log files to troubleshoot errors and monitor system activity in your Oracle Communications Network Bridge cloud native system.

Topics in this document:

About Logging

Network Bridge cloud native uses the Apache Log4j2 Java logging utility to log information and errors about the following:

- Start up and shut down activity.
- Interaction with external applications at integration points, including interactions with the ingress controller, 5G Network Function (NF), and 4G Network Element (NE).
- Network Manager components: Mediation, HTTP to Diameter Adapter, Diameter Proxy, Diameter to HTTP Adapter, and Egress.

For general information about Java logging, see *Java Platform, Standard Edition Core Libraries*. For information about Log4j2, see: https://logging.apache.org/log4j/2.x/manual/index.html.

When you deploy Network Bridge cloud native, logging is automatically set up and running in your Network Bridge cloud native environment with default settings.

- To access the log files, see "Accessing the Network Bridge Logs".
- To change the default Network Bridge logging levels, see "Changing the Log Levels".

Accessing the Network Bridge Logs

You access the logs by using the **kubectl** command in the Network Bridge namespace.

To access the logs:

1. Retrieve the names of the Network Bridge pods by entering this command:

   ```
   kubectl -n NBNameSpace get pods
   ```

   where *NBNameSpace* is the namespace in which Kubernetes objects for the Network Bridge Helm chart reside.

   The following is an example of the command's output, with the pod names in bold:

   ```
   NAME                                        READY   STATUS      RESTARTS   AGE
   diameter-adapter-d2h-6f79d95887-lp7qs       1/1     Running     0
   6d17h
   ```

```
diameter-adapter-h2d-5496bf8d94-vjgn7   1/1     Running     0
6d17h
diameter-proxy-d5ccf6dbd-l968b          1/1     Running     0
6d17h
diameter-proxy-db-job-22bsg             0/1     Completed   0          7d5h
egress-7b974f4488-wx9qz                 1/1     Running     0          7d4h
mediation-75f9dcd99d-vrrwg              1/1     Running     0          7d4h
oc-ccs-ndb-mgmd-0                       1/1     Running     0          7d5h
oc-ccs-ndb-mgmd-1                       1/1     Running     0          7d5h
oc-ccs-ndb-mysqld-0                     1/1     Running     0          7d5h
oc-ccs-ndb-mysqld-1                     1/1     Running     0          7d5h
oc-ccs-ndb-ndbmtd-0                     1/1     Running     0          7d5h
oc-ccs-ndb-ndbmtd-1                     1/1     Running     0          7d5h
```

2. Access the logs for a pod by entering this command:

   kubectl -n *NBNameSpace* **logs** *PodName*

   For example, to access the mediation logs, you would enter:

   kubectl -n *NBNameSpace* logs mediation-75f9dcd99d-vrrwg

The following is an example of the logs for the Mediation component:

```
2023-11-30T19:53:49,745+00:00 | INFO |  | main       | .helidon.common.LogConfig |
Logging at initialization configured using defaults
2023-11-30T19:53:50,626+00:00 | INFO |  | main       | iation.factory.KieFactory |
Loading rules file from '/app/config/rules' with glob pattern 'glob:/app/config/rules/
*.drl'
2023-11-30T19:53:51,030+00:00 | INFO |  | main       | iation.factory.KieFactory |
Adding rules file: '/app/config/rules/pt_n40_to_gy.drl'
```

> **Note:**
>
> This task shows how to access a single log at a time. To tail logs from multiple pods, Oracle recommends using the Kubernetes Stern tool.

Changing the Log Levels

You can change each Network Bridge component's logging at the root level and the package level to one of the following:

- **TRACE**: This log level provides verbose information, including each row loaded into the database.
- **DEBUG**: This log level provides information about the steps for each loading function.
- **WARN**: This log level provides non-critical warnings.
- **INFO**: This log level provides a one-line summary of each file processed.
- **ERROR**: This log level provides only error information.

To change the log levels for a Network Bridge component:

1. Create an **override-values.yaml** file for your Network Bridge Helm chart.
2. To set a component's root-level logging, set the **logging.rootLevel** key to the desired logging level.

3. To set a component's package-level logging, set the following keys:
   - **logging.packageLogging.name**: The name of the logging package, such as io.helicon or org.jboss.
   - **logging.packageLogging.level**: The logging level, such as **TRACE**, **DEBUG**, **WARN**, **INFO**, or **ERROR**.
4. Save and close your **override-values.yaml** file.
5. Update your Network Bridge Helm release:

   ```
   helm upgrade NBReleaseName oc-ccs-helm-chart-version --values override-values.yaml -n NBNameSpace
   ```

The following shows sample logging settings for the HTTP to Diameter Adapter component:

```
network-bridge:
   diameter-adapter-h2d:
      protocolTransform:
         logging:
            format:
               type: TEXT
               pattern: *loggingPattern
            rootLevel: DEBUG
            packageLogging:
               - name: com.oracle
                 level: DEBUG
               - name: io.helidon
                 level: DEBUG
               - name: io.jaegertracing.internal.reporters
                 level: DEBUG
```