Oracle® Communications Billing and Revenue Management

ECE Composable Services Installation and Administration Guide

Release 15.2
F61229-05
June 2026

ORACLE

Oracle Communications Billing and Revenue Management ECE Composable Services Installation and Administration Guide, Release 15.2

F61229-05

Contents

About This Content

This guide describes how to install and administer Oracle Communications Elastic Charging Engine (ECE) composable services.

Audience

This guide is intended for anyone who installs, configures, administers, customizes, or uses ECE composable services.

1

Overview of the ECE Composable Service Deployment

The Oracle Communications Elastic Charging Engine (ECE) composable services are deployed in a containerized and orchestrated deployment architecture.

For more information about the ECE composable services, see *ECE Composable Services User's Guide*.

Topics in this document:

About Deploying the ECE Composable Services

The ECE composable services are deployed in a cloud native environment. They support a Kubernetes-orchestrated containerized multiservice architecture to facilitate continuous integration, continuous delivery, and DevOps practices.

The ECE composable services support the following deployment models:

- On Private Kubernetes Cluster: The ECE composable services can run on a general deployment of Kubernetes.
- On Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE): The ECE composable services can run on Oracle's hosted Kubernetes OKE service.
- On Oracle Cloud Native Environment: The ECE composable services can run on the Oracle Cloud Native Environment.

ECE Composable Services Architecture

In the ECE composable services architecture, each service runs as a container and is deployed as a Kubernetes pod, which is the fundamental building block of Kubernetes. The services can be deployed and managed as multiple replicas within a Kubernetes replica set.

About the ECE Composable Service Pods

Table 1-1 lists the ECE composable service pods, their containers, and the services exposed through them.

Table 1-1 ECE Composable Service Pods

Pod Name	Admin Port	Container Port	Service Type (Default)
cgf-coordinator	8081/TCP	NA	ClusterIP
charging-gateway	8081/TCP	NA	ClusterIP
nchf-converged-charging	8081/TCP	8080/TCP	ClusterIP
nchf-nrf	8081/TCP	NA	ClusterIP

About the ECE Composable Service Processors

The ECE composable services use a processor-based framework to control how requests are handled. Processors define the individual steps that a service performs on events or records as they move through its pipeline. Each processor performs a specific function within the request-processing pipeline, such as mediation, transformation, or logging. By combining processors into a processing chain, you can control how a service handles incoming events and prepares them for downstream systems.

Processors run sequentially within a service. The output from one processor becomes the input to the next processor in the chain. This modular approach allows you to customize processing behavior while keeping the configuration flexible and easier to maintain.

Table 1-2 lists the processor types supported by the ECE CHF and CGF composable services.

Table 1-2 Processor Types Used by ECE Composable Services

Processor Type	Composable Services that Use It (by Default)	Description
AGGREGATOR_PROCESSOR	CGF	This processor aggregates unrated events into unrated 5G CDRs based on the configured partial-record method. The processor also detects duplicate events, missing invocation sequence numbers (ISNs), and out-of-order events during processing. Use this processor as the primary mechanism for generating unrated 5G CDRs from incoming unrated events.
CLOUDEVENT_LOGGER	CGF	This processor writes CloudEvent payloads to the application logs for debugging and operational analysis. The processor does not transform or publish the event. Use this processor to inspect CloudEvent data as it moves through the processing pipeline.
COMPOSITE_PROCESSOR	CGF and CHF	This processor groups multiple processors into a single logical processing chain. Use this processor to organize related processing steps, such as deduplication and publishing, within a unified workflow. This processor simplifies configuration and improves pipeline maintainability.

Table 1-2 (Cont.) Processor Types Used by ECE Composable Services

Processor Type	Composable Services that Use It (by Default)	Description
DEDUP_PROCESSOR	CGF	This processor detects and suppresses duplicate events based on a configured deduplication key and time window. It helps prevent duplicate processing that can occur because of retries, retransmissions, or at-least-once delivery semantics. Use this processor to improve data integrity and reduce duplicate downstream records.
KAFKA_PUBLISHER_PROCESSOR	CGF and CHF	This processor publishes processed records to outbound Kafka topics for downstream consumption. It supports durable retry handling, message key generation, topic creation, and topic modification. Use this processor to provide reliable event delivery between ECE composable services, such as from CHF to CGF, and from the ECE composable services to external consumers, such as CGF delivery of rated and unrated CDRs to billing systems and other external consumers.
LOGGING_PROCESSOR	CGF and CHF	This processor writes request and event details to the application logs to help operators monitor processing activity and troubleshoot issues. This processor does not modify the payload or affect message routing. Use this processor to improve operational visibility within the processing pipeline.
MUTATOR_PROCESSOR	CHF	This processor applies request or response transformations by using configurable mutation rules and expressions. Use this processor for payload enrichment, normalization, or session identifier manipulation.
NCHF	CHF	This processor is the core CHF orchestration processor responsible for evaluating charging requests, coordinating downstream processing, and managing charging workflow behavior.
NCHF_LOGGING	CHF	This is a specialized logging processor for capturing and logging CHF request and response details. It is typically used for troubleshooting or request tracing.
NCHF_REST_PROCESSOR	CHF	This processor forwards eligible online charging requests to the ECE HTTP Gateway for rating, quota management, and charging-session processing. It supports retry and circuit-breaker fault-tolerance policies.
REST_PROCESSOR	CHF	This processor sends REST requests to ECE composable services or external services. It is commonly used for spending limit control integration and notification-related processing.

Configure processors to run in the order that matches the logical flow of event processing. In most deployments, processors that inspect or validate data run first, processors that transform or aggregate data run next, and processors that publish or deliver data run last.

For example, a typical unrated 5G CDR pipeline in CGF uses the following order:

1. LOGGING_PROCESSOR: Logs incoming events for troubleshooting and operational visibility.

2. AGGREGATOR_PROCESSOR: Aggregates unrated events into unrated 5G CDRs and detects duplicates, gaps, and out-of-order events.
3. KAFKA_PUBLISHER_PROCESSOR: Publishes the unrated 5G CDRs to an outbound Kafka topic.

Using a clear processor order helps ensure predictable processing behavior, improves troubleshooting, and reduces the risk of duplicate or incomplete downstream records.

Part I

Getting Started with Deployment

This part provides information about getting started with your Oracle Communications Elastic Charging Engine (ECE) composable services deployment, including installing the prerequisite software and downloading the deployment package.

It contains the following chapters:

2

About Configuring and Deploying the ECE Composable Services

Learn about the high-level steps for configuring and deploying the Oracle Communications Elastic Charging Engine (ECE) composable services.

Topics in this document:

About Configuring and Deploying the ECE Composable Services

You install the ECE composable services by configuring and deploying their Helm chart. The Helm chart includes YAML template descriptors for all Kubernetes resources and a **values.yaml** file that provides default configuration values for each chart.

When you install the Helm chart, Helm generates valid Kubernetes manifest files by replacing default values from the **values.yaml** file with custom values from your **override-values.yaml** file, and creates Kubernetes resources.

> **Note**
>
> This documentation uses the **override-values.yaml** file name for ease of use, but you can name the file whatever you want.

High-Level Installation Tasks

You install the ECE composable services by performing these high-level tasks:

1. Install and configure all prerequisite software.

   See "Setting Up Prerequisite Software".

2. Download the ECE composable services, extract the Helm chart, and load their images.

   See "Preparing Your System for ECE Composable Services".

3. Configure the ECE composable services needed to implement your desired features.
   - To generate unrated 5G CDRs, see "Generating Unrated 5G CDRs with CHF and CGF".
   - To integrate the Charging Manager composable service into your existing ECE cloud native system, see "Integrating ECE CHF Composable Service with ECE for 5G Charging".

4. Deploy the ECE composable services in your cloud native environment:

- If you are migrating the CHF and CGF composable services into an existing ECE active-active deployment, see "Migrating the CHF and CGF Composable Services into an Existing ECE System".
- If you are deploying the CHF and CGF composable services independently, see "Deploying the ECE Composable Services".

3

Setting Up Prerequisite Software

You must perform prerequisite tasks, such as installing Kubernetes and Helm, before deploying the Oracle Communications Elastic Charging Engine (ECE) composable services on your cloud native environment.

Topics in this document:

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

Prerequisite Tasks for ECE Composable Services

As part of preparing your environment for the ECE composable services, 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 the ECE composable services. See *BRM Compatibility Matrix*.
2. Create a Kubernetes cluster.
3. Establish network access.
4. Install a container platform supported by Kubernetes, such as Docker, Podman, or containerd.
5. Install Helm.
6. Set up your client machine.
7. Install Apache Kafka.
8. Set up your Oracle cnDBTier database.

9. Install and configure an ingress controller.
10. If you plan to trace the flow of API calls through the ECE composable services, install and configure Jaeger.

    For more information about tracing, see "Tracing the Flow of API Calls".
11. If you plan to autoscale your pods using Kubernetes Horizontal Pod Autoscaler:
    - Install and configure Kubernetes Metrics Server.
    - Optionally install and configure a service mesh.

    For more information about autoscaling, see "Setting Up Autoscaling of Pods".
12. If you plan to monitor the ECE composable service operations:
    - Install and configure Prometheus Operator.
    - Install and configure Grafana.

    For more information, see "Monitoring ECE Composable Service Processes".

> **Note**
>
> Prometheus and Grafana are the preferred observability tools, and the ECE composable services include templates for these tools. The ECE composable services also support OpenTelemetry and OpenMetrics standards, which enable integration with other observability tools.

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

Installing Apache Kafka

The ECE composable services require Apache Kafka and use it as an internal message bus.

Installing Oracle cnDBTier

Oracle cnDBTier is a key component of the 5G Service-Based Architecture. It is the geodiverse database layer provided as part of the Cloud Native Environment and provides persistent storage for the state data and subscriber data in a cloud environment.

To install and configure cnDBTier:

1. Install and configure cnDBTier by following the instructions in Cloud Native Core, cnDBTier Installation, Upgrade, and Fault Recovery Guide.

   For the list of supported cnDBTier versions, see "ECE Composable Services Software Compatibility Requirements" in *BRM Compatibility Matrix*.
2. Ensure you create and write down the database administration user name and password.
3. Before deploying cnDBTier, configure the **custom_values.yaml** keys in the following list based on your deployment sizing:
   - **additionalndbconfigurations.api.TotalSendBufferMemory**
   - **additionalndbconfigurations.mgm.TotalSendBufferMemory**
   - **additionalndbconfigurations.ndb.FragmentLogFileSize**

- **additionalndbconfigurations.ndb.MaxNoOfConcurrentOperations**
- **additionalndbconfigurations.ndb.MaxNoOfConcurrentScans**
- **additionalndbconfigurations.ndb.MaxNoOfExecutionThreads**
- **additionalndbconfigurations.ndb.NoOfFragmentLogFiles**
- **additionalndbconfigurations.ndb.RedoBuffer**
- **additionalndbconfigurations.ndb.TotalSendBufferMemory**
- **additionalndbconfigurations.tcp.ReceiveBufferMemory**
- **additionalndbconfigurations.tcp.SendBufferMemory**
- **additionalndbconfigurations.tcp.TCP_RCV_BUF_SIZE**
- **additionalndbconfigurations.tcp.TCP_SND_BUF_SIZE**
- **global.ndbappReplicaCount**
- **global.ndbappReplicaMaxCount**
- **global.ndbReplicaCount**
- **mgm.ndbdisksize**
- **mgm.resources.limits.cpu**
- **mgm.resources.limits.memory**
- **mgm.resources.requests.cpu**
- **mgm.resources.requests.memory**
- **ndb.datamemory**
- **ndb.ndbbackupdisksize**
- **ndb.ndbdisksize**
- **ndb.resources.limits.cpu**
- **ndb.resources.limits.memory**
- **ndb.resources.requests.cpu**
- **ndb.resources.requests.memory**
- **ndbapp.ndbdisksize**
- **ndbapp.resources.limits.cpu**
- **ndbapp.resources.limits.memory**
- **ndbapp.resources.requests.cpu**
- **ndbapp.resources.requests.memory**

Exposing the ECE Composable Services Outside the Kubernetes Cluster

Using an ingress controller exposes ECE composable services outside the Kubernetes cluster and allows clients to communicate with the services. Ingress controllers monitor ingress objects and act on the configuration embedded in these objects to expose HTTP services to the external network.

Adding an external load balancer provides a 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 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 ECE composable services 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.

Installing Jaeger

The Jaeger tracing tool helps you trace the flow of messages through the ECE composable services, 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 ECE composable service pods.

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

Installing Prometheus Operator

Prometheus Operator is an open-source toolkit that scrapes metric data from the ECE composable services and then stores it in a time-series database. You use it to monitor the operation of ECE composable service processes. See "Monitoring ECE Composable Service 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 the ECE composable services to view performance data.

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

4

Preparing Your System for ECE Composable Services

You can prepare your cloud native environment by downloading the Oracle Communications Elastic Charging Engine (ECE) composable services package, extracting the Helm chart, and loading its images.

Topics in this document:

Tasks for Preparing Your System

As part of preparing your cloud native environment for the ECE composable services, you download the package and then load the necessary images into your Kubernetes container system.

The following shows the high-level tasks:

1. Download the package from My Oracle Support to your cloud native environment.
2. Extract the Helm charts from the downloaded package.
3. Load the images into your container runtime.
4. Create the required user IDs, passwords, and Secrets.
5. (Optional) Create Secrets for your container registry.

Downloading the Deployment Package

To download the ECE composable services package, go to the My Oracle Support website (https://support.oracle.com). Search for and then download ECE Interim Patch 15.2.0.0.2.

Table 4-1 lists the packages in the downloaded archive file. Replace *version* with the release number. For example, for release 15.2 Patch Set 2, the file name for the Charging Gateway Function composable service would be **oc-ccs-charging-gateway-15.2.0.0.2.tar**.

Table 4-1 ECE Composable Service Packages

Package Name	File Name
Charging Manager	**oc-ccs-nchf-converged-charging-***version***.tar**
Charging Gateway Function	**oc-ccs-charging-gateway-***version***.tar**
Database Client	**oc-ccs-db-client-***version***.tar**

Table 4-1 (Cont.) ECE Composable Service Packages

Package Name	File Name
NRF Management	**oc-ccs-nrf-management-*version*.tar**
ECE Composable Services Helm Chart	**oc-ccs-15.2.0.0.2+d9e763dc.tgz**

Extract the Helm chart by running this command:

```
tar -xvzf oc-ccs-15.2.0.0.2+d9e763dc.tgz
```

Loading ECE Composable Service Images

The images shipped with the ECE composable services 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 Charging Gateway Function image into a Docker system, you would perform the following steps:

1. Download the **oc-ccs-charging-gateway-*version*.tar** file to the system where Docker is installed, where *version* is the release number, such as **15.2.0.0.2**.
2. Load the image into Docker by entering this command:

   ```
   docker load --input oc-ccs-charging-gateway-version.tar
   ```

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

   ```
   docker images charging-gateway: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 would push the image to the registry like this:

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

   ```
   docker tag charging-gateway:version RepoHost:RepoPort/oracle/cagbu/ccs/
   charging-gateway:version
   ```

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

   ```
   docker push RepoHost:RepoPort/oracle/cagbu/ccs/charging-gateway:version
   ```

Creating Users and Secrets for the ECE Composable Services

To create the users and Secrets required to deploy the ECE composable services:

1. As the database administrator, create the CGF service user name and password that the CGF uses at runtime to connect to the CGF database schema.
2. Create a Kubernetes Secret that contains the CGF service user credentials.
3. Create a Kubernetes Secret that contains the database administrator credentials.

For information about creating Kubernetes Secrets, see *Managing Secrets* in the Kubernetes documentation: https://kubernetes.io/docs/tasks/configmap-secret/.

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 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 ECE composable services Helm chart.

For example:

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

Part II

Installing and Configuring the ECE Composable Services

This part provides information about configuring and deploying the Oracle Communications Elastic Charging Engine (ECE) composable services.

It contains the following chapters:

5

Generating Unrated 5G CDRs with CHF and CGF

You can generate unrated 5G call detail records (CDRs) by using the Oracle Communications Elastic Charging Engine (ECE) composable services. You might want unrated CDRs for roaming partners, data warehousing, and legacy billing systems.

Topics in this document:

About Generating 5G Unrated CDRs with CHF and CGF

The ECE composable services generate and process unrated 5G CDRs by using the Charging Manager (CHF) and Charging Gateway Function (CGF) composable services. The CHF receives charging requests from 5G network functions, such as the Session Management Function (SMF), and publishes unrated charging events to the CHF-CGF Kafka topic. The CGF consumes and processes these events and publishes unrated 5G CDRs for downstream systems and northbound consumers.

The CGF composable service uses a cnDBTier as a persistent state store for operational and processing data. During unrated 5G CDR processing, cnDBTier stores CDR state, processed ISNs, retry records, deduplication data, and durable publishing information. This enables CGF to track aggregation state, detect duplicate, missing, and out-of-order events, support retry and recovery workflows, maintain sequencing and gap-handling data, and persist failed Kafka publication retries. As the authoritative state store, the cnDBTier ensures reliable CDR processing during retries, failovers, Kafka outages, and multisite recovery scenarios.

The ECE composable services support unrated CDR replication to enhance service resiliency and operational continuity in multisite deployments. This capability improves sequencing accuracy, duplicate detection, and aggregation consistency during transaction failover scenarios, helping ensure reliable and uninterrupted processing across sites.

> **Note**
>
> In contrast, ECE with CDR Gateway relies on an Oracle database that does not support replication. As a result, during a transaction failover, the system cannot reliably determine whether another site has already processed the transaction. For information about ECE with CDR Gateway, see "Generating CDRs for External Systems" in *ECE Implementing Charging*.

High-Level Configuration Tasks for Generating Unrated 5G CDRs

You can generate unrated 5G CDRs using the ECE CHF and CGF composable services by integrating CHF and CGF into an existing ECE cloud native deployment.

To configure the ECE to generate unrated CDRs with CHF and CGF:

1. Ensure you have installed and configured all prerequisite software, such as Kafka Server and Oracle cnDBTier.

   See "Setting Up Prerequisite Software".
2. Create an **override-values.yaml** file for the ECE composable services Helm chart.
3. Configure the mandatory values for generating unrated CDRs for 5G clients.

   See "Configuring Mandatory Values for Generating Unrated CDRs".
4. Configure the CHF composable service to send unrated events to the CGF composable service through the CHF-CGF Kafka topic.

   See "Configuring the Charging Manager to Generate Unrated CDRs".
5. Configure the CGF composable service to aggregate and clean up unrated events, convert them into 5G unrated CDRs, and send them to 5G clients through the CGF-CDR Kafka topic.

   See "Configuring the Charging Gateway Function to Generate Unrated CDRs".
6. Do one of the following:
   - If you are replacing the CDR Gateway component in your existing ECE deployment with the ECE CHF and CGF composable services, see "Migrating the CHF and CGF Composable Services into an Existing ECE System".
   - If you are deploying CHF and CGF in a standalone cloud native environment, see "Deploying the ECE Composable Services".
7. Set up your external system to consume the unrated 5G CDRs from the CGF-CDR Kafka topic.

Configuring Mandatory Values for Generating Unrated CDRs

To configure the ECE composable services to generate unrated 5G CDRs, set the following keys in your **override-values.yaml** file for **oc-ccs-*version***. These keys are required to create a simple or demonstration version that you can use immediately.

These keys apply to all ECE composable services, but you can override them for a specific composable service by setting a different value in the composable service's key. For example, to use a different pull policy for the Charging Gateway Function composable service, set the **cgf.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 ""
imagePullSecrets: &imagePullSecrets
imagePullPolicy: &imagePullPolicy IfNotPresent
nodeSelector: &nodeSelector {}

partOfLabel: &partOfLabel oc-ccs
additionalLabels: &additionalLabels { }
annotations: &annotations { }
```

```
# shared webserver defaults
webserverDefaults: &webserver
  maxPayloadSize: 2000000

# common component images
dbClientImage: &dbClientImage "ccs/infra/db-client:15.2.0.0.2"
# ccs converged charging component images
nrfManagementImage: &nrfManagementImage "ccs/nrf-management:15.2.0.0.2"
nchfConvergedChargingImage: &nchfConvergedChargingImage "ccs/chf/nchf-converged-
charging:15.2.0.0.2"
cgfImage: &cgfImage "ccs/charging-gateway:15.2.0.0.2"

# database services
db:
  cgf:
    serviceHostname: &db-cgf-serviceHostname
    adminCredentialsSecretName: &db-cgf-adminCredentialsSecretName
    serviceCredentialsSecretName: &db-cgf-serviceCredentialsSecretName

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

tracingEnabled: &tracingEnabled false
tracingHost: &tracingHost otlp-collector-host
tracingPort: &tracingPort 14250

nchfTopicName: &nchfPublisherTopicName nchf-cgf-cdr

nchfCdrMultiSite: &nchf-cdr-multisite
  enabled: false
  coordinationId: "nchf-cdr-coordination"
  failoverMode: ALL_REMAINING
  kafkaFailoverStabilizationWindow: "PT0S"
  remoteSites:
    - site2
    - site3

kafka-brokers:
  ccs-broker-producer-1: &ccs-broker-producer-1
    name: ccs-broker-producer
    bootstrapServers: kafka:9093
    auth:
      enabled: true
      tls:
        trustStore:
          existingSecret: cgf-billing-cluster-ca-cert
          passwordKey: ca.password
        keystore:
          existingSecret: cgf-billing-user
          passwordKey: user.password
    extraConfig:
      reconnect.backoff.ms: "100"
      reconnect.backoff.max.ms: "1000"
      max.block.ms: "30000"
      request.timeout.ms: "15000"
      delivery.timeout.ms: "60000"
  ccs-broker-consumer-1: &ccs-broker-consumer-1
    name: ccs-broker-consumer
    bootstrapServers: kafka:9093
    auth:
      enabled: true
```

```
      tls:
        trustStore:
          existingSecret: cgf-cluster-ca-cert
          passwordKey: ca.password
        keyStore:
          existingSecret: nchf-user
          passwordKey: user.password
    extraConfig:
      auto.offset.reset: latest
      reconnect.backoff.ms: "175"
      reconnect.backoff.max.ms: "1000"
      allow.auto.create.topics: "false"

serviceMonitor: &serviceMonitor
  enabled: false
  namespace:
  additionalLabels: {}

grafanaDashboards: &grafanaDashboards
  enabled: false
  grafanaNamespace:
  labels:
    grafana_dashboard: "1"
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"
```

Table 5-1 describes each key.

Table 5-1 Mandatory Keys for Generating Unrated CDRs

Key	Description
imageRepository	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	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	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.
nodeSelector	The nodes on which to deploy the ECE composable services. Labels appear underneath **nodeSelector** in a set of key-value pairs.
partOfLabel	The value to assign to the composable service's **app.kubernetes.io/part-of label**. This label identifies the application that the resource belongs to. The default is **oc-ccs**.
additionalLabels	The labels map used when the ECE composable services are deployed.
annotations	The key-value map, which is read to determine behavior or actions.

Table 5-1 (Cont.) Mandatory Keys for Generating Unrated CDRs

Key	Description
webserverDefaults.maxPayloadSize	The maximum size of data in the request body.
dbClientImage	The name of the Database Client image. The default is **ccs/infra/db-client:15.2.0.0.2**.
nrfManagementImage	The name of the NRF image. The default is **ccs/nrf-management:15.2.0.0.2**.
nchfConvergedChargingImage	The name of the nCHF image. The default is **ccs/chf/nchf-converged-charging:15.2.0.0.2**.
cgfImage	The name of the CGF image. The default is **ccs/charging-gateway:15.2.0.0.2**.
db.cgf.*	The details about the database services for CGF. • **serviceHostname**: The host name for the CGF database service. • **adminCredentialsSecretName**: The name of the Kubernetes Secret that holds the database administrator credentials used by the CGF database. This is the Secret you created in "Creating Users and Secrets for the ECE Composable Services". • **serviceCredentialsSecretName**: The name of the Kubernetes Secret that holds the CGF service user credentials used at runtime to connect to the CGF database. This is the Secret you created in "Creating Users and Secrets for the ECE Composable Services".
loggingPattern	The structure, format, and content of log messages.
tracingEnabled	Whether tracing through Jaeger is enabled. The default is **false**.
tracingHost	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	The port number for Jaeger. The default is **14250**. **Note:** This key applies only if tracing is enabled.
nchfTopicName	The name of the CHF-CGF Kafka topic. The CHF publishes unrated events to this topic. The default is **nchf-cgf-cdr**.

Table 5-1 (Cont.) Mandatory Keys for Generating Unrated CDRs

Key	Description
nchfCdrMultiSite.*	The details for deploying the CGF on multiple sites. • **enabled**: Whether to deploy nCHF on multiple sites. • **coordinationId**: The correlation key used to tie a request or failover decision together across sites. • **failoverMode**: The CGF multisite failover mode, which is driven by a site-aware Kafka partition assignment strategy, where local-site consumers have highest priority and automatically take ownership of partitions, with remote sites acting as backups. – **NEXT_SITE**: A single designated backup site takes over all processing when the primary site fails. – **ALL_REMAINING**: All available remote sites share the load during a failure. When the failed site recovers, partition ownership automatically shifts back to the local site, restoring the original active-active balance. This is the default. • **kafkaFailoverStabilizationWindow**: The grace period in ISO 8601 duration format. CGF waits after Kafka looks healthy again before switching consumers back and treating the site as stable. It helps prevent flapping during recovery and makes sure broker and topic health, partition assignment, and consumer groups have settled before normal processing resumes. The default value is **PT0S**, which means there is no waiting period. • **remoteSites**: The name of each remote site.

Table 5-1 (Cont.) Mandatory Keys for Generating Unrated CDRs

Key	Description
kafka-brokers.ccs-broker-producer-1	The details for connecting the producer to the Kafka Server: • **name**: The name of the Kafka broker producer. The default is **ccs-broker-producer**. • **bootstrapServers**: The host name and port pair for the Apache Kafka cluster. The default is **kafka:9093**. • **auth.enabled**: Whether the Kafka client is configured to use authenticated TLS for its broker connection (**true**) or not (**false**). The default is **true**. • **auth.tls.truststore.existingSecret**: The name of the Kubernetes Secret that contains the Certificate Authority (CA) file. The default is **cgf-billing-cluster-ca-cert**. • **auth.tls.truststore.passwordKey**: The data key inside the Secret that contains the TrustStore password. The default is **ca.password**. • **auth.tls.keystore.existingSecret**: The name of the Kubernetes Secret that contains the client identity KeyStore and its password. The default is **cgf-billing-user**. • **auth.tls.keystore.passwordKey**: The data key inside the Secret that holds the KeyStore password. The default is **user.password**. • **extraConfig.reconnect.backoff.ms**: The initial wait, in milliseconds, before retrying a connection to a broker after a failure. The default is **100**. • **extraConfig.reconnect.backoff.max.ms**: The maximum delay between successive TCP connection attempts to a broker after a connection is lost or fails. The default is **1000** (1 second). • **extraConfig.max.block.ms**: The Kafka client timeout limiting how long a producer or admin client blocks while waiting for resources such as cluster metadata, buffer space, or a connection to the broker. The default is **30000**. • **extraConfig.request.timeout.ms**: The amount of time the Kafka client waits for a response to a request before retrying or failing the operation once retries are exhausted. The default is **15000**. • **extraConfig.delivery.timeout.ms**: The Kafka producer's overall upper bound on how long a record can take from send until Kafka reports success or failure. The default is **60000**.

Table 5-1 (Cont.) Mandatory Keys for Generating Unrated CDRs

Key	Description
kafka-brokers.ccs-broker-consumer-1	The details for connecting the Charging Gateway consumer to the Kafka Server: • **name**: The name of the Kafka broker consumer. The default is **ccs-broker-consumer**. • **bootstrapServers**: The host name and port pair for the Apache Kafka cluster. The default is **kafka:9093**. • **auth.enabled**: Whether the Kafka client is configured to use authenticated TLS for its broker connection (**true**) or not (**false**). The default is **true**. • **auth.tls.truststore.existingSecret**: The name of the Kubernetes Secret that contains the Certificate Authority (CA) file. The default is **nchf-cluster-ca-cert**. • **auth.tls.truststore.passwordKey**: The data key inside the Secret that contains the TrustStore password. The default is **ca.password**. • **auth.tls.keystore.existingSecret**: The name of the Kubernetes Secret that contains the client identity KeyStore and its password. The default is **nchf-user**. • **auth.tls.keystore.passwordKey**: The data key inside the Secret that holds the KeyStore password. The default is **user.password**. • **extraConfig.auto.offset.reset**: Where to start reading the log when a consumer group has not committed an offset for a partition: **latest**, **earliest**, or **none**. The default is **latest**, which means read only new records. • **extraConfig.reconnect.backoff.ms**: The initial wait, in milliseconds, before retrying a connection to a broker after a failure. The default is **175**. • **extraConfig.reconnect.backoff.max.ms**: The maximum delay between successive TCP connection attempts to a broker after a connection is lost or fails. The default is **1000** (1 second). • **extraConfig.allow.auto.create.topics**: Whether the client can trigger automatic creation of a topic on the broker when it references a non-existent topic (**true**) or not (**false**). The default is **false**.
serviceMonitor.*	The details for the Service Monitor: • **enabled**: Whether to enable Kubernetes Service Monitor, which is used to monitor a group of services. The default is **false**. • **namespace**: The namespace in which the Service Monitor is deployed. • **additionalLabels**: The labels map used when Service Monitor is deployed. See "Monitoring ECE Composable Service Processes".
grafanaDashboards.*	The details for the Grafana Dashboards: • **enabled**: Whether to enable Grafana Dashboards for Grafana Operator. The default is **false**. • **grafanaNamespace**: The namespace in which Grafana is deployed. • **labels.grafana_dashboard**: The labels to add to Grafana CRDs. This helps Grafana discover the dashboards. The default is **1**. • **annotations.k8s-sidecar-target-directory**: The directory in which the Grafana Dashboards are deployed. The default is **/tmp/dashboards/ccs**. See "Monitoring ECE Composable Service Processes".

Configuring the Charging Manager to Generate Unrated CDRs

When configured for unrated 5G CDR processing, the CHF receives charging requests from 5G network functions, converts them into unrated events, and publishes the unrated events to the CHF-CGF Kafka topic.

To configure the CHF pod, set the following keys in your **override-values.yaml** file:

```
nchfConvergedCharging:
  enabled: true
  imageRepository: *imageRepository
  imagePullSecrets: *imagePullSecrets
  imagePullPolicy: *imagePullPolicy
  partOfLabel: *partOfLabel
  webserver: *webserver

  nchfConvergedCharging:
    name: nchf-converged-charging
    fullname: "nchf-converged-charging"
    additionalLabels: *additionalLabels
    annotations: *annotations
    nodeSelector: *nodeSelector
    container:
      image: *nchfConvergedChargingImage
    mutatorRulesConfigMapOverride:

    restServices:
      - name: "ConvergedCharging"
        type: NCHF_CONVERGED_CHARGING
        config:
          path: "/nchf-convergedcharging/v3/chargingdata"
          ingressOrigin:
        processors:
          - name: mediation-processor
            type: COMPOSITE_PROCESSOR
            processors:
              - name: mutation-processor-standin
                producerOnly: true
                type: LOGGING_PROCESSOR
                config:
                  logLevel: INFO
                  atMostSeconds: 10
          - name: nchf-processor
            type: NCHF
            config:
              subscriberValidator:
                enabled: false
                config:
                  baseUri: "http://subscriber-profile-repository:8080/
subscriberProfileRepository/v1"
              sessionManager:
                enabled: false
                config:
                  baseUri: "http://session-manager:8080/sessionManager/v1"
                  expiryIntervalInSeconds: 300
            processors:
              - name: transform-nchf-request
                type: MUTATOR_PROCESSOR
                config:
                  rules:
                    - source: INLINE
```

```
                        rule:
                          ruleName: onlineSessionIdEnricher
                          criterion:
                            expression:
                              nchfRequest:
                                criteriaType: MVEL_EXPRESSION
                                expression: "value.requestType == RequestType.CREATE &&
value.isOnlineChargingRequest()"
                          mutation:
                            expression:
                              associatedSession:
                                - mutationType: MVEL_TRANSFORM
                                  expression: "value.id = 'online_'+value.id"
                - name: cdrPublisher
                  type: KAFKA_PUBLISHER_PROCESSOR
                  config:
                    publisher:
                      name: nchf
                      topic:
                        name: nchf-cgf-cdr
                        numberOfPartitions: 2
                        replicationFactor: 2
                        create:
                          enabled: true
                        modification:
                          enabled: true
                      keyGenerations:
                        - criteria: $.subscriberIdentifier
                          keyPaths:
                            - $.subscriberIdentifier
                      broker: *ccs-broker-producer-1

    networkFunction:
      nfProfile: &profileFile nfProfile.json
      existingNfProfileConfigMap: &existingNfProfileConfigMap

    terminationGracePeriodSeconds: 60
    resources:
      memoryRequest: "256Mi"
      cpuRequest: "250m"
      memoryLimit: "512Mi"
      cpuLimit: "500m"
    replicas: 1
    jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:InitialRAMPercentage=80 -
XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -Dlog4j.configurationFile=/app/config/
log4j2.yaml"
    restartCount: 0

    service:
      additionalLabels: *additionalLabels
      additionalAnnotations: *annotations
      type: ClusterIP

    adminService:
      additionalLabels: *additionalLabels
      additionalAnnotations: *annotations
      type: ClusterIP

    logging:
      format:
        type: TEXT
        pattern: *loggingPattern
```

```
    rootLevel: INFO
    packageLogging:
      - name: com.oracle.cagbu
        level: INFO
      - name: io.helidon
        level: INFO

  tracing:
    enabled: *tracingEnabled
    service: nchf-converged-charging
    maxQueueSize: 42
    flushIntervalMs: 10001
    host: *tracingHost
    port: *tracingPort
    samplerType: ratio
    samplerParam: 0.01
    samplerManager:
    logSpans: true
    webServer:
      spans:
        httpRequest:
          contentWrite: false
          contentRead: false
   hpa:
    enabled: true
    minReplicas: 1
    maxReplicas: 8
    metrics:
      cpuAverageUtilization: 65
    scaleDown:
      selectPolicy: Max
      stabilizationWindowSeconds: 300
      periodSeconds: 180
    scaleUp:
      selectPolicy: Max
      stabilizationWindowSeconds: 5
      periodSeconds: 20
```

Table 5-2 describes each key.

Table 5-2 Charging Manager Keys

Key	Path in values.yaml	Description
enabled	**nchfConvergedCharging**	Whether the CHF pod is enabled. The default is **true**.
imageRepository	**nchfConvergedCharging**	Specifies the registry server where you pushed images, typically in this format: *RepoHost*:*RepoPort*. **Note:** The composable service uses the global **imageRepository** setting by default. Set this key only if you want to use a different registry server.
imagePullSecrets	**nchfConvergedCharging**	Specifies the name of the Secret that contains credentials for accessing images from your private image server. **Note:** The composable service uses the global **imagePullSecrets** setting by default. Set this key only if you want to use a different Secret.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
imagePullPolicy	**nchfConvergedCharging**	Sets when Kubernetes pulls images: **Always**, **IfNotPresent**, or **Never**. **Note:** The composable service uses the global **imagePullPolicy** setting by default. Set this key only if you want to use a different setting.
partOfLabel	**nchfConvergedCharging**	Sets the value to assign to the composable service's **app.kubernetes.io/part-of** label. This label identifies the application that the resource belongs to. **Note:** The composable service uses the global **partOfLabel** setting by default. Set this key only if you want to use a different setting.
webserver	**nchfConvergedCharging**	The Helidon HTTP server configuration.
nchfConvergedCharging.*	**nchfConvergedCharging**	The details about the CHF. • **name**: The name of the pod. This is used to identify the pod in the cluster. The default is **nchf-converged-charging**. • **fullname**: The full name of the composable service. This name is used in place of the *ReleaseName-Name* pattern. The default is **nchf-converged-charging**. **Note:** The full name must be unique across the sites. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **annotations**: The additional annotations to add to a Kubernetes CRD. • **nodeSelector**: The nodes on which to deploy the CHF composable service. • **container.image**: The name of the CHF image.
mutatorRulesConfigMapOverride	**nchfConvergedCharging.nchfConvergedCharging**	The name of an existing ConfigMap that contains custom mutator rules to use instead of the default rules provided by the chart. Use this setting when you need to supply your own mutator rule configuration, such as environment-specific policies or customized transformation behavior. When this value is set, the deployment references the specified ConfigMap and bypasses the chart-managed default mutator rules ConfigMap.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
restServices[].*	**nchfConvergedCharging.nchfConvergedCharging**	The details about the REST services exposed by the CHF pod. • **name**: The name of the Converged Charging REST service that processes charging requests and generates unrated CDR output. The default is **ConvergedCharging**. • **type**: Set this to **NCHF_CONVERGED_CHARGING**. This is the service type for the Converged Charging request flow. • **config.path**: The REST endpoint used to receive charging requests. The default is **/nchf-convergedcharging/v3/chargingdata**. • **config.ingressOrigin**: The external origin used when constructing absolute response URLs, such as the HTTP Location header. This value should include only the scheme, host, and optional port.
processors.* (Mediation Processor)	**nchfConvergedCharging.nchfConvergedCharging**	The details about the mediation processor. • **name**: The name of the processor. For this configuration, **mediation-processor** identifies the top-level processor used to coordinate the configured processing chain. • **type**: The processor implementation type. **COMPOSITE_PROCESSOR** indicates that this processor contains and invokes one or more nested processors. • **processors.name**: The name of the nested processor. **mutation-processor-standin** identifies the default stand-in processor used in the chain. • **processors.producerOnly**: Whether the nested processor is used only on the producer side of the flow. When set to **true**, the processor is applied only when producing records. • **processors.type**: The implementation type of the nested processor. **LOGGING_PROCESSOR** indicates that the processor logs activity instead of modifying CDR data. • **processors.config.logLevel**: The logging level used by the logging processor. The default is **INFO**. • **processors.config.atMostSeconds**: The minimum interval, in seconds, between log messages. This setting limits how frequently the logging processor writes messages. The default is **10**.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
processors.* (NCHF Processor)	**nchfConvergedCharging.nchfConvergedCharging**	The ordered chain of processors executed for the charging request. • **name**: The name of the processor. The default is **nchf-processor**. • **type**: Set this to **NCHF**. This processor is the NCHF domain processor that handles converged charging logic. • **config.subscriberValidator.enabled**: Whether to enable subscriber validation for this deployment. The default is **false**. • **config.subscriberValidator.config.baseUri**: The base URL of the subscriber profile repository service used for subscriber lookups. • **config.sessionManager.enabled**: Whether to enable the session manager integration for this deployment. The default is **false**. • **config.sessionManager.config.baseUri**: The base URL of the session manager service. • **config.sessionManager.config.expiryIntervalInSeconds**: The session expiration interval, in seconds, used by the session manager integration.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
processors.processors.* (transform-nchf-request Sub-processor)	**nchfConvergedCharging.nchfConvergedCharging**	• **name**: The name of the processor: **transform-nchf-request**. • **type**: The processor type: **MUTATOR_PROCESSOR**. • **config.rules[].source**: The list of configured mutation rules. The default is **INLINE**, which specifies that the mutation rule is defined directly in the configuration file and is applied in place during request processing. • **config.rules[].rule.ruleName**: The name of the rule. The default is **onlineSessionIdEnricher**. • **config.rules[].rule.criterion.expression.nchfRequest.criteriaType**: The expression language used to evaluate the rule condition for an nCHF request. It tells CHF how to interpret the criterion expression and determine whether the mutation rule should be applied during request processing. The default is **MVEL_EXPRESSION**. • **config.rules[].rule.criterion.expression.nchfRequest.expression**: The condition that CHF evaluates to determine whether the mutation rule applies to an nCHF request. • **config.rules[].rule.mutation.expression.associatedSession.mutationType**: The transformation type used to modify the associated session data when CHF applies the mutation rule. The default is **MVEL_EXPRESSION**. • **config.rules[].rule.mutation.expression.associatedSession.expression**: The expression used to modify or enrich the associated session data during request processing.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
processors.processors.* (cdrPublisher Sub-processor)	**nchfConvergedCharging.nchfConvergedCharging**	Configure how the processor publishes the processed CHF request to Kafka. • **name**: The name of the processor. The default is **cdrPublisher**. • **type**: Set this to **KAFKA_PUBLISHER_PROCESSOR**. This processor publishes messages to Kafka. • **config.publisher.name**: The publisher name used to identify the outbound CDR publisher. The default is **nchf**. • **config.publisher.topic.name**: The name of the Kafka topic where unrated CDRs are published for CGF consumption. The default is **nchf-cgf-cdr**. • **config.publisher.topic.numberOfPartitions**: The number of partitions to create or use for the topic. The default is **2**. • **config.publisher.topic.replicationFactor**: The Kafka replication factor for the topic. The default is **2**. • **config.publisher.topic.create.enabled**: Whether to create the Kafka topic automatically if it does not already exist. • **config.publisher.topic.modification.enabled** : Whether to update the Kafka topic configuration if the existing topic does not match the desired settings. • **config.publisher.keyGenerations[0].criteria**: The subscriber identifier as the basis for key generation. • **config.publisher.keyGenerations[0].keyPaths[0]**: The subscriber identifier field as the Kafka key, which helps keep related CDRs for the same subscriber grouped consistently. • **config.publisher.broker**: A reference to the Kafka broker configuration used for publishing. The default is ***ccs-broker-producer-1**.
networkFunction.*	**nchfConvergedCharging.nchfConvergedCharging**	The details for how the service gets its NF profile information for NRF registration. • **nfProfile**: The NF profile JSON file that contains the network function's registration metadata, such as its identity, services, endpoints, and capabilities. • **existingNfProfileConfigMap**: The Kubernetes source that contains the NF profile.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
terminationGracePeriodSeconds	**nchfConvergedCharging.nchfConvergedCharging**	The amount of time, in seconds, the 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 after which the processes running in the pod are sent a termination signal. The processes are forcibly terminated. Set this value longer than the expected cleanup time for your process. The default is **60**.
resources.*	**nchfConvergedCharging.nchfConvergedCharging**	The minimum and maximum amount of memory and CPU that the coordinator can use. See "Setting Up Autoscaling of Pods".
replicas	**nchfConvergedCharging.nchfConvergedCharging**	The desired number of pod replicas. The default is **1**. This key is ignored if HPA is enabled.
jvmOpts	**nchfConvergedCharging.nchfConvergedCharging**	The JVM options to use.
restartCount	**nchfConvergedCharging.nchfConvergedCharging**	The number of times the CHF service has been restarted. Increment this value by 1 and run the Helm upgrade command to force a rolling restart of the CHF service. The default is **0**.
service.*	**nchfConvergedCharging.nchfConvergedCharging**	The details of the CHF service. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **additionalAnnotations**: The additional annotations to add to a Kubernetes CRD. • **type**: 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. This is the default.
adminService.*	**nchfConvergedCharging.nchfConvergedCharging**	The details of the admin service. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **additionalAnnotations**: The additional annotations to add to Kubernetes CRD. • **type**: 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. This is the default.

Table 5-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
logging.*	**nchfConvergedCharging.nchfConvergedCharging**	The logging levels for the Charging Manager composable service. See "Using Logging for ECE Composable Services".
tracing.*	**nchfConvergedCharging.nchfConvergedCharging**	Specifies how to set up tracing for the CHF composable service. See "Tracing the Flow of API Calls".
hpa.*	**nchfConvergedCharging.nchfConvergedCharging**	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 Pods".

Configuring the Charging Gateway Function to Generate Unrated CDRs

When configured to generate 5G unrated CDRs, the CGF does the following:

1. Consumes unrated events from the CHF-CGF Kafka topic
2. Aggregates the unrated events
3. (Optionally) Performs one or more of the following functions, depending on your configuration:
   - Detects and handles duplicate, missing, and out-of-order events
   - Tracks sequence information, such as processed ISNs, record sequence numbers, and local sequence numbers
   - Performs state handling
   - Cleans up records from the cnDBTier
   - Persists records in the cnDBTier
4. Transforms the unrated events into unrated 5G CDRs
5. Publishes the unrated 5G CDRs to the CGF-CDR topic

To configure the CGF to generate unrated 5G CDRs, set the following keys in your **override-values.yaml** file for the **oc-ccs-*version*** Helm chart:

```
cgf:
  enabled: true
  imageRepository: *imageRepository
  imagePullSecrets: *imagePullSecrets
  imagePullPolicy: *imagePullPolicy
  partOfLabel: *partOfLabel
  webserver: *webserver
  chargingGateway:
    name: cgf
    fullname: "cgf"
    leaderElection:
      enabled: true
      namePrefix: "cgf"
      namespace: ""
      identity: ""
      serviceAccount:
```

```
      create: true
      name: ""
      annotations: {}
    rbac:
      create: true
      resourceNames: []
  site:
    siteName: "site1"
    remoteSites: []
    failoverMode:
    kafkaFailoverStabilizationWindow: "PT0S"
  additionalLabels: *additionalLabels
  annotations: *annotations
  nodeSelector: *nodeSelector
  container:
    image: *cgfImage
  initContainer:
    image: *dbClientImage

  db:
    initializationFailTimeout: 10000
    connectionTimeout: 2000
    serviceHostname: *db-cgf-serviceHostname
    servicePort: 3306
    adminCredentialsSecretName: *db-cgf-adminCredentialsSecretName
    serviceCredentialsSecretName: *db-cgf-serviceCredentialsSecretName
    dbFailoverStabilizationWindow: "PT0S"
    siteStatusEndpoints:
      - siteName: "site1"
        host:
        port:
        path: "/db-tier/status/cluster/local/realtime"

  coordinator:
    enabled: true
    tasks:
      closedCdrRemoval:
        enabled: true
        cronSchedule: "0 0/5 * * * ?"
        config:
          ageOfRecord: PT5M
          batchSize: 10000
      suspectCdrRetry:
        enabled: false
        cronSchedule: "0 * * * * ? *"
        config:
          ageOfRecord: PT5M
          batchSize: 10000
    resources:
      memoryRequest: "512Mi"
      cpuRequest: "500m"
      memoryLimit: "1024Mi"
      cpuLimit: "1.0"
    jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:InitialRAMPercentage=80 -
XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -Dlog4j.configurationFile=/app/config/
log4j2.yaml"
    restartCount: 0
    adminService:
      additionalLabels: {}
      additionalAnnotations: {}
      type: ClusterIP
    messaging:
```

```
    services:
      - name: "ProcessSuspectCdr"
        type: KAFKA_CLOUDEVENT_CGF_REQUEST
        enabled: true
        multiSite: *nchf-cdr-multisite
        config:
          subscribers: []
        processors:
          - name: cdrPublisher
            type: KAFKA_PUBLISHER_PROCESSOR
            producerOnly: false
            config:
              publisher:
                name: cgf-cdr
                topic:
                  name: cgf-cdr-topic
                  numberOfPartitions: 2
                  replicationFactor: 2
                  create:
                    enabled: true
                  modification:
                    enabled: true
                keyGenerations:
                  - criteria: $.payload.chargingDataRequest
                    keyPaths:
                      - $.payload.chargingDataRequest.subscriberIdentifier
                broker:
                  name: ccs-broker
                  auth:
                    enabled: true
                    tls:
                      trustStore:
                        existingSecret: cgf-cluster-ca-cert
                        passwordKey: ca.password
                      keyStore:
                        existingSecret: cgf-user
                        passwordKey: user.password
                  bootstrapServers: kafka:9093
                  extraConfig:
                    reconnect.backoff.ms: "175"
                    reconnect.backoff.max.ms: "1000"

worker:
  messaging:
    services:
      - name: "ProcessNchfCdr"
        type: KAFKA_CLOUDEVENT_CGF_REQUEST
        enabled: true
        multiSite: *nchf-cdr-multisite
        config:
          subscribers:
            - name: nchf-chf-cdr
              brokerSite: site1
              failoverOrder:
                - site2
              instanceCount: 1
              processorRetryPolicy:
                retryCount: 3
                retryInterval: "PT0.2S"
                maxRequeueAttempts: 10
                retryTopic:
                  name: nchf-cgf-cdr-retry
```

```
                    suspendTopic:
                      name: nchf-cgf-cdr-suspend
                  topic:
                    name: *nchfPublisherTopicName
                  group: cgf-cdr-0
                  broker: *ccs-broker-consumer-1
              processors:
                - name: loggingProcessor
                  producerOnly: true
                  type: LOGGING_PROCESSOR
                  config:
                    logLevel: INFO
                    atMostSeconds: 5
                - name: aggregator
                  type: AGGREGATOR_PROCESSOR
                  config:
                    partialRecordMethod: DEFAULT
                - name: cdrPublisher
                  type: KAFKA_PUBLISHER_PROCESSOR
                  durability:
                    enabled: true
                    processRetryEnabled: true
                    batchSize: 1000
                    schedulerFrequencyInSec: 5
                    delayInSec: 10
                  producerOnly: false
                  config:
                    publisher:
                      name: cgf-cdr
                      topic:
                        name: cgf-cdr-topic
                        numberOfPartitions: 2
                        replicationFactor: 2
                        create:
                          enabled: true
                        modification:
                          enabled: true
                      keyGenerations:
                        - criteria: $.payload.chargingDataRequest
                          keyPaths:
                            - $.payload.chargingDataRequest.subscriberIdentifier
                      broker: *ccs-broker-producer-1
      resources:
        memoryRequest: "512Mi"
        cpuRequest: "500m"
        memoryLimit: "1024Mi"
        cpuLimit: "1.0"
      replicas: 1
      jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:InitialRAMPercentage=80 -
XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -Dlog4j.configurationFile=/app/config/
log4j2.yaml"
      restartCount: 0
      adminService:
        additionalLabels: *additionalLabels
        additionalAnnotations: *annotations
        type: ClusterIP
    terminationGracePeriodSeconds: 60
    logging:
      format:
        type: TEXT
        pattern: *loggingPattern
      rootLevel: INFO
```

```
    packageLogging:
      - name: com.oracle.cagbu
        level: INFO
      - name: io.helidon
        level: INFO
  tracing:
    enabled: *tracingEnabled
    service: charging-gateway
    maxQueueSize: 42
    flushIntervalMs: 10001
    host: *tracingHost
    port: *tracingPort
    samplerType: ratio
    samplerParam: 0.01
    samplerManager:
    logSpans: true
    webServer:
      spans:
        httpRequest:
          contentWrite: false
          contentRead: false
  hpa:
    enabled: true
    minReplicas: 1
    maxReplicas: 8
    metrics:
      cpuAverageUtilization: 65
    scaleDown:
      selectPolicy: Max
      stabilizationWindowSeconds: 300
      periodSeconds: 180
    scaleUp:
      selectPolicy: Max
      stabilizationWindowSeconds: 5
      periodSeconds: 20

grafanaDashboards: *grafanaDashboards
serviceMonitor: *serviceMonitor

prometheusRule:
  enabled: false
  namespace:
  additionalLabels: { }
  remoteConsumptionAlert:
    enabled: false
    window: 2m
    forDuration: 0m
    severity: warning
  retransmissionAlert:
    enabled: false
    window: 2m
    forDuration: 0m
    severity: warning
  suspendTopicAlert:
    enabled: false
    window: 2m
    forDuration: 0m
    severity: warning
  highCloudEventBacklogAlert:
    enabled: false
    processor: "cdrPublisher"
    threshold: 100
```

```
    forDuration: 5m
    severity: critical
  cloudEventRepositoryMinorAlert:
    enabled: false
    processor: "cdrPublisher"
    threshold: 0
    forDuration: 0m
    severity: warning
```

Table 5-3 describes each key.

Table 5-3 Charging Gateway Keys

Key	Path in values.yaml	Description
enabled	**cgf**	Specifies whether the charging-gateway pod is enabled. The default is **true**.
imageRepository	**cgf**	Specifies the registry server where you pushed images, typically in this format: *RepoHost***:***RepoPort*. **Note:** The ECE composable services use the global **imageRepository** setting by default. Set this key only if you want to use a different registry server.
imagePullSecrets	**cgf**	Specifies the name of the Secret that contains credentials for accessing images from your private image server. **Note:** The ECE composable services use the global **imagePullSecrets** setting by default. Set this key only if you want to use a different Secret.
imagePullPolicy	**cgf**	Sets when Kubernetes pulls images: **Always**, **IfNotPresent**, or **Never**. **Note:** The ECE composable services use the global **imagePullPolicy** setting by default. Set this key only if you want to use a different setting.
partOfLabel	**cgf**	Sets the value to assign to the ECE composable services' **app.kubernetes.io/part-of** label. This label identifies the application that the resource belongs to. **Note:** The ECE composable services use the global **partOfLabel** setting by default. Set this key only if you want to use a different setting.
webserver	**cgf**	The details for the Helidon HTTP server configuration.
chargingGateway.*	**cgf**	The details about the charging-gateway pod. • **name**: The name of the ECE composable services. This is used to identify the ECE composable services in the cluster. The default is **cgf**. • **fullname**: The full name of the ECE composable services. This name will be used in place of the *ReleaseName-Name* pattern. The default is **cgf**. **Note:** The full name must be unique across the sites.

Table 5-3 (Cont.) Charging Gateway Keys

Key	Path in values.yaml	Description
leaderElection.*	**cgf.chargingGateway**	The details for the leader election, which ensures only one instance acts as the leader at a time for certain responsibilities. • **enabled**: Whether the charging-gateway pod uses Kubernetes leader election to coordinate exclusive responsibilities across multiple pod replicas. When set to **true**, only one pod performs leader-only operations, such as running scheduled coordinator tasks or managing shared resources, while other pods remain standby for failover. This prevents duplicate execution and ensures consistent behavior in HA deployments. The default is **true**. • **namePrefix**: The prefix used when creating leader-election Lease names for CGF-managed ECE composable services. The default is **cgf**. • **namespace**: The namespace where leader-election Lease objects are created. Leave it blank to use the release namespace. • **identity**: The explicit leader-election identity for the pod. If left blank, the application uses the host name. • **serviceAccount.create**: Create a dedicated Kubernetes ServiceAccount for leader-election and related CGF permissions. The default is **true**. • **serviceAccount.name**: The existing ServiceAccount name to use instead of creating a new one. • **serviceAccount.annotations**: The annotations to apply to the ServiceAccount. • **rbac.create**: Create the RBAC resources needed for CGF to access and update leader-election Lease objects. The default is **true**. • **rbac.resourceNames**: The fixed Lease resource names to scope RBAC permissions more tightly.

Table 5-3 (Cont.) Charging Gateway Keys

Key	Path in values.yaml	Description
site.*	**cgf.chargingGateway**	The details about the local CGF site. • **siteName**: The name of the local CGF site. Used as the site identity for multisite routing, ownership, and failover behavior. • **remoteSites**: The list of other site names that this deployment can fail over to or coordinate with in a multisite topology. • **failoverMode**: The failover mode. This controls how Kafka consumption and publishing behave during multisite failover, such as whether only the preferred site is active or whether remaining healthy sites can share work. • **kafkaFailoverStabilizationWindow**: The time to wait after a Kafka site change before switching traffic or rebalancing. This helps avoid churn during brief connectivity flaps.
additionalLabels	**cgf.chargingGateway**	The additional labels to add to a Kubernetes custom resource definition (CRD).
annotations	**cgf.chargingGateway**	The additional annotations to add to a Kubernetes CRD.
nodeSelector	**cgf.chargingGateway**	The nodes on which to deploy the charging-gateway pod.
container.image	**cgf.chargingGateway**	The name of the CGF image.
initContainer.image	**cgf.chargingGateway**	The name of the Database Client image.

Table 5-3 (Cont.) Charging Gateway Keys

Key	Path in values.yaml	Description
db.*	**cgf.chargingGateway**	The details for connecting to the CGF cnDBTier database. • **initializationFailTimeout**: The initial wait before retrying a connection to the cnDBTier database after a failure. The default is **10000**. • **connectionTimeout**: The maximum delay between successive connection attempts to the cnDBTier database after a connection is lost. The default is **2000**. • **serviceHostname**: The host name for the CGF database service. If set, this value overrides the global setting. • **servicePort**: The port number for the database service. The default is **3306**. • **adminCredentialsSecretName**: The name of the Kubernetes Secret that holds the database administrator credentials used by the CGF database. If populated, this value overrides the global setting. • **serviceCredentialsSecretName**: The name of the Kubernetes Secret that holds the CGF service user credentials used at runtime to connect to the CGF database. If populated, this value overrides the global setting. • **dbFailoverStabilizationWindow**: The time to wait after a database failover event before resuming normal database access, to help avoid brief failover flaps. The default is **PT0S**. • **siteStatusEndpoints[n].siteName**: The name of the site whose cnDBTier status endpoint is being configured. The default is **site1**. • **siteStatusEndpoints[n].host**: The host name or IP address of the site for cnDBTier status checks. • **siteStatusEndpoints[n].port**: The port used to reach the cnDBTier for status checks. • **siteStatusEndpoints[n].path**: The HTTP path for the cnDBTier real-time cluster status endpoint used to check local cluster health and availability. The default is **/db-tier/status/cluster/local/realtime**.

Table 5-3 (Cont.) Charging Gateway Keys

Key	Path in values.yaml	Description
coordinator.*	**cgf.chargingGateway**	The coordinator can run scheduled tasks on the CDR cnDBTier. • **enabled**: Whether the coordinator pod is enabled. The default is **true**. • **tasks.***: The details about the tasks to run, which include the ability to purge old CDR records and to republish old suspect CDRs. See "Setting Up Coordinator Tasks". • **resources.***: The minimum and maximum amount of memory and CPU that the coordinator can use. See "Setting Up Autoscaling of Pods". • **jvmOpts**: The JVM options to use when starting the coordinator tasks. • **restartCount**: The number of times the coordinator service has been restarted. Increment this value by 1 and run the Helm upgrade command to force a rolling restart of the service. The default is **0**. • **adminService.***: Exposes the coordinator pod's internal admin endpoints for health, metrics, and operational checks using an internal Kubernetes Service. • **messaging.***: The details that the coordinator uses to publish messages to a Kafka topic. See "Setting Up the Coordinator Messaging Service".
worker.messaging.*	**cgf.chargingGateway**	The details for the CGF worker messaging service. See "Setting Up the CGF Worker Messaging Service".
worker.resources.*	**cgf.chargingGateway**	The minimum and maximum amount of memory and CPU that the CGF worker can use. See "Setting Up Autoscaling of Pods".
worker.replicas	**cgf.chargingGateway**	The desired number of pod replicas. The default is **1**. This key is ignored if HPA is enabled.
worker.jvmOpts	**cgf.chargingGateway**	The JVM options to use.
worker.restartCount	**cgf.chargingGateway**	The number of times the CGF worker service has been restarted. Increment this value by 1 and run the Helm upgrade command to force a rolling restart of the service. The default is **0**.

Table 5-3 (Cont.) Charging Gateway Keys

Key	Path in values.yaml	Description
worker.adminService.*	**cgf.chargingGateway**	The details for how the CGF worker exposes an internal admin endpoint: • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **additionalAnnotations**: The additional annotations to add to a Kubernetes CRD. • **type**: 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. This is the default.
terminationGracePeriodSeconds	**cgf.chargingGateway**	The amount of time, in seconds, the 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 after which the processes running in the pod are sent a termination signal. The processes are forcibly terminated. Set this value longer than the expected cleanup time for your process. The default is **60**.
logging.*	**cgf.chargingGateway**	The logging levels for the CGF composable services. See "Using Logging for ECE Composable Services".
tracing.*	**cgf.chargingGateway**	Specifies how to set up tracing for the CGF composable service. See "Tracing the Flow of API Calls".
hpa.*	**cgf.chargingGateway**	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 Pods".
grafanaDashboards.*	**cgf**	The Grafana Dashboard settings. If populated, this value overrides the global setting. See "Monitoring ECE Composable Service Processes".
serviceMonitor.*	**cgf**	The details for the Service Monitor. If populated, this value overrides the global setting. See "Monitoring ECE Composable Service Processes".
prometheusRule.*	**cgf**	The alert modes and threshold rules to run with the Prometheus Operator. See "Configuring Alerting Rules and Thresholds".

Setting Up Coordinator Tasks

The coordinator pod manages scheduled background tasks that operate on the CDR table. These tasks support data cleanup and recovery workflows to ensure data consistency and

reliability in distributed deployments. You can configure the coordinator to run one, both, or none of the available scheduled tasks, depending on your system requirements.

- The **closedCdrRemoval** task removes CDR records that meet the following conditions:
  - The record is older than a configured duration (ageOfRecord)
  - The record has one of the following statuses:
    - **PUBLISHED**: Fully published (**cdrPublishStatus** is set to **PUBLISHED** in the CDR table)
    - **INVALID**: Invalidated during ISN merges (**cdrState** is set to **INVALID** in the CDR table)
    - **SUSPEND**: Published with durable retry (**cdrPublishStatus** is set to **SUSPEND** in the CDR table)

  This task does not remove CDRs immediately after publishing. This ensures that duplicate terminate CDRs received after session closure can still be de-duplicated and rejected. Records with the **SUSPECT** CDR publish status are intentionally preserved to allow proper reconciliation before publication.
- The **suspectCdrRetry** task handles the reprocessing of incomplete or inconsistent CDRs. It republishes suspect CDRs that are older than a specified amount of time.

  In a multisite deployment, when the CGF receives a CDR Terminate event, it performs the following steps:

  1. Scans previously processed ISNs from the database
  2. Detects gaps in aggregated ISNs
  3. Sets the CDR state to **CLOSED** if gaps are not detected
  4. Marks the CDR publish status as **SUSPECT** if gaps are detected

> **Note**
>
> Both tasks run as cron-scheduled Helidon tasks. Only one coordinator instance should be running at a time. This is enforced by deploying the coordinator as a singleton pod using the Helm chart.

To configure coordinator tasks, define the following structure in your configuration file:

```
coordinator:
  tasks:
    scheduledTaskName:
      enabled: true
      cronSchedule: "0 0/5 * * * ?"
      config:
        ageOfRecord: PT5M
        batchSize: 10000
```

Table 5-4 describes each key.

Table 5-4 Coordinator Task Keys

Key	Description
scheduledTaskName	The name of the task to configure: **closedCdrRemoval** or **suspectCdrRetry**.
enabled	Whether to enable the task.
cronSchedule	The frequency at which to automatically run the task. The default value specifies to run every 5 minutes on the hour, every day of the week.
config.ageOfRecord	The length of time a record has remained in the CDR table in ISO 8601 duration format. After that amount of time, the task performs an action on the record. The default value is **PT5M**, which means 5 minutes.
config.batchSize	The maximum number of records processed in a single batch.

Setting Up the Coordinator Messaging Service

The Coordinator Messaging Service is a CGF composable service that manages the publication and recovery of CDRs outside the primary event-processing pipeline. The service runs within the coordinator pod and complements the worker-based CDR generation workflow.

The coordinator messaging service performs the following functions:

- Republishes CDRs that were not delivered successfully
- Supports retry and recovery workflows, such as recovery from transient Kafka failures
- Acts as a centralized publisher for coordinator-managed CDR workflows
- Ensures consistent behavior in high availability (HA) deployments

To configure the coordinator messaging service, enable the coordinator pod and configure the required keys under **cgf.chargingGateway.coordinator**:

> **Note**
>
> The coordinator topic name for the ProcessSuspectCdr service must match the topic name for the ProcessNchfCdr service.

```
messaging:
  services:
    - name: "ProcessSuspectCdr"
      type: KAFKA_CLOUDEVENT_CGF_REQUEST
      enabled: true
      multiSite: *nchf-cdr-multisite
      config:
        subscribers: [ ]
      processors:
        - name: cdrPublisher
          type: KAFKA_PUBLISHER_PROCESSOR
          producerOnly: false
          config:
            publisher:
              name: cgf-cdr
              topic:
                name: cgf-cdr-topic
                numberOfPartitions: 2
```

```
  replicationFactor: 2
  create:
     enabled: true
  modification:
     enabled: true
keyGenerations:
  - criteria: $.payload.chargingDataRequest
    keyPaths:
      - $.payload.chargingDataRequest.subscriberIdentifier
broker:
   name: ccs-broker
   auth:
      enabled: true
      tls:
         trustStore:
            existingSecret: cgf-cluster-ca-cert
            passwordKey: ca.password
         keyStore:
            existingSecret: cgf-user
            passwordKey: user.password
   bootstrapServers: kafka:9093
   extraConfig:
      reconnect.backoff.ms: "175"
      reconnect.backoff.max.ms: "1000"
```

Table 5-5 describes each key under **cgf.chargingGateway.coordinator**.

Table 5-5 Coordinator Messaging Service Keys

Key	Path in values.yaml	Description
name	**messaging.services[]**	The name of the coordinator messaging service that publishes suspect CDRs. The default is **ProcessSuspectCdr**.
type	**messaging.services[]**	The service type that enables the CGF Kafka CloudEvent processing framework for this publishing flow. The default is **KAFKA_CLOUDEVENT_CGF_REQUEST**.
enabled	**messaging.services[]**	Whether to enable this messaging service. The default is **true**.
multisite	**messaging.services[]**	The configuration details for multisite processing and failover behavior for the coordinator messaging service.
processors.name	**messaging.services[]**	The name of the processor that publishes the CDR to Kafka. The default is **cdrPublisher**.
processors.type	**messaging.services[]**	The processor type that sends processed records to an outbound Kafka topic. The default is **KAFKA_PUBLISHER_PROCESSOR**.
processors.producerOnly	**messaging.services[]**	Whether this processor can participate in the service chain and is not limited to producer-only use. The default is **false**.
publisher.name	**messaging.services[].processors.config**	The logical name of the Kafka publisher used by this service. The default is **cgf-cdr**.

Table 5-5 (Cont.) Coordinator Messaging Service Keys

Key	Path in values.yaml	Description
topic.name	**messaging.services[].processors.config.publisher**	The name of the outbound Kafka topic that receives the published CDRs. The default is **cgf-cdr-topic**. **Note:** The topic name for the ProcessSuspectCdr service must match the topic name for the ProcessNchfCdr service.
topic.numberOfPartitions	**messaging.services[].processors.config.publisher**	The number of partitions to create or use for the outbound topic. The default is **2**.
topic.replicationFactor	**messaging.services[].processors.config.publisher**	The replication factor for the outbound Kafka topic. The default is **2**.
topic.create.enabled	**messaging.services[].processors.config.publisher**	Whether the processor can automatically create topics if one does not already exist. The default is **true**.
topic.modification.enabled	**messaging.services[].processors.config.publisher**	Whether to update the topic when the configured topic settings differ from the existing one. The default is **true**.
keyGenerations[].criteria	**messaging.services[].processors.config.publisher**	The JSONPath expression used to select the source data for message key generation.
keyGenerations[].keyPaths[]	**messaging.services[].processors.config.publisher**	The JSONPath expression that identifies the field used as the Kafka message key.
broker.name	**messaging.services[].processors.config.publisher**	The logical name of the Kafka broker configuration used by the publisher. The default is **ccs-broker**.
broker.auth.enabled	**messaging.services[].processors.config.publisher**	Whether to enable broker authentication. The default is **true**.
broker.auth.tls.trustStore.existingSecret	**messaging.services[].processors.config.publisher**	The Kubernetes Secret that contains the Kafka TrustStore certificate. The default is **cgf-cluster-ca-cert**.
broker.auth.tls.trustStore.passwordKey	**messaging.services[].processors.config.publisher**	The Secret key that stores the TrustStore password. The default is **ca.password**.
broker.auth.tls.keyStore.existingSecret	**messaging.services[].processors.config.publisher**	The Kubernetes Secret that contains the client certificate and private key. The default is **cgf-user**.
broker.auth.tls.keyStore.passwordKey	**messaging.services[].processors.config.publisher**	The Secret key that stores the KeyStore password. The default is **user.password**.
broker.bootstrapServers	**messaging.services[].processors.config.publisher**	The Kafka bootstrap server address used by the publisher. The default is **kafka:9093**.
broker.extraConfig.reconnect.backoff.ms	**messaging.services[].processors.config.publisher**	The time in milliseconds to wait before retrying a failed Kafka connection attempt. The default is **175**.
broker.extraConfig.reconnect.backoff.max.ms	**messaging.services[].processors.config.publisher**	The maximum back-off time in milliseconds for Kafka reconnect retries. The default is **1000**.

Setting Up the CGF Worker Messaging Service

You use the ProcessNchfCdr worker messaging service to consume unrated events from the CHF-CGF Kafka topic and generate 5G unrated CDRs for downstream consumption. It performs the following functions:

- Processes CloudEvent payloads
- Applies aggregation logic
- Detects sequencing and duplicate conditions
- Publishes the 5G unrated CDRs to the CGF-CDR Kafka topic

> **Note**
>
> The service supports multisite failover and includes retry and suspend-topic handling for messages that cannot be processed successfully.

By default, the service uses the following processors to perform these tasks, but you can configure the worker messaging service to use additional or different processors:

- LOGGING_PROCESSOR: Writes request and event details to the application logs to help operators monitor processing activity and troubleshoot issues
- AGGREGATOR_PROCESSOR: Applies the unrated CDR aggregation logic
- KAFKA_PUBLISHER_PROCESSOR: Publishes the resulting CDRs to the outbound CGF-CDR Kafka topic

For more information about processors, see "About the ECE Composable Service Processors".

To configure the worker messaging service, set the following keys under **cgf.chargingGateway**:

```
worker:
   messaging:
      services:
         - name: "ProcessNchfCdr"
           type: KAFKA_CLOUDEVENT_CGF_REQUEST
           enabled: true
           multiSite: *nchf-cdr-multisite
           config:
              subscribers:
                 - name: nchf-chf-cdr
                   brokerSite: site1
                   failoverOrder:
                      - site2
                   instanceCount: 1
                   processorRetryPolicy:
                      retryCount: 3
                      retryInterval: "PT0.2S"
                      maxRequeueAttempts: 10
                      retryTopic:
                         name: nchf-cgf-cdr-retry
                      suspendTopic:
                         name: nchf-cgf-cdr-suspend
                   topic:
```

```
            name: *nchfPublisherTopicName
         group: cgf-cdr-0
         broker: *ccs-broker-consumer-1
processors:
   - name: loggingProcessor
     producerOnly: true
     type: LOGGING_PROCESSOR
     config:
        logLevel: INFO
        atMostSeconds: 5

   - name: aggregator
     type: AGGREGATOR_PROCESSOR
     config:
        partialRecordMethod: DEFAULT

   - name: cdrPublisher
     type: KAFKA_PUBLISHER_PROCESSOR
     durability:
        enabled: true
        processRetryEnabled: true
        batchSize: 1000
        schedulerFrequencyInSec: 5
        delayInSec: 10
     producerOnly: false
     config:
        publisher:
           name: cgf-cdr
           topic:
              name: cgf-cdr-topic
              numberOfPartitions: 2
              replicationFactor: 2
              create:
                 enabled: true
              modification:
                 enabled: true
           keyGenerations:
              - criteria: $.payload.chargingDataRequest
                keyPaths:
                   - $.payload.chargingDataRequest.subscriberIdentifier
           broker: *ccs-broker-producer-1
```

Table 5-6 describes each key.

> **Note**
>
> Configure only one subscriber per broker site within a messaging service. In a single-site deployment, configure one subscriber for the local broker site only. In a multisite deployment, configure one subscriber for each participating broker site.

Table 5-6 Worker Messaging Service Keys

Key	Path in values.yaml	Description
services[n].*	**worker.messaging**	The details for the CGF request-processing service for NCHF charging data. • **name**: The name of the worker messaging service that processes unrated 5G charging events. The default is **ProcessNchfCdr**. • **type**: The messaging service type used for Kafka-based CloudEvent CGF request processing. The default is **KAFKA_CLOUDEVENT_CGF_REQUEST**. • **enabled**: Whether to enable the worker messaging service. The default is **true**. • **multiSite**: The configuration details for multisite processing and failover behavior for the worker messaging service.

Table 5-6 (Cont.) Worker Messaging Service Keys

Key	Path in values.yaml	Description
subscribers[n].*	**worker.messaging.services[n].config**	The details for how the CGF subscribes to and consumes charging requests from Kafka. • **name**: The name of the Kafka subscriber configuration. The default is **nchf-chf-cdr**. **Note:** Each subscriber name must be unique within a site and across all sites so CGF can identify subscriber configurations correctly. • **brokerSite**: The preferred site that actively consumes messages for this subscriber. The default is **site1**. • **failoverOrder**: The ordered list of remote sites that can take over message consumption during failover conditions. • **instanceCount**: The number of Kafka consumer instances created for this subscriber. The default is **1**. • **processorRetryPolicy.retryCount**: The number of retry attempts for failed message processing operations. The default is **3**. • **processorRetryPolicy.retryInterval**: The interval between retry attempts. The default is **PT0.2S**. • **processorRetryPolicy.maxRequeueAttempts**: The maximum number of times a failed message can be requeued before suspension. The default is **10**. • **processorRetryPolicy.retryTopic.name**: The name of the Kafka topic used for retrying failed message processing operations. The default is **nchf-cgf-cdr-retry**. • **processorRetryPolicy.suspendTopic.name**: The name of the Kafka topic used for suspended messages that cannot be processed successfully. The default is **nchf-cgf-cdr-suspend**. • **topic.name**: The name of the CHF-CGF Kafka topic from which the service consumes unrated events. **Note:** The topic name for the ProcessNchfCdr service must match the topic name for the ProcessSuspectCdr service. • **group**: The Kafka consumer group name used by the subscriber. The default is **cgf-cdr-0**. • **broker**: The Kafka broker configuration used by the subscriber to consume messages.

Table 5-6 (Cont.) Worker Messaging Service Keys

Key	Path in values.yaml	Description
processors[n].* (Logging Processor)	**worker.messaging.services[n]**	The details for the logging processor. • **name**: The name of the processor in the processing chain. The default for the logging processor is **loggingProcessor**. • **producerOnly**: Whether the processor is restricted to producer-only behavior. The default for the logging processor is **true**. • **type**: The processor type used in the processing pipeline. The default for the logging process is **LOGGING_PROCESSOR**. • **config.logLevel**: The logging level used by the logging processor. The default is **INFO**. • **config.atMostSeconds**: The maximum interval, in seconds, between log messages generated by the logging processor. The default is **5**.
processors[n].* (Aggregator Processor)	**worker.messaging.services[n]**	The details for the aggregator processor. • **name**: The name of the processor in the processing chain. The default for the aggregator processor is **aggregator**. • **type**: The processor type used in the aggregator pipeline. The default for the aggregator processor is **AGGREGATOR_PROCESSOR**. • **config.partialRecordMethod**: The aggregation mode used for unrated CDR processing: – **DEFAULT**: The CGF aggregates charging requests that belong to the same session and generates aggregated CDRs. This is the default. – **INDIVIDUAL**: The CGF does not aggregate charging requests. The CDR for each operation type in a given session is published immediately after completion of the aggregator processor.

Table 5-6 (Cont.) Worker Messaging Service Keys

Key	Path in values.yaml	Description
processors[n].* (CDR Publisher Processor)	**worker.messaging.services[n]**	The details for the CDR publisher processor. • **name**: The name of the processor in the processing chain. The default for the CDR publishing processor is **cdrPublisher**. • **type**: The processor type used in the CDR publishing pipeline. The default for the CDR publishing processor is **KAFKA_PUBLISHER_PROCESSOR**. • **durability.enabled**: Whether to enable durable retry persistence for failed Kafka publish operations. The default is **true**. • **durability.processRetryEnabled**: Whether to enable retry processing for durable publisher failures. The default is **true**. • **durability.batchSize**: The number of retry records processed in each retry batch. The default is **1000**. • **durability.schedulerFrequencyInSec**: The interval, in seconds, at which the retry scheduler runs. The default is **5**. • **durability.delayInSec**: The delay, in seconds, before retry processing begins for failed publish operations. The default is **10**. • **producerOnly**: Whether the processor is restricted to producer-only behavior. The default is **false**. • **config.publisher.name**: The logical name of the Kafka publisher. The default is **cgf-cdr**. • **publisher.topic.name**: The name of the outbound Kafka topic used to publish CDRs. The default is **cgf-cdr-topic**. • **publisher.topic.numberOfPartitions**: The number of Kafka partitions for the outbound topic. The default is **2**. • **publisher.topic.replicationFactor**: The Kafka replication factor for the outbound topic. The default is **2**. • **publisher.topic.create.enabled**: Whether to enable automatic creation of the outbound Kafka topic. The default is **true**. • **publisher.topic.modification.enabled**: Whether to enable automatic modification of existing Kafka topic settings. The default is **true**. • **publisher.keyGenerations[n].criteria**: The JSONPath criteria used to determine when message key generation applies. The default is **$.payload.chargingDataRequest**. • **publisher.keyGenerations[n].keyPaths[n]**: The JSONPath expression used to generate the Kafka message key. The default is **$.payload.chargingDataRequest.subscriberIdentifier**.

Table 5-6 (Cont.) Worker Messaging Service Keys

Key	Path in values.yaml	Description
		• **publisher.broker**: The Kafka broker configuration used to publish outbound CDRs.

6

Integrating ECE CHF Composable Service with ECE for 5G Charging

You can integrate the Oracle Communications Elastic Charging Engine (ECE) Charging Manager (CHF) composable service with your existing ECE cloud native deployment to perform 5G charging.

Topics in this document:

About Integrating the ECE CHF Composable Service into ECE for 5G Charging

When the ECE Charging Manager (CHF) composable service is integrated with an existing ECE deployment, the composable services work together to provide real-time 5G charging, quota management, spending limit control, and charging notification processing for 5G services.

The CHF provides the public 3GPP Nchf service interfaces used by 5G network functions, such as the Session Management Function (SMF) and Policy Control Function (PCF). The CHF acts as the network-facing charging gateway and mediation layer between the 5G core network and ECE.

The ECE Elastic Charging Server (ECS) acts as the primary charging engine and is responsible for subscriber charging sessions, balance reservation and consumption, real-time rating, policy evaluation, and spending limit enforcement. ECS also generates rated charging events and charging-related notifications as part of the charging workflow.

This architecture enables scalable, cloud native 5G charging while separating the network-facing charging interfaces provided by the CHF from the real-time charging, balance management, and policy processing capabilities provided by ECS.

ECE and the CHF work together to generate 5G unrated and rated CDRs through the following workflow:

1. The CHF converts the request into an unrated event and sends it to the CHF-CGF Kafka topic. The Charging Gateway Function (CGF) consumes the unrated event from the Kafka topic and generates unrated 5G call detail records (CDRs).

> **Note**
>
> The CHF always sends unrated events to the CGF to generate unrated 5G CDRs, even when the charging request is quota managed and also requires rating. This behavior supports 3GPP charging and charging-record generation requirements.

2. The CHF determines whether a request requires online charging processing by checking for quota-management information, such as requested units or online-charging indicators.
3. If the request requires online charging, the CHF forwards the request to the ECE HTTP Gateway.

   For active online charging sessions, the CHF also routes session termination requests to the HTTP Gateway, even when quota-related information is no longer present in the request. This behavior helps prevent orphaned sessions and ensures the proper release of reserved resources.
4. The HTTP Gateway forwards charging requests to ECS for rating and charging processing.
5. ECS generates rated events and charging-related notifications as part of the charging workflow.
6. ECS delivers charging and spending limit notifications asynchronously through Kafka-based notification distribution.
7. The HTTP Gateway consumes the notification events from Kafka and delivers notifications to external network functions by using the notification URIs associated with the charging or spending-limit sessions.

High-Level Configuration Tasks to Use CHF for 5G Charging

To configure the ECE CHF composable service and your existing ECE cloud native deployment for 5G charging:

1. Ensure you have installed and configured all prerequisite software, such as Kafka server.

   See "Setting Up Prerequisite Software".
2. Create an **override-values.yaml** file for the ECE composable services Helm chart.
3. Configure the mandatory values for rating 5G events.

   See "Configuring Mandatory Values for 5G Charging".
4. Configure the CHF composable service to process charging and SLC requests and route them appropriately.

   See "Configuring the CHF Composable Service for 5G Charging".
5. Configure the NRF Management Agent to register the CHF with the 5G NRF.

   See "Configuring the NRF Management Agent".
6. Migrate the CHF and CGF composable services into your existing ECE deployment.

   See "Migrating the CHF and CGF Composable Services into an Existing ECE System".

Configuring Mandatory Values for 5G Charging

To configure the ECE composable services to perform real-time charging, quota management, and spending limit control for 5G services, set the following keys in your **override-values.yaml** file for **oc-ccs-*version***. These keys are required to create a simple or demonstration version that you can use immediately.

These keys apply to all ECE composable services, but you can override them for a specific composable service by setting a different value in the composable service's key. For example, to use different Service Monitor settings for the NRF Management Agent, set the keys under **nchfConvergedCharging.nchfConvergedCharging.nrfManagementAgent.serviceMonitor**.

> **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 ""
imagePullSecrets: &imagePullSecrets
imagePullPolicy: &imagePullPolicy IfNotPresent
nodeSelector: &nodeSelector {}

partOfLabel: &partOfLabel oc-ccs
additionalLabels: &additionalLabels { }
annotations: &annotations { }

webserverDefaults: &webserver
  maxPayloadSize: 2000000

nrfManagementImage: &nrfManagementImage "ccs/nrf-management:15.2.0.0.2"
nchfConvergedChargingImage: &nchfConvergedChargingImage "ccs/chf/nchf-converged-
charging:15.2.0.0.2"

tracingEnabled: &tracingEnabled false
tracingHost: &tracingHost otlp-collector-host
tracingPort: &tracingPort 14250

nchfTopicName: &nchfPublisherTopicName nchf-cgf-cdr

nchfCdrMultiSite: &nchf-cdr-multisite
  enabled: false
  coordinationId: "nchf-cdr-coordination"
  failoverMode: ALL_REMAINING
  kafkaFailoverStabilizationWindow: "PT0S"
  remoteSites:
    - site2
    - site3

kafka-brokers:
  ccs-broker-producer-1: &ccs-broker-producer-1
    name: ccs-broker-producer
    bootstrapServers: kafka:9093
    auth:
      enabled: true
      tls:
        trustStore:
          existingSecret: cgf-billing-cluster-ca-cert
          passwordKey: ca.password
        keyStore:
          existingSecret: cgf-billing-user
          passwordKey: user.password
    extraConfig:
      reconnect.backoff.ms: "100"
      reconnect.backoff.max.ms: "1000"
      max.block.ms: "30000"
      request.timeout.ms: "15000"
      delivery.timeout.ms: "60000"
  ccs-broker-consumer-1: &ccs-broker-consumer-1
    name: ccs-broker-consumer
    bootstrapServers: kafka:9093
```

```
    auth:
      enabled: true
      tls:
        trustStore:
          existingSecret: cgf-cluster-ca-cert
          passwordKey: ca.password
        keyStore:
          existingSecret: nchf-user
          passwordKey: user.password
    extraConfig:
      auto.offset.reset: latest
      reconnect.backoff.ms: "175"
      reconnect.backoff.max.ms: "1000"
      allow.auto.create.topics: "false"

serviceMonitor: &serviceMonitor
  enabled: false
  namespace:
  additionalLabels: {}

grafanaDashboards: &grafanaDashboards
  enabled: false
  grafanaNamespace:
  labels:
    grafana_dashboard: "1"
  annotations:
    k8s-sidecar-target-directory: "/tmp/dashboards/ccs"
```

Table 6-1 describes each key.

Table 6-1 Mandatory Keys for 5G Charging

Key	Description
imageRepository	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	The image pull secret or list of image pull secrets used by pods to authenticate to the private image registry. 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	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.
nodeSelector	The nodes on which to deploy ECE composable services. Labels appear underneath **nodeSelector** in a set of key-value pairs.
partOfLabel	The value to assign to the composable service's **app.kubernetes.io/part-of label**. This label identifies the application that the resource belongs to. The default is **oc-ccs**.
additionalLabels	A key-value map of additional Kubernetes labels applied to deployed resources.

Table 6-1 (Cont.) Mandatory Keys for 5G Charging

Key	Description
annotations	A key-value map of Kubernetes annotations applied to deployed resources.
webserverDefaults.maxPayloadSize	The maximum allowed size of an HTTP request body. The default is **2000000**.
nrfManagementImage	The name of the NRF image. The default is **ccs/nrf-management:15.2.0.0.2**.
nchfConvergedChargingImage	The name of the nCHF image. The default is **ccs/chf/nchf-converged-charging:15.2.0.0.2**.
tracingEnabled	Whether tracing through Jaeger is enabled. The default is **false**.
tracingHost	The host name for the Jaeger server. The default is **otlp-collector-host**. **Note:** This key applies only if tracing is enabled.
tracingPort	The port number for Jaeger. The default is **14250**. **Note:** This key applies only if tracing is enabled.
nchfTopicName	The name of the CHF-CGF Kafka topic. The CHF publishes unrated events to this topic. The default is **nchf-cgf-cdr**.
nchfCdrMultiSite.*	The details for deploying CHF on multiple sites. • **enabled**: Whether to deploy CHF on multiple sites. • **coordinationId**: The correlation key used to tie a request or failover decision together across sites. The default is **nchf-cdr-coordination**. • **failoverMode**: The CHF multisite failover mode, which is driven by a site-aware Kafka partition assignment strategy, where local-site consumers have highest priority and automatically take ownership of partitions, with remote sites acting as backups. – **NEXT_SITE**: A single designated backup site takes over all processing when the primary site fails. – **ALL_REMAINING**: All available remote sites share the load during a failure. When the failed site recovers, partition ownership automatically shifts back to the local site, restoring the original active-active balance. This is the default. • **kafkaFailoverStabilizationWindow**: The grace period in ISO 8601 duration format. CHF waits after Kafka looks healthy again before switching consumers back and treating the site as stable. It helps prevent flapping during recovery and makes sure broker and topic health, partition assignment, and consumer groups have settled before normal processing resumes. The default value is **PT0S**, which means there is no waiting period. • **remoteSites**: The list of remote site names. The default is **site2** and **site3**.

Table 6-1 (Cont.) Mandatory Keys for 5G Charging

Key	Description
kafka-brokers.ccs-broker-producer-1	The details for connecting the producer to the Kafka Server: • **name**: The name of the Kafka broker producer. The default is **ccs-broker-producer**. • **bootstrapServers**: The host name and port pair for the Apache Kafka cluster. The default is **kafka:9093**. • **auth.enabled**: Whether the Kafka client is configured to use authenticated TLS for its broker connection (**true**) or not (**false**). The default is **true**. • **auth.tls.trustStore.existingSecret**: The name of the Kubernetes Secret that contains the Certificate Authority (CA) file. The default is **cgf-billing-cluster-ca-cert**. • **auth.tls.trustStore.passwordKey**: The data key inside the Secret that contains the TrustStore password. The default is **ca.password**. • **auth.tls.keyStore.existingSecret**: The name of the Kubernetes Secret that contains the client identity KeyStore and its password. The default is **cgf-billing-user**. • **auth.tls.keyStore.passwordKey**: The data key inside the Secret that holds the KeyStore password. The default is **user.password**. • **extraConfig.reconnect.backoff.ms**: The initial wait, in milliseconds, before retrying a connection to a broker after a failure. The default is **100**. • **extraConfig.reconnect.backoff.max.ms**: The maximum delay between successive TCP connection attempts to a broker after a connection is lost or fails. The default is **1000** (1 second). • **extraConfig.max.block.ms**: The Kafka client timeout limiting how long a producer or admin client blocks while waiting for resources such as cluster metadata, buffer space, or a connection to the broker. The default is **30000**. • **extraConfig.request.timeout.ms**: The amount of time the Kafka client waits for a response to a request before retrying or failing the operation once retries are exhausted. The default is **15000**. • **extraConfig.delivery.timeout.ms**: The Kafka producer's overall upper bound on how long a record can take from send until Kafka reports success or failure. The default is **60000**.

Table 6-1 (Cont.) Mandatory Keys for 5G Charging

Key	Description
kafka-brokers.ccs-broker-consumer-1	The Kafka consumer configuration used by the CGF to consume unrated events from CHF. • **name**: The name of the Kafka broker consumer. The default is **ccs-broker-consumer**. • **bootstrapServers**: The host name and port pair for the Apache Kafka cluster. The default is **kafka:9093**. • **auth.enabled**: Whether the Kafka client is configured to use authenticated TLS for its broker connection (**true**) or not (**false**). The default is **true**. • **auth.tls.trustStore.existingSecret**: The name of the Kubernetes Secret that contains the Certificate Authority (CA) file. The default is **cgf-cluster-ca-cert**. • **auth.tls.trustStore.passwordKey**: The data key inside the Secret that contains the TrustStore password. The default is **ca.password**. • **auth.tls.keyStore.existingSecret**: The name of the Kubernetes Secret that contains the client identity KeyStore and its password. The default is **nchf-user**. • **auth.tls.keyStore.passwordKey**: The data key inside the Secret that holds the KeyStore password. The default is **user.password**. • **extraConfig.auto.offset.reset**: Where to start reading the log when a consumer group has not committed an offset for a partition: **latest**, **earliest**, or **none**. The default is **latest**, which means it reads only new records. • **extraConfig.reconnect.backoff.ms**: The initial wait, in milliseconds, before retrying a connection to a broker after a failure. The default is **175**. • **extraConfig.reconnect.backoff.max.ms**: The maximum delay between successive TCP connection attempts to a broker after a connection is lost or fails. The default is **1000** (1 second). • **extraConfig.allow.auto.create.topics**: Whether the client can trigger automatic creation of a topic on the broker when it references a nonexistent topic (**true**) or not (**false**). The default is **false**.
serviceMonitor.*	The details for the Service Monitor: • **enabled**: Whether to enable Kubernetes Service Monitor, which is used to monitor a group of services. The default is **false**. • **namespace**: The namespace in which the Service Monitor is deployed. • **additionalLabels**: The labels map used when Service Monitor is deployed. See "Monitoring ECE Composable Service Processes".
grafanaDashboards.*	The details for the Grafana Dashboards: • **enabled**: Whether to enable Grafana Dashboards for Grafana Operator. The default is **false**. • **grafanaNamespace**: The namespace in which Grafana is deployed. • **labels.grafana_dashboard**: The labels to add to Grafana CRDs. This helps Grafana discover the dashboards. The default is **1**. • **annotations.k8s-sidecar-target-directory**: The directory in which the Grafana Dashboards are deployed. The default is **/tmp/dashboards/ccs**. See "Monitoring ECE Composable Service Processes".

Configuring the CHF Composable Service for 5G Charging

When configured for 5G charging, the CHF performs these functions:

- Exposes the 5G Nchf service interfaces for:
  - Converged charging (**nchf-convergedcharging**)
  - Spending limit control (**nchf-spendinglimitcontrol**)
- Processes charging requests originating from SMF and other network functions
- Processes spending limit control subscription and policy requests originating from PCF
- Mediates and transforms incoming requests and outgoing responses
- Routes online charging requests to the ECS for rating, quota reservation and consumption, session lifecycle management, balance management, and spending limit enforcement

To configure the CHF pod to process 5G charging requests, set the following keys in your **override-values.yaml** file:

```
nchfConvergedCharging:
  enabled: true
  imageRepository: *imageRepository
  imagePullSecrets: *imagePullSecrets
  imagePullPolicy: *imagePullPolicy
  partOfLabel: *partOfLabel
  webserver: *webserver
  nchfConvergedCharging:
    name: nchf-converged-charging
    fullname: "nchf-converged-charging"
    additionalLabels: *additionalLabels
    annotations: *annotations
    nodeSelector: *nodeSelector
    container:
      image: *nchfConvergedChargingImage
    mutatorRulesConfigMapOverride:
    restServices:
      - name: "ConvergedCharging"
        type: NCHF_CONVERGED_CHARGING
        config:
          path: "/nchf-convergedcharging/v3/chargingdata"
          ingressOrigin:
        processors:
          - name: mediation-processor
            type: COMPOSITE_PROCESSOR
            processors:
              - name: mutation-processor-standin
                producerOnly: true
                type: LOGGING_PROCESSOR
                config:
                  logLevel: INFO
                  atMostSeconds: 10
          - name: nchf-processor
            type: NCHF
            config:
              subscriberValidator:
                enabled: false
                config:
                  baseUri: "http://subscriber-profile-repository:8080/
subscriberProfileRepository/v1"
              sessionManager:
```

```
                    enabled: false
                    config:
                      baseUri: "http://session-manager:8080/sessionManager/v1"
                      expiryIntervalInSeconds: 300
                processors:
                  - name: transform-nchf-request
                    type: MUTATOR_PROCESSOR
                    config:
                      rules:
                        - source: INLINE
                          rule:
                            ruleName: onlineSessionIdEnricher
                            criterion:
                              expression:
                                nchfRequest:
                                  criteriaType: MVEL_EXPRESSION
                                  expression: "value.requestType == RequestType.CREATE &&
value.isOnlineChargingRequest()"
                            mutation:
                              expression:
                                associatedSession:
                                  - mutationType: MVEL_TRANSFORM
                                    expression: "value.id = 'online_'+value.id"
                  - name: cdrPublisher
                    type: KAFKA_PUBLISHER_PROCESSOR
                    config:
                      publisher:
                        name: nchf
                        topic:
                          name: *nchfPublisherTopicName
                          numberOfPartitions: 2
                          replicationFactor: 2
                          create:
                            enabled: true
                          modification:
                            enabled: true
                        keyGenerations:
                          - criteria: $.subscriberIdentifier
                            keyPaths:
                              - $.subscriberIdentifier
                        broker: *ccs-broker-producer-1
                  - name: rest-nchf-httpgw
                    type: NCHF_REST_PROCESSOR
                    config:
                      host: "httpgw"
                      port: "8080"
                    faultTolerance:
                      retry:
                        delay: 100
                        calls: 3
                        delayFactor: 1.0
                      circuitBreaker:
                        delay: 200
                        errorRatio: 30
                        successThreshold: 2
                        volume: 10
                    criteria:
                      criterion:
                        expression:
                          nchfRequest:
                            criteriaType: MVEL_EXPRESSION
                            expression: "(value.associatedSession.id != null &&
```

```
value.associatedSession.id.startsWith('online_')) || value.isOnlineChargingRequest()"
      - name: "SpendingLimitControl"
        type: NCHF_SPENDING_LIMIT_CONTROL
        config:
          path: "/nchf-spendinglimitcontrol/v1/subscriptions"
        processors:
          - name: slc-logger
            producerOnly: true
            type: LOGGING_PROCESSOR
            config:
              logLevel: INFO
              atMostSeconds: 10
          - name: rest-slc-httpgw
            type: REST_PROCESSOR
            config:
              uri: "http://httpgw:8080/nchf-spendinglimitcontrol/v1/subscriptions"
            faultTolerance:
              retry:
                delay: 100
                calls: 3
                delayFactor: 1.0
              circuitBreaker:
                delay: 200
                errorRatio: 30
                successThreshold: 2
                volume: 10

    networkFunction:
      nfProfile: &profileFile nfProfile.json
      existingNfProfileConfigMap: &existingNfProfileConfigMap

    terminationGracePeriodSeconds: 60
    resources:
      memoryRequest: "256Mi"
      cpuRequest: "250m"
      memoryLimit: "512Mi"
      cpuLimit: "500m"
    replicas: 1
    jvmOpts: "-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:InitialRAMPercentage=80 -
XX:MaxRAMPercentage=80 -XX:MinRAMPercentage=80 -Dlog4j.configurationFile=/app/config/
log4j2.yaml"
    restartCount: 0
    service:
      additionalLabels: *additionalLabels
      additionalAnnotations: *annotations
      type: ClusterIP
    adminService:
      additionalLabels: *additionalLabels
      additionalAnnotations: *annotations
      type: ClusterIP
    logging:
      format:
        type: TEXT
        pattern: *loggingPattern
      rootLevel: INFO
      packageLogging:
        - name: com.oracle.cagbu
          level: INFO
        - name: io.helidon
          level: INFO
    tracing:
      enabled: *tracingEnabled
```

```
    service: nchf-converged-charging
    maxQueueSize: 42
    flushIntervalMs: 10001
    host: *tracingHost
    port: *tracingPort
    samplerType: ratio
    samplerParam: 0.01
    samplerManager:
    logSpans: true
    webServer:
      spans:
        httpRequest:
          contentWrite: false
          contentRead: false
  hpa:
    enabled: true
    minReplicas: 1
    maxReplicas: 8
    metrics:
      cpuAverageUtilization: 65
    scaleDown:
      selectPolicy: Max
      stabilizationWindowSeconds: 300
      periodSeconds: 180
    scaleUp:
      selectPolicy: Max
      stabilizationWindowSeconds: 5
      periodSeconds: 20

grafanaDashboards: *grafanaDashboards
serviceMonitor: *serviceMonitor

ingress:
  ingressClassName:
```

Table 6-2 describes each key.

Table 6-2 Charging Manager Keys

Key	Path in values.yaml	Description
enabled	**nchfConvergedCharging**	Whether the CHF converged charging service is enabled. The default is **true**.
imageRepository	**nchfConvergedCharging**	The registry server where you pushed images, typically in this format: *RepoHost*:*RepoPort*. **Note:** The composable service uses the global **imageRepository** setting by default. Set this key only if you want to use a different registry server.
imagePullSecrets	**nchfConvergedCharging**	The name of the Secret that contains credentials for accessing images from your private image server. **Note:** The composable service uses the global **imagePullSecrets** setting by default. Set this key only if you want to use a different Secret.
imagePullPolicy	**nchfConvergedCharging**	The Kubernetes image pull policy for CHF pods: **Always**, **IfNotPresent**, or **Never**. **Note:** The composable service uses the global **imagePullPolicy** setting by default. Set this key only if you want to use a different setting.

Table 6-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
partOfLabel	**nchfConvergedCharging**	Sets the value to assign to the composable service's **app.kubernetes.io/part-of** label. This label identifies the application that the resource belongs to. **Note:** The composable service uses the global **partOfLabel** setting by default. Set this key only if you want to use a different setting.
webserver	**nchfConvergedCharging**	The Helidon HTTP server configuration.
nchfConvergedCharging.*	**nchfConvergedCharging**	The details about the CHF. • **name**: The name of the pod. This is used to identify the pod in the cluster. The default is **nchf-converged-charging**. • **fullname**: The full Kubernetes resource name prefix for the composable service. This name is used in place of the *ReleaseName-Name* pattern. The default is **nchf-converged-charging**. **Note:** The full name must be unique across the sites. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **annotations**: The additional annotations to add to a Kubernetes CRD. • **nodeSelector**: The nodes on which to deploy the CHF composable service. • **container.image**: The name of the CHF image.
mutatorRulesConfigMapOverride	**nchfConvergedCharging**	The path to a different ConfigMap for mutation rules. This lets you override the standard transformation rules used by the mutation processor. For example, you could use it to: • Adjust request mapping for a particular environment • Change how online charging requests are enriched • Update mutation logic without rebuilding the service

Table 6-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
restServices[0].* (Converged Charging Service)	**nchfConvergedCharging.nchfConvergedCharging**	The details about the Converged Charging Service. • **name**: The name of the Converged Charging REST service that processes charging requests and generates unrated CDR output. • **type**: Set this to **NCHF_CONVERGED_CHARGING**. This is the service type for the Converged Charging request flow. • **config.path**: The REST endpoint used to receive charging requests. The default is **/nchf-convergedcharging/v3/chargingdata**. • **config.ingressOrigin**: The external origin used when constructing absolute response URLs, such as the HTTP Location header. This value should include only the scheme, host, and optional port. • **processors.***: This is the list of processors to run in the Converged Charging Service. See "Configuring Processors for the Converged Charging Service".
restServices[0].* (Spending Limit Control Service)	**nchfConvergedCharging.nchfConvergedCharging**	The details about the Spending Limit Control Service. • **name**: The name of the SLC REST service. The default is **SpendingLimitControl**. • **type**: Set this to **NCHF_SPENDING_LIMIT_CONTROL**. This is the service type for the SLC request flow. • **config.path**: The REST endpoint used to receive SLC requests. The default is **/nchf-spendinglimitcontrol/v1/subscriptions**. • **processors.***: The details for the SLC Service. See "Configuring Processors for the Spending Limit Control Service".
networkFunction.*	**nchfConvergedCharging.nchfConvergedCharging**	The details for how the service gets its NF profile information for NRF registration. • **nfProfile**: The NF profile JSON file that contains the network function's registration metadata, such as its identity, services, endpoints, and capabilities. • **existingNfProfileConfigMap**: The Kubernetes source that contains the NF profile.

Table 6-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
terminationGracePeriodSeconds	**nchfConvergedCharging.nchfConvergedCharging**	The amount of time, in seconds, the 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 after which the processes running in the pod are sent a termination signal. The processes are forcibly terminated. Set this value longer than the expected cleanup time for your process. The default is **60**.
resources.*	**nchfConvergedCharging.nchfConvergedCharging**	The minimum and maximum amount of memory and CPU that the CHF composable service can use. See "Setting Up Autoscaling of Pods".
replicas	**nchfConvergedCharging.nchfConvergedCharging**	The desired number of pod replicas. The default is **1**. This key is ignored if HPA is enabled.
jvmOpts	**nchfConvergedCharging.nchfConvergedCharging**	The JVM options to use.
restartCount	**nchfConvergedCharging.nchfConvergedCharging**	The number of times the CHF service has been restarted. Increment this value by 1 and run the Helm upgrade command to force a rolling restart of the CHF service. The default is **0**.
service.*	**nchfConvergedCharging.nchfConvergedCharging**	The details of the CHF service. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **additionalAnnotations**: The additional annotations to add to a Kubernetes CRD. • **type**: 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. This is the default.
adminService.*	**nchfConvergedCharging.nchfConvergedCharging**	The details of the admin service. • **additionalLabels**: The additional labels to add to a Kubernetes custom resource definition (CRD). • **additionalAnnotations**: The additional annotations to add to a Kubernetes CRD. • **type**: 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. This is the default.

Table 6-2 (Cont.) Charging Manager Keys

Key	Path in values.yaml	Description
logging.*	**nchfConvergedCharging.nchfConvergedCharging**	The logging levels for the CHF composable service. See "Using Logging for ECE Composable Services".
tracing.*	**nchfConvergedCharging.nchfConvergedCharging**	Specifies how to set up tracing for the CHF composable service. See "Tracing the Flow of API Calls".
hpa.*	**nchfConvergedCharging.nchfConvergedCharging**	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 Pods".
grafanaDashboards.*	**nchfConvergedCharging.nchfConvergedCharging**	The details for the CHF Grafana Dashboards. See "Monitoring ECE Composable Service Processes".
serviceMonitor.*	**nchfConvergedCharging.nchfConvergedCharging**	The details for the CHF Service Monitor. See "Monitoring ECE Composable Service Processes".
ingress.ingressClassName	**nchfConvergedCharging.nchfConvergedCharging**	The Kubernetes Ingress class used for the CHF deployment when ingress is enabled. It determines which ingress controller manages external access to the CHF service.
nrfManagementAgent	**nchfConvergedCharging.nchfConvergedCharging**	The details for the NRF Management Agent. See "Configuring the NRF Management Agent".

Configuring Processors for the Converged Charging Service

The Converged Charging service provides the 5G Nchf converged charging interface, which is used by network functions to submit charging requests to the CHF. This service processes charging data requests, evaluates whether online charging is required, publishes usage events for unrated CDR generation, and routes eligible requests to ECS for rating and quota management.

The Converged Charging service exposes the standard 3GPP Nchf charging endpoint and uses a configurable processor framework to perform request mediation, transformation, Kafka publication, and ECE integration.

By default, the Converged Charging service processes requests by running the following processors in sequence:

1. **mediation-processor**: Performs preliminary mediation and optional logging behavior before charging orchestration begins.
2. **nchf-processor**: Runs CHF-specific charging orchestration and downstream request handling. By default, it also calls the following sub-processors:
    a. **transform-nchf-request**: Applies request transformation and session enrichment logic.
    b. **cdrPublisher**: Publishes usage events to Kafka for unrated 5G CDR generation.

c. **rest-nchf-httpgw**: Routes eligible online charging requests to HTTP Gateway and ECE.

For more information about processors, see "About the ECE Composable Service Processors".

Table 6-3 lists the keys for configuring the nchf-processor.

Table 6-3 Converged Charging Service Keys

Key	Path in values.yaml	Description
name	**restServices.processors**	The name of the service: **ConvergedCharging**.
type	**restServices.processors**	The service type: **NCHF_CONVERGED_CHARGING**.
processors[].* (mediation-processor)	**restServices.processors**	This processor performs preliminary mediation and optional logging. • **name**: The logical name of the processor: **mutation-processor-standin**. • **producerOnly**: Whether the processor is used only for producing log output. The default is **true**. • **type**: The processor type: **COMPOSITE_PROCESSOR**. • **config.logLevel**: The logging level used by the processor. • **config.atMostSeconds**: The maximum frequency for repeated log messages.

Table 6-3 (Cont.) Converged Charging Service Keys

Key	Path in values.yaml	Description
processors[].* (nchf-processor)	**restServices.processors**	This processor forwards requests to HTTP Gateway and ECS. • **name**: The logical name of the processor: **nchf-processor**. • **type**: The processor type: **NCHF**. • **config.subscriberValidator.enabled**: Whether optional subscriber validation is enabled. The default is **false**. • **config.subscriberValidator.config.baseUri**: The base URL of the subscriber profile repository service. The default is **http://subscriber-profile-repository:8080/subscriberProfileRepository/v1**. • **config.sessionManager.enabled**: Whether session manager integration is enabled. The default is **false**. • **config.sessionManager.config.baseUri**: The base URL of the session manager service. The default is **http://session-manager:8080/sessionManager/v1**. • **config.sessionManager.config.expiryIntervalInSeconds**: The session expiration interval, in seconds, used by the session manager integration. The default is **300**. • **processors**: The list of sub-processors to run in the specified order. The **nchf-processor** can run one or more of these sub-processors: – **transform-nchf-request**: Applies request transformation and session enrichment logic. See "Table 6-4". – **cdrPublisher**: Publishes usage events to Kafka for unrated 5G CDR generation. See "Table 6-5". – **rest-nchf-httpgw**: Routes eligible online charging requests to HTTP Gateway and ECE. See "Table 6-6".

Table 6-4 lists the keys for configuring the **transform-nchf-request** sub-processor to transform the request and enrich sessions.

Table 6-4 transform-nchf-request Sub-processor Keys

Key	Description
name	The name of the processor: **transform-nchf-request**.
type	The processor type: **MUTATOR_PROCESSOR**.
config.rules[].source	The list of configured mutation rules. The default is **INLINE**, which specifies that the mutation rule is defined directly in the configuration file and is applied in place during request processing.

Table 6-4 (Cont.) transform-nchf-request Sub-processor Keys

Key	Description
config.rules[].rule.ruleName	The name of the rule. The default is **onlineSessionIdEnricher**.
config.rules[].rule.criterion.expression.nchfRequest.criteriaType	The expression language used to evaluate the rule condition for an nCHF request. It tells CHF how to interpret the criterion expression and determine whether the mutation rule should be applied during request processing. The default is **MVEL_EXPRESSION**.
config.rules[].rule.criterion.expression.nchfRequest.expression	The condition that CHF evaluates to determine whether the mutation rule applies to an nCHF request.
config.rules[].rule.mutation.expression.associatedSession.mutationType	The transformation type used to modify the associated session data when CHF applies the mutation rule. The default is **MVEL_EXPRESSION**.
config.rules[].rule.mutation.expression.associatedSession.expression	The expression used to modify or enrich the associated session data during request processing.

Table 6-5 lists the keys for configuring the **cdrPublisher** sub-processor to publish usage events to Kafka.

Table 6-5 cdrPublisher Sub-processor Keys

Key	Description
name	The name of the processor: **cdrPublisher**.
type	The processor type: **KAFKA_PUBLISHER_PROCESSOR**.
config.publisher.name	The logical publisher name. The default is **nchf**.
config.publisher.topic.name	The Kafka topic used for event publication.
config.publisher.topic.numberOfPartitions	The number of Kafka topic partitions. The default is **2**.
config.publisher.topic.replicationFactor	The Kafka topic replication factor. The default is **2**.
config.publisher.topic.create.enabled	Whether Kafka topic auto-creation is enabled. The default is **true**.
config.publisher.topic.modification.enabled	Whether Kafka topic modification is enabled. The default is **true**.
config.publisher.keyGenerations	The key-generation rules used for Kafka message partitioning.
config.publisher.broker	The Kafka broker configuration used by the publisher.

Table 6-6 lists the keys for configuring the **rest-nchf-httpgw** sub-processor to route online charging requests to HTTP Gateway and ECS.

Table 6-6 rest-nchf-httpgw Sub-processor Keys

Key	Description
name	The name of the processor: **rest-nchf-httpgw**.
type	The processor type: **NCHF_REST_PROCESSOR**.
config.host	The target HTTP Gateway host name. The default is **httpgw**.
config.port	The target HTTP Gateway port. The default is **8080**.
faultTolerance.retry.delay	The initial retry delay in milliseconds. The default is **100**.
faultTolerance.retry.calls	The maximum number of retry attempts. The default is **3**.
faultTolerance.retry.delayFactor	The retry backoff factor. The default is **1.0**.
faultTolerance.circuitBreaker.delay	The delay before circuit-breaker retry attempts. The default is **200**.
faultTolerance.circuitBreaker.errorRatio	The error ratio threshold used to open the circuit breaker. The default is **30**.
faultTolerance.circuitBreaker.successThreshold	The number of successful calls required to close the circuit breaker. The default is **2**.
faultTolerance.circuitBreaker.volume	The sliding window size used by the circuit breaker. The default is **10**.
criteria.criterion.expression.nchfRequest.criteriaType	The expression used to determine whether the request is routed to ECE for online charging. The default is **MVEL_EXPRESSION**.
criteria.criterion.expression.nchfRequest.expression	The condition CHF evaluates to determine whether the request matches the criteria and should be routed to the associated processor.

Configuring Processors for the Spending Limit Control Service

The Spending Limit Control (SLC) service provides the 5G Nchf spending limit control interface, which is used by network functions to manage subscriber spending limit subscriptions and policy notifications. This service receives spending limit control requests, applies optional mediation and logging behavior, and forwards the requests to HTTP Gateway and ECS for spending limit processing and policy enforcement.

The SLC service exposes the standard 3GPP Nchf spending limit control REST endpoint and uses a configurable processor framework to perform request logging, fault-tolerant forwarding, and ECS integration.

By default, the SLC service processes requests by running the following processors in sequence:

1. **slc-logger**: Performs optional request logging for operational visibility and troubleshooting.
2. **rest-slc-httpgw**: Routes spending limit control requests to HTTP Gateway and ECS for spending limit processing.

For more information about processors, see "About the ECE Composable Service Processors".

Table 6-7 lists the keys for configuring the SLC service processors.

Table 6-7 SLC Service Keys

Key	Path in values.yaml	Description
name	**restServices.processors**	The logical name of the spending limit control service: **SpendingLimitControl**.
type	**restServices.processors**	The service type identifier for the Nchf spending limit control interface: **NCHF_SPENDING_LIMIT_CONTROL**.
config.path	**restServices.processors**	The REST endpoint used to receive SLC requests. The default is **/nchf-spendinglimitcontrol/v1/subscriptions**.
processors[].* (slc-logger)	**restServices.processors**	This processor performs preliminary mediation and optional logging. • **name**: The logical name of the processor: **slc-logger**. • **producerOnly**: Whether the processor is used only for producing log output. The default is **true**. • **type**: The logging processor type: **LOGGING_PROCESSOR**. • **config.logLevel**: The logging level used by the processor. The default is **INFO**. • **config.atMostSeconds**: The maximum frequency for repeated log messages. The default is **10**.
processors[].* (rest-slc-httpgw)	**restServices.processors**	This processor runs CHF-specific charging orchestration and downstream request handling. • **name**: The logical name of the processor: **rest-slc-httpgw**. • **type**: The processor type: **REST_PROCESSOR**. • **config.uri**: The target URI used to forward spending limit control requests to HTTP Gateway. The default is **http://httpgw:8080/nchf-spendinglimitcontrol/v1/subscriptions**. • **faultTolerance.retry.delay**: The initial retry delay in milliseconds. The default is **100**. • **faultTolerance.retry.calls**: The maximum number of retry attempts. The default is **3**. • **faultTolerance.retry.delayFactor**: The retry backoff factor. The default is **1.0**. • **faultTolerance.circuitBreaker.delay**: The delay before circuit-breaker retry attempts. The default is **200**. • **faultTolerance.circuitBreaker.errorRatio**: The error ratio threshold used to open the circuit breaker. The default is **30**. • **faultTolerance.circuitBreaker.successThreshold**: The number of successful calls required to close the circuit breaker. The default is **2**. • **faultTolerance.circuitBreaker.volume**: The sliding window size used by the circuit breaker. The default is **10**.

Configuring the NRF Management Agent

The NRF Management Agent manages the registration and lifecycle of the CHF with the 5G NRF. It publishes the CHF network function profile and exposes Nchf services so that 5G core network functions can discover and access the services dynamically. The agent continuously monitors CHF health and updates NRF registration status based on service availability and readiness. It also provides secure communication with the NRF by using configured OAuth 2.0 authentication and TLS settings.

In multisite deployments, the NRF Management Agent supports resilient connectivity and automatic failover across redundant NRF endpoints.

To configure the NRF Management Agent, set the following keys in your **override-values.yaml** file:

```
nrfManagementAgent:
  enabled: true
  imageRepository: *imageRepository
  imagePullSecrets: *imagePullSecrets
  imagePullPolicy: *imagePullPolicy
  partOfLabel: *partOfLabel
  nrfManagement:
    fullname: nchf-nrf
    additionalLabels: *additionalLabels
    annotations: *annotations
    nodeSelector: *nodeSelector
    container:
      image: *nrfManagementImage
    existingNfProfileConfigMap: *existingNfProfileConfigMap
    nrfConfig:
      nrfClientInstanceName: nchf-nrf
      profileFile: *profileFile
      nfService:
        health:
          enabled: true
          endpoint: http://nchf-converged-charging-admin:8081/health/ready
          timeout: PT5S
          cacheTimeout: PT30S
          defaultCapacity: 100
          suspendOnFailure: true
      tlsProfiles:
        tls1: &tls
          enabled: true
          hostnameVerification: true
          secretName: "nrf-tls-client-creds"
          mtls:
            enabled: true
      authProfiles:
        nrfOauth: &nrfOauth
          type: OAUTH2
          oauth2:
            secretName: "nrf-oauth2-credentials"
            identityUri: ""
            tokenUri: "https://keycloak.svc.cluster.local:8443/realms/nrf-realm/protocol/
openid-connect/token"
            scopes:
              - nnrf-nfm
            tokenTls: *tls
      apiEndpoints:
        - name: NrfSite1
```

```
          heartBeatInterval: 5
          primary:
            name: ad1-pri
            uri: https://nrf-1:8443/
            authentication: *nrfOauth
            tls: *tls
          fallbacks:
            - name: ad2
              uri: https://nrf-2:8443/
              authentication: *nrfOauth
              tls: *tls
            - name: ad3
              uri: https://nrf-3:8443/
              authentication: *nrfOauth
              tls: *tls
  grafanaDashboards: *grafanaDashboards
  serviceMonitor: *serviceMonitor
```

Table 6-8 describes these keys.

Table 6-8 Keys for the NRF Management Agent

Key	Path to values.yaml	Description
enabled	**nchfConvergedCharging.nrfManagementAgent**	Whether the NRF Management Agent is enabled. The default is **true**.
imageRepository	**nchfConvergedCharging.nrfManagementAgent**	The shared image repository used for the NRF Management Agent images.
imagePullSecrets	**nchfConvergedCharging.nrfManagementAgent**	The image pull secrets used to retrieve container images.
imagePullPolicy	**nchfConvergedCharging.nrfManagementAgent**	The image pull policy applied to NRF Management Agent containers.
partOfLabel	**nchfConvergedCharging.nrfManagementAgent**	The shared Kubernetes part-of label value applied to NRF Management Agent resources.
nrfManagement.*	**nchfConvergedCharging.nrfManagementAgent**	The NRF management configuration, including: • **fullname**: The full name value used as the NRF management resource name. The default is **nchf-nrf**. **Note:** The full name must be unique across the sites. • **additionalLabels**: The additional labels applied to NRF management resources. • **annotations**: The annotations applied to NRF management resources. • **nodeSelector**: The node selector used to constrain pod scheduling. • **container.image**: The container configuration, including the NRF management container image. • **existingNfProfileConfigMap**: The existingNfProfileConfigMap value used to reference an existing NF profile ConfigMap.

Table 6-8 (Cont.) Keys for the NRF Management Agent

Key	Path to values.yaml	Description
nrfConfig.*	**nchfConvergedCharging.nrfManagementAgent**	The nrfConfig configuration. • **nrfClientInstanceName**: The NRF client instance name. The default is **nchf-nrf**. • **profileFile**: The profile file used for the NRF client configuration.
nfService.health.*	**nchfConvergedCharging.nrfManagementAgent.nrfConfig**	The health monitoring settings used by the NRF management agent to determine CHF service availability and registration status. • **enabled**: Whether NF service health checks are enabled. The default is **true**. • **endpoint**: The endpoint used to check NF service readiness. The default is **http://nchf-converged-charging-admin:8081/health/ready**. • **timeout**: The amount of time to wait before the health check request times out. The default is **PT5S**. • **cacheTimeout**: The amount of time to cache the health check result. The default is **PT30S**. • **defaultCapacity**: The default NF service capacity value. The default is **100**. • **suspendOnFailure**: Whether the NF service is suspended when the health check fails. The default is **true**.
tlsProfiles.tls1.*	**nchfConvergedCharging.nrfManagementAgent.nrfConfig**	The TLS configuration profiles that the NRF Management Agent uses when connecting to NRF API endpoints and related services. • **enabled**: Whether the TLS profile is enabled. The default is **true**. • **hostnameVerification**: Whether standard HTTPS hostname verification is enabled. The default is **true**. • **secretName**: The Kubernetes Secret name that contains the NRF TLS client credentials. The default is **nrf-tls-client-creds**. • **mtls.enabled**: Whether mutual TLS is enabled for the TLS profile. The default is **true**.

Table 6-8 (Cont.) Keys for the NRF Management Agent

Key	Path to values.yaml	Description
authProfiles.nrfOauth.*	**nchfConvergedCharging.nrfManagementAgent.nrfConfig**	The OAuth 2.0 authentication profile used by CHF when connecting to NRF. • **type**: The authentication profile type. The default is **OAUTH2**. • **oauth2.secretName**: The Kubernetes Secret that contains the OAuth 2.0 credentials for this authentication profile. The default is **nrf-oauth2-credentials**. • **oauth2.identityUri**: The OAuth 2.0 identity URI for this authentication profile. • **oauth2.tokenUri**: The OAuth 2.0 token endpoint URI used to retrieve access tokens. The default is **https://keycloak.svc.cluster.local:8443/realms/nrf-realm/protocol/openid-connect/token**. • **oauth2.scopes**: The OAuth 2.0 scopes requested for this authentication profile. The default is **nnrf-nfm**. • **oauth2.tokenTls**: The TLS or mutual TLS profile used for the OAuth 2.0 token endpoint.
apiEndpoints.*	**nchfConvergedCharging.nrfManagementAgent.nrfConfig**	The NRF endpoint set that CHF uses for NRF communication and failover. • **name**: The name of the NRF API endpoint site. The default is **NrfSite1**. • **heartBeatInterval**: The heartbeat interval for the NRF API endpoint. The default is **5**. • **primary.name**: The name of the primary NRF API endpoint. The default is **ad1-pri**. • **primary.uri**: The URI of the primary NRF API endpoint. The default is **https://nrf-1:8443/**. • **primary.authentication**: The authentication profile used for the primary NRF API endpoint. • **primary.tls**: The TLS profile used for the primary NRF API endpoint. • **fallbacks.name**: The name of the fallback NRF API endpoint. The defaults are **ad2** and **ad3**. • **fallbacks.uri**: The URI of the fallback NRF API endpoint. The defaults are **https://nrf-2:8443/** and **https://nrf-3:8443/**. • **fallbacks.authentication**: The authentication profile used for the fallback NRF API endpoint. • **fallbacks.tls**: The TLS profile used for the fallback NRF API endpoint.
grafanaDashboards.*	**nchfConvergedCharging.nrfManagementAgent**	The Grafana dashboard configuration used for NRF Management Agent observability.
serviceMonitor.*	**nchfConvergedCharging.nrfManagementAgent**	The ServiceMonitor configuration used to expose NRF Management Agent metrics to Prometheus.

7

Migrating the CHF and CGF Composable Services into an Existing ECE System

You can migrate the Oracle Communications Elastic Charging Engine (ECE) Charging Manager (CHF) and Charging Gateway Function (CGF) composable services into an existing ECE multisite deployment to perform 5G rating and unrated CDR replication.

Topics in this document:

About Migrating CHF and CGF into an Existing ECE Deployment

You can deploy the ECE composable services in a standalone environment to generate 5G unrated CDRs. You can also integrate the ECE CHF and CGF composable services into an existing ECE cloud native deployment to support the following functions:

- **Generate 5G unrated CDRs**: In this deployment model, the ECE CHF and CGF composable services replace the CDR Gateway and CDR Formatter components in the ECE cloud native deployment.
- **Perform quota-managed charging and spending limit control (SLC) processing**: In this deployment model, the ECE CHF composable service is added to the ECE cloud native deployment. 5G clients send requests to the CHF rather than the HTTP Gateway.

The migration procedure assumes that the following conditions are met:

- Two active sites are available, and traffic can be administratively redirected with negligible service impact.
- Downstream consumers can process CGF-published unrated CDR payloads from the designated Kafka topics.
- Kafka capacity and availability are sufficient at both sites.
- Traffic consists of offline-only, quota-managed, or SLC processing requests.

Migrating CHF and CGF into an ECE Active-Active Deployment

Perform this procedure only when migrating the ECE CHF and CGF composable services into an existing ECE active-active deployment.

For clarity, the tasks in this document use the following terminology:

- **Primary Site**: The site that processes all live traffic during the migration window.
- **Migrating Site**: The site undergoing migration activities during the current migration cycle.

During the first migration cycle, the site roles are defined as follows:

- Site 1 is the Primary Site.
- Site 2 is the Migrating Site.

In the second migration cycle, the site roles are reversed.

The following sections describe the high-level tasks required to migrate CHF and CGF into an existing ECE active-active system:

1. Prepare and deploy the ECE composable services in each active ECE site. See "Preparing and Deploying Components".
2. Use simulator traffic and, optionally, a limited volume of friendly customer traffic during a controlled maintenance window, to send 5G offline-only charging requests to the CHF endpoint at each site. Then, do the following:
   - Validate the end-to-end processing, aggregation, publication, and consumption of 5G unrated CDRs, including confirmation that all expected records are processed without data loss or duplication.
   - Verify the replication status of the unrated CDR tables in the cnDBTier across Cluster 1 and Cluster 2.
   - Confirm that the deployment meets the defined performance targets, including acceptable replication lag thresholds, throughput requirements, and system stability metrics.
3. Divert traffic away from the Migrating Site and disable HTTP Gateway and CDR generation. See "Redirecting Traffic and Disabling the HTTP Gateway-to-CDR Generation Flow".
4. Apply the endpoint migration changes to the Migrating Site. See "Applying Endpoint Migration Changes".
5. Enable the Migrating Site for live participation. See "Enabling Live Traffic Participation".
6. Validate the Migrating Site to confirm service availability, data consistency, and successful replication. See "Performing Canary Validation and Ramping Live Traffic".
7. Repeat the migration cycle for the remaining site. See "Repeating the Migration Cycle".
8. Restore the normal active-active traffic distribution configuration. See "Restoring Active-Active Traffic Distribution".
9. Scale down the CDR Gateway and Formatter components. See "Scaling Down CDR Gateway and Formatter Components".

Preparing and Deploying Components

Perform the following steps at each active ECE site before migrating live 5G charging traffic to the ECE CHF and CGF composable services:

1. Deploy the ECE CHF and CGF composable services in each active ECE site. See "Deploying the ECE Composable Services".

   Ensure that the deployment includes the CGF cnDBTier database and verify that all CHF, CGF, and cnDBTier pods reach the READY state before proceeding.
2. In the **nfProfile** ConfigMap (**configmap-nfprofile.yaml**), set the **nfStatus** attribute to **UNDISCOVERABLE**:

   ```
   "nfStatus": "UNDISCOVERABLE"
   ```

   This setting prevents other NF services from recognizing and routing network traffic to the CHF.

> **Note**
>
> If the NRF is not used to control the NF network traffic flow to the CHF, defer bringing up the CHF until the traffic is ready to be restored.

3. Configure the CHF producer and CGF consumer for the internal 5G CHF-to-CGF unrated event messaging topic.

   Verify that the messaging topic configuration is identical across both sites and confirm that producer and consumer connectivity is operational before enabling traffic processing.

4. Configure the CGF Kafka producer cluster and topic for external publication of 5G unrated CDRs.

   Ensure that the Kafka bootstrap server configuration, topic names, authentication settings, and replication parameters are validated before enabling traffic processing.

5. Enable CGF cnDBTier replication between the active sites.

   Verify that database replication initializes successfully and that replication status remains healthy before proceeding with traffic validation activities.

6. During the migration window, do not republish pricing data from PDC for updated or newly created offers and price plans.

   The **ReplicatedFederatedCache** federated service manages pricing caches across the active sites. Republishing pricing data during the migration window can cause inconsistencies between the active sites and may result in inconsistent charging behavior.

Redirecting Traffic and Disabling the HTTP Gateway-to-CDR Generation Flow

Perform the following steps to redirect live traffic to the Primary Site, prevent new traffic from reaching the Migrating Site, and safely disable the HTTP Gateway-to-CDR generation flow before enabling CHF-based processing.

1. Move all 5G network traffic processing to the Primary Site using operator-defined procedures.

   Before proceeding to the next step, ensure that the Migrating Site is not processing any 5G network traffic.

2. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **skipActiveActivePreferredSiteRouting** key to **true** only if the current value is **false**. Then, run the **helm upgrade** command to apply the change.

   > **Note**
   >
   > This configuration change is required only during the first site migration cycle.

3. Stop **ReplicatedFederatedCache** federation between the Migrating Site and the Primary Site.

   Verify that cache synchronization activity from the Migrating Site has stopped before continuing.

4. Disable CDR generation on the Migrating Site by setting the following key in the **override-values.yaml** file for **oc-cn-ece-helm-chart**. Then, run the **helm upgrade** command and restart the affected composable services, if required.

   ```
   cdrGenerationEnabled: false
   ```

5. At the Migrating Site, ensure that the active CDR Formatter processes all pending CDR records.

   Before continuing, verify that no unprocessed CDR records remain in the ECE tables and that all queued CDR records have been successfully formatted and delivered.

6. At the Migrating Site, verify the HTTP Gateway registration status in the NRF. If the NRF still shows the service as registered, deregister the HTTP Gateway entry.

7. In the **override-values.yaml** file for **oc-cn-ece-helm-chart**, remove the **nrfRestEndPointUrl** configuration from the HTTP Gateway deployment. Run the **helm upgrade** command to apply the change.

   This configuration change prevents the HTTP Gateway from sending NRF heartbeat requests.

8. Restart the HTTP Gateway deployment.

   After the restart completes, verify that the HTTP Gateway no longer sends heartbeat requests and that the NRF deregisters the service successfully.

Applying Endpoint Migration Changes

Perform the following steps to transition external charging traffic handling from the HTTP Gateway to the CHF deployment at the Migrating Site.

1. At the Migrating Site, configure the CHF deployment as the public nCHF endpoint.
   - Replace the HTTP Gateway endpoint configuration with the CHF endpoint configuration so external network functions route charging requests directly to the CHF service.
   - Verify that the NRF registration information, service endpoint configuration, ingress settings, and load balancer mappings correctly reference the CHF deployment.
2. At the Migrating Site, configure the ECE HTTP Gateway bridge information for the CHF deployment.
   - Ensure that the CHF configuration references the appropriate HTTP Gateway bridge endpoints and communication parameters required for ECS-generated notifications and related charging workflows.
   - Verify that the bridge configuration supports the delivery of RAR and SLC notifications to the appropriate 5G network composable services.

Enabling Live Traffic Participation

Perform the following steps to enable the Migrating Site to receive live charging traffic.

1. Start **ReplicatedFederatedCache** federation between the Migrating Site and the Primary Site.

   Verify that cache federation initializes successfully and that cache synchronization remains healthy before enabling live traffic participation.

2. In the NF Profile ConfigMap (**configmap-nfprofile.yaml**), set the **nfStatus** attribute to **REGISTERED**:

```
"nfStatus": "REGISTERED"
```

This setting re-enables NRF discovery after validation completes and allows live traffic to route safely to the Migrating Site.

Performing Canary Validation and Ramping Live Traffic

Perform the following steps to gradually introduce live traffic to the Migrating Site and verify system stability before restoring normal traffic distribution.

1. Redistribute a small percentage of live 5G traffic to the Migrating Site using operator-defined policies.

   Use a limited traffic volume during the initial validation phase to minimize operational impact if issues occur.

2. Verify that the CHF, CGF, Kafka messaging, cnDBTier replication, and related charging services process traffic successfully without errors or abnormal performance conditions.

   Monitor system metrics, logs, alarms, replication lag, throughput, latency, and resource utilization throughout the soak period.

   > **Note**
   >
   > Proceed only after the deployment remains stable throughout the defined soak interval.

3. Incrementally increase the amount of live traffic directed to the Migrating Site. Gradually adjust traffic-steering policies in controlled increments until the Migrating Site processes the planned production traffic volume.

4. Validate system stability after each traffic increase. Confirm that charging requests, unrated CDR generation, Kafka publication, notification delivery, and database replication continue to operate normally after each traffic increase.

   Before proceeding to the next traffic increment, verify that no service degradation, abnormal latency, replication backlog, or message-processing failures have occurred.

Repeating the Migration Cycle

Perform the following steps to reverse the site roles and repeat the migration procedure for the remaining active site.

1. Verify that both sites are operating normally after federation restoration and live traffic ramp-up complete.
   - Confirm that charging traffic processing, Kafka publication, cnDBTier replication, notification delivery, and cache federation operate successfully across both sites.
   - Review system logs, alarms, dashboards, and replication metrics to verify that no critical errors or performance issues exist before continuing.

2. Swap the site roles for the next migration cycle:
   - Site 2 becomes the new Primary Site.
   - Site 1 becomes the Migrating Site.

   Ensure that traffic-steering policies, NRF registration information, and operational procedures are updated to reflect the revised site roles before continuing.

3. Repeat steps 3 through 6 in "Migrating CHF and CGF into an ECE Active-Active Deployment" to migrate the remaining site and validate live traffic processing in the updated active-active deployment configuration.

Restoring Active-Active Traffic Distribution

Perform the following steps to restore the normal active-active traffic distribution model after both sites successfully complete the migration procedure:

1. In the **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **skipActiveActivePreferredSiteRouting** key to **false** only if you changed the value earlier during the migration procedure. After running the **helm upgrade** command, verify that the updated routing configuration is applied successfully across the deployment.
2. Incrementally redistribute live traffic across both active sites by adjusting the SMF and PCF traffic-steering configuration in controlled increments until the deployment reaches the normal production traffic distribution model.
3. Confirm steady-state traffic processing across both active sites. Verify that the CHF, CGF, Kafka messaging, cnDBTier replication, and cache federation services operate normally at both sites.

   Confirm that charging requests, unrated CDR processing, notification delivery, and database replication complete successfully without errors or performance degradation. Review system logs, alarms, dashboards, throughput metrics, latency metrics, and replication status to confirm stable long-term operation across the active-active deployment.

Scaling Down CDR Gateway and Formatter Components

Perform the following steps after the migration procedure completes and both active sites successfully process live traffic through the CHF and CGF deployment:

1. Verify that the CGF processes all charging requests successfully. Perform the following validation activities:
   - Confirm that unrated CDR aggregation, Kafka publication, notification processing, and downstream integrations operate normally at both active sites.
   - Review system logs, dashboards, alarms, and Kafka topic metrics to verify that no processing failures or message backlogs exist.
2. Verify that no pending CDR records exist in the ECE tables at either site.
   - Ensure that the active Formatter deployment processes and exports all remaining CDR records successfully before continuing.
   - Confirm that no queued, partially processed, or failed CDR records remain in the system.
3. Scale down the CDR Gateway deployment at both active sites. Verify that all CDR Gateway pods terminate successfully and that no traffic continues to route through the legacy HTTP Gateway-to-CDR generation flow.
4. Scale down the Formatter deployment at both active sites.

   After the scale-down operation completes, verify that the deployment no longer runs Formatter processing workloads and that all charging traffic continues to process successfully through the CHF and CGF deployment.

8

Deploying the ECE Composable Services

You can deploy the Oracle Communications Elastic Charging Engine (ECE) composable services individually or in an existing ECE system.

Topics in this document:

Deploying the ECE Composable Services

To deploy the ECE composable services in your cloud native environment:

1. Create a namespace for the ECE composable services Helm chart:

   `kubectl create namespace` *`EceCompServicesNameSpace`*

   where *EceCompServicesNameSpace* is the namespace in which to create Kubernetes objects for the ECE composable services Helm chart.

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

   `helm lint --strict oc-ccs-`*`version`* **`--values`** *`OverrideValuesFile`*

   If the command completes successfully, you see output similar to the following:

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

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

   `helm template` *`EceCompServicesReleaseName`* **`oc-ccs-`***`version`*

   where *EceCompServicesReleaseName* is the release name for **oc-ccs-***version*. Helm uses this name to track the installation instance.

4. Deploy the ECE composable services by running this command:

   `helm install` *`EceCompServicesReleaseName`* **`oc-ccs-`***`version`* **`--namespace`** *`EceCompServicesNameSpace`* **`--values override-values.yaml`**

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

   `kubectl -n` *`EceCompServicesNameSpace`* **`get pods`**

   You should see something similar to this:

   ```
   NAME                                      READY   STATUS     RESTARTS   AGE
   cgf-0                                     1/1     Running    0          3d16h
   cgf-coordinator-6ff6d9d747-vg2bf          1/1     Running    0          3d16h
   nchf-converged-charging-688fd64c6c-zqf8v  1/1     Running    0          3d16h
   nchf-nrf-8545d9b5b8-9mz4h                 1/1     Running    0          3d16h
   ```

Part III

Administering the ECE Composable Services

This part describes how to perform administration tasks on your Oracle Communications Elastic Charging Engine (ECE) composable services.

It contains the following chapters:

9

Managing ECE Composable Service Pods

You can manage the pods in the Oracle Communications Elastic Charging Engine (ECE) composable services by setting up autoscaling.

Topics in this document:

About Autoscaling Pods for ECE Composable Services

You can use the Kubernetes Horizontal Pod Autoscaler to automatically scale replicas up or down based on CPU or memory utilization. In the ECE composable services, the Horizontal Pod Autoscaler can monitor and scale the following pods:

- charging-gateway
- nchf-converged-charging

To achieve the best results, design your autoscaling strategy to balance efficient infrastructure resource usage with minimizing frequent ReplicaSet scaling events.

Setting Up Autoscaling of Pods

To set up and enable autoscaling for the ECE composable service pods:

1. Ensure that your cluster is set up and the system is in the **UsageProcessing** state.
2. Open your **override-values.yaml** file.
3. Ensure the Horizontal Pod Autoscaler is enabled for all pods needed to meet your business needs. You enable or disable it by using these keys in your **override-values.yaml** file:
   - For the **charging-gateway** pod, set the **cgf.chargingGateway.hpa.enabled** key. The default is **true**.
   - For the **nchf-converged-charging** pod, set the **nchfConvergedCharging.nchfConvergedCharging.hpa.enabled** key. The default is **true**.
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 **nchfConvergedCharging.nchfConvergedCharging.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 **nchfConvergedCharging.nchfConvergedCharging.hpa** section.

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

     If the average utilization across a pod's replicas falls below **metrics.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 **metrics.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 for the same deployment. For example, if a cluster has three charging-gateway pod replicas, the average is 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 **nchfConvergedCharging.nchfConvergedCharging.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 **nchfConvergedCharging.nchfConvergedCharging.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-up 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 the ECE composable services (if you have not already deployed them). See "Deploying the ECE Composable Services".
    - If you have already deployed the ECE composable services, update your Helm release:

      ```
      helm upgrade EceCompServicesReleaseName oc-ccs-version --values
      override-values.yaml -n EceCompServicesNameSpace
      ```

      where:

      – *EceCompServicesReleaseName* is the release name for the ECE composable services deployment.

      – *version* is the ECE composable services Helm chart number, such as 3.0.1+e59503ab.

      – *EceCompServicesNameSpace* is the namespace in which to create Kubernetes objects for the Helm chart.

10

Tracing the Flow of API Calls

You can trace the flow of API calls made to the Oracle Communications Elastic Charging Engine (ECE) composable services through Jaeger.

Topics in this document:

About Tracing in the ECE Composable Services

You can trace the flow of messages through the ECE composable services by using the Jaeger tracing tool integrated with the Helidon framework. You use this tool to understand request failures and troubleshoot performance issues in the ECE composable services.

Helidon is a collection of Java libraries used by the ECE composable services, 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

You can trace these ECE composable service pods:

- charging-gateway
- nchf-converged-charging

To set up tracing in the ECE composable services:

1. Install the Jaeger Operator. See the Jaeger Operator for Kubernetes documentation: https://www.jaegertracing.io/docs/1.41/operator/.
2. Enable Jaeger tracing in the ECE composable services. See "Enabling Tracing in the ECE Composable Services".

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

Enabling Tracing in the ECE Composable Services

By default, tracing is disabled in the ECE composable services.

To enable tracing with Jaeger:

1. Create an **override-values.yaml** file for **oc-ccs-***version*.
2. Enable tracing across all ECE composable services by setting these keys:
   - **tracingEnabled**: Set this to **true**. The default is **false**.
   - **tracingHost**: Set this to the host name of the server on which tracing is running. For example: **opentelemetry.telemetry.svc.cluster.local**. The default is **otlp-collector-host**.

- **tracingPort**: Set this to the trace server port. The default is **14250**.

3. (Optional) Customize how the ECE composable services sample messages. Set these keys under the **tracing** section of each ECE composable service:
   - **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 Jaeger sends a batch. The default is **10001** (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 no 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.
   - **samplerManager**: Specify the implementation used to control distributed tracing sampling behavior. A sampler manager can dynamically determine which traces are sampled, how sampling decisions are applied, and whether sampling policies can be updated at runtime.
   - **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 Helm release by running the **helm install** command:

   ```
   helm install EceCompServicesReleaseName oc-ccs-version --values override-values.yaml
   -n EceCompServicesNameSpace
   ```

   where:
   - *EceCompServicesReleaseName* is the release name for the Helm chart. Helm uses this name to track the installation instance.
   - *EceCompServicesNameSpace* is the namespace in which to create Kubernetes objects for the Helm chart.

The following sample **override-values.yaml** file configures tracing for an ECE composable service. It specifies creating a queue of 42 spans, holding events in the queue for 10 seconds, sampling all traces, and logging spans.

```
tracing:
  service: mediation
  maxQueueSize: 42
  flushIntervalMs: 10001
  samplerType: const
  samplerParam: 1
  samplerManager:
  logSpans: true
```

11

Monitoring ECE Composable Service Processes

You can monitor system processes, such as memory and thread usage, in the Oracle Communications Elastic Charging Engine (ECE) composable services.

Topics in this document:

About Monitoring the ECE Composable Services

You can set up monitoring of your ECE composable services using a Kubernetes Service Monitor, the 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.

Setting Up Monitoring of the ECE Composable Services

Setting up monitoring of the ECE composable services involves the following high-level tasks:

1. Deploying the following prerequisite software on your system:
   - Deploying the Prometheus Operator for the ECE composable services. See "Installing Prometheus Operator".
   - Deploying Grafana for the ECE composable services. See "Installing Grafana".

   For the list of compatible versions, see *BRM Compatibility Matrix*.
2. Enabling the Service Monitor and Grafana Dashboards. See "Enabling the Service Monitor".

Enabling the Service Monitor

By default, the Service Monitor and Grafana Dashboards are disabled in the ECE composable services. 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-***version* Helm chart.
2. Enable the Service Monitor and Grafana Dashboards by setting these keys:
   - **serviceMonitor.enabled**: Set this to **true**.

- **serviceMonitor.namespace**: Set this to the namespace in which to deploy the Service Monitor CRD.
- **grafanaDashboards.enabled**: Set this to **true**.
- **grafanaDashboards.grafanaNamespace**: Set this to the namespace in which to deploy the Grafana Dashboards.

3. Optionally, modify the default settings for Grafana:
   - **grafanaDashboards.labels.grafana_dashboard**: Specify the labels to add to the Grafana CRDs. This helps Grafana discover the dashboards.
   - **grafanaDashboards.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 Helm release:

   ```
   helm upgrade EceCompServicesReleaseName oc-ccs-version --namespace
   EceCompServicesNameSpace --values override-values.yaml
   ```

   where *EceCompServicesReleaseName* is the release name for the ECE composable services, and *EceCompServicesNameSpace* is the namespace in which to create Kubernetes objects for the Helm chart.

Metrics for the ECE Composable Services

The ECE composable services can collect metrics in the following groups to produce monitoring data:

- Charging Gateway Metrics
- ECS Publisher Metrics

Charging Gateway Metrics

The Charging Gateway group contains standard metrics for JVM CPU and memory utilization. Table 11-1 lists the metrics in this group.

Table 11-1 Charging Gateway Metrics

Metric	Type	Description
ccs_service_processor_seconds_count	Histogram	Contains the time taken to process the request.
ccs_service_request_seconds_count	Gauge	Contains the total amount of time to process the request.
ccs_service_topic_messages_received	Counter	Tracks the count of messages received from the topic.
ccs_service_request_failed	Counter	Tracks the count of failed subscriber requests.
ccs_service_cgf_kafka_subscriber_count	Gauge	Contains the count of instances of Kafka subscribers.
ccs_service_processor_duplicate	Counter	Tracks the count of duplicate CHF CDR events.
ccs_service_processor_isn_gaps	Counter	Tracks the count of ISN gaps.

Table 11-1 (Cont.) Charging Gateway Metrics

Metric	Type	Description
ccs_service_processor_isn_gap_occurrences	Counter	Tracks the count of ISN gap occurrences.
ccs_service_processor_isn_gap_filled	Counter	Tracks the count of ISN gaps filled.
ccs_service_processor_closed_cdr_count	Counter	Tracks the count of closed CDRs.
ccs_service_retransmitted_requests	Counter	Tracks the number of retransmitted requests.
ccs_cgf_remote_records_consumed_total	Counter	Tracks the number of records consumed from the remote site.
ccs_cgf_remote_records_published_total	Counter	Tracks the count of records published that were consumed from the remote site.
ccs_suspect_retry_count	Counter	Tracks the suspect records retried by the coordinator.
ccs_suspect_published_count	Counter	Tracks the suspect records successfully published by the coordinator.
ccs_local_site_ownership_change_record_count	Counter	Tracks the CDR site ownership transfers handled by the local site ownership service.
ccs_remote_site_ownership_change_record_count	Counter	Tracks the CDR site ownership transfers handled by the remote site ownership service.
ccs_cgf_db_stable_state	Gauge	Contains the stable database availability state: up (**1**) or down (**0**).
ccs_cgf_db_observed_state	Gauge	Contains the most recently observed availability state of the database: up (**1**) or down (**0**).
ccs_cgf_assignor_remote_assignment_active	Gauge	Contains the remote assignment active state: remote assignment active (**1**) or normal (**0**).
ccs_cgf_assignor_partitions_assigned	Gauge	Contains the current partition assignments by group, topic, and site.
ccs_scheduled_task_seconds_count	Counter	Tracks the time taken to execute a scheduled task.
ccs_scheduled_task_enabled	Gauge	Whether the scheduled task is enabled (**1**) or disabled (**0**).
ccs_scheduled_task_last_run	Gauge	Contains the epoch seconds of the most recent task execution.

ECS Publisher Metrics

The ECS Publisher group contains metrics for tracking the processing performance of events sent to the Kafka server. Table 11-2 lists the metrics in this group.

Table 11-2 ECS Publisher Metrics

Metric	Type	Description
ece.ratedevent.publisher.publish.time	Timer	The end-to-end latency of a transactional batch publish, including the mapping/envelope build, asynchronous publish, and the wait up to **publishTimeoutSeconds**. It is recorded after a successful commit.
ece.ratedevent.publisher.batch.size	DistributionSummary	The number of records published in the current transactional batch.
ece.ratedevent.publisher.error	Counter	A monotonic count of publisher errors by bounded errorKind and errorScope.
ece.ratedevent.publisher.dlq.published	Counter	Counts the number of records successfully published to the Dead Letter Queue (DLQ).
ece.ratedevent.publisher.dlq.error	Counter	Counts the number of records that failed to be published to the DLQ.

12

Generating Alerts

You can monitor and route alerts in Oracle Communications Elastic Charging Engine (ECE) composable services by using Prometheus Alertmanager. When alerting is enabled, the ECE composable services continuously evaluate system health and generate alerts based on configurable conditions.

Topics in this document:

About Generating Alerts

Alerts provide a real-time way to monitor the health, performance, and reliability of charging and CDR processing pipelines. They help you detect critical operational conditions, such as processing backlogs, failover activity, retransmissions, and message routing issues, before those issues affect downstream systems or billing accuracy.

In high-throughput, distributed environments, alerts help you quickly identify and respond to failures or delays that can impact financial and operational outcomes. Alerts also support high availability (HA) and multisite deployments by detecting failover conditions and data flow anomalies across sites.

The ECE composable services generate alerts through the following workflow:

1. The internal AlertMetricsService, embedded in the charging-gateway pod, exposes alert and health metrics through an internal **/metrics** endpoint. This endpoint is accessible only from within the charging-gateway pod.
2. Prometheus Operator continuously monitors the charging-gateway pod for conditions that match your configured alert rules. When a rule condition is met, Prometheus generates an alert and forwards it to Alertmanager.
3. Prometheus Alertmanager manages notifications by grouping related alerts, suppressing repeated notifications, and routing alerts to configured channels such as email, Slack, Microsoft Teams, or PagerDuty.

Setting Up Alerts for the ECE Composable Services

To set up alerts for the ECE composable services:

1. Make sure the Prometheus Operator and Alertmanager are installed and running in your Kubernetes environment. For compatible versions, see *BRM Compatibility Matrix*.

   For more information about installing Prometheus Operator and Alertmanager, see the Prometheus Operator documentation on the GitHub website: https://github.com/prometheus-operator/prometheus-operator/tree/main/Documentation/getting-started.
2. Enable alerts, define the alerting rules and thresholds for each alert type, and then deploy or upgrade the **oc-ccs-***version* Helm chart. See "Configuring Alerting Rules and Thresholds".

3. Configure Prometheus Operator to scrape metrics from the charging-gateway pod endpoint.

   For an example, see the **servicemonitor.yaml** file included in the **oc-ccs-***version* Helm chart.

4. Configure Prometheus Alertmanager to send alerts to notification channels such as email, Slack, or PagerDuty.
5. You can view and manage alerts in the Prometheus UI, which shows both real-time and historical alert activity.

Configuring Alerting Rules and Thresholds

The ECE composable services include several built-in alert types that help you monitor behavior and data pipeline health. These alerts detect failover conditions, data reliability issues, and processing backlogs, giving operators visibility into system stability and data integrity.

- **Remote Consumption Alert**: This alert indicates that a site is processing traffic in disaster recovery mode instead of its primary role. Use this alert to identify failover scenarios and verify that traffic shifts correctly during outages or site disruptions.
- **Retransmission Alert**: This alert triggers when the ECE composable services retransmit messages. Retransmissions can indicate upstream instability, duplicate delivery, or network issues. Use this alert to identify conditions that may affect message ordering, duplicate handling, or overall processing performance.
- **Suspend Topic Alert**: This alert triggers when the ECE composable services route messages to a suspend topic because they cannot process them successfully. Use this alert to identify potential data quality or processing issues that require manual investigation or intervention.
- **High CloudEvent Backlog Alert**: This alert monitors the durable retry backlog and triggers when the number of pending events exceeds a configured threshold. Use this alert to detect bottlenecks in Kafka publishing or downstream systems that may delay CDR delivery.
- **CloudEvent Repository Minor Alert**: This lower-severity alert triggers whenever a backlog exists in the retry repository. Use this alert to identify issues early before they grow into larger backlogs or affect the system.

By default, all alerts are disabled.

To enable and configure alerts and thresholds:

1. Open your **override-values.yaml** file for the **oc-ccs-***version* Helm chart.
2. Set the **prometheusRule.enabled** key to **true**.
3. Enable the DR consumption alert by configuring these keys under **prometheusRule**:
   - **remoteConsumptionAlert.enabled**: Set this to **true** to enable the alert.
   - **remoteConsumptionAlert.window**: The amount of time Prometheus evaluates metric data to determine whether the alert condition is occurring. The default is **2m**, which means the last 2 minutes.
   - **remoteConsumptionAlert.forDuration**: The amount of time the alert condition must continuously remain true before the alert is triggered. The default is **0m**, which means to trigger the alert immediately.
   - **remoteConsumptionAlert.severity**: The severity level assigned to alerts. The default is **warning**.

4. Enable the retransmission alert by configuring these keys under **prometheusRule**:
   - **retransmissionAlert.enabled**: Set this to **true** to enable the alert.
   - **retransmissionAlert.window**: The amount of time Prometheus evaluates metric data to determine whether the alert condition is occurring. The default is **2m**, which means the last 2 minutes.
   - **retransmissionAlert.forDuration**: The amount of time the alert condition must continuously remain true before the alert is triggered. The default is **0m**, which means to trigger the alert immediately.
   - **retransmissionAlert.severity**: The severity level assigned to alerts. The default is **warning**.
5. Enable the suspend topic alert by configuring these keys under **prometheusRule**:
   - **suspendTopicAlert.enabled**: Set this to **true** to enable the alert.
   - **suspendTopicAlert.window**: The amount of time Prometheus evaluates metric data to determine whether the alert condition is occurring. The default is **2m**, which means the last 2 minutes.
   - **suspendTopicAlert.forDuration**: The amount of time the alert condition must continuously remain true before the alert is triggered. The default is **0m**, which means to trigger the alert immediately.
   - **suspendTopicAlert.severity**: The severity level assigned to alerts. The default is **warning**.
6. Enable the high CloudEvent backlog alert by configuring these keys under **prometheusRule**:
   - **highCloudEventBacklogAlert.enabled**: Set this to **true** to enable the alert.
   - **highCloudEventBacklogAlert.processor**: The name of the CGF publisher processor that the alert should monitor for retry backlog growth. The default is **cdrPublisher**.
   - **highCloudEventBacklogAlert.threshold**: The size of the retry event backlog that triggers an alert. The default is **100**.
   - **highCloudEventBacklogAlert.forDuration**: The amount of time the alert condition must continuously remain true before the alert is triggered. The default is **5m**.
   - **highCloudEventBacklogAlert.severity**: The severity level assigned to alerts. The default is **critical**.
7. Enable the CloudEvent repository minor alert by configuring these keys under **prometheusRule**:
   - **cloudEventRepositoryMinorAlert.enabled**: Set this to **true** to enable the alert.
   - **cloudEventRepositoryMinorAlert.processor**: The name of the CGF publisher processor that the alert should monitor for retry backlog growth. The default is **cdrPublisher**.
   - **cloudEventRepositoryMinorAlert.threshold**: The size of the retry event backlog that triggers an alert. The default is **0**.
   - **cloudEventRepositoryMinorAlert.forDuration**: The amount of time the alert condition must continuously remain true before the alert is triggered. The default is **0m**, which means to trigger the alert immediately.
   - **cloudEventRepositoryMinorAlert.severity**: The severity level assigned to alerts. The default is **warning**.
8. Save and close your **override-values.yaml** file.

9. Deploy or redeploy the Helm release by running the **helm install** command:

   ```
   helm install EceCompServicesReleaseName oc-ccs-version --values override-values.yaml
   -n EceCompServicesNameSpace
   ```

   where:

   - *EceCompServicesReleaseName* is the release name for the Helm chart. Helm uses this name to track the installation instance.
   - *EceCompServicesNameSpace* is the namespace in which to create Kubernetes objects for the Helm chart.

13

Using Logging for ECE Composable Services

You can review log files to troubleshoot errors and monitor system activity in the Oracle Communications Elastic Charging Engine (ECE) composable services.

Topics in this document:

About Logging

The ECE composable services use the Apache Log4j2 Java logging utility to log information and errors about the following:

- Startup and shutdown activity
- Interaction with external applications at integration points, including interactions with the ingress controller and 5G Network Functions (NFs)
- Charging Manager (CHF) and Charging Gateway Function (CGF) composable services

For general information about:

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

When you deploy the ECE composable services, logging is automatically set up and running in your system with default settings.

- To access the log files, see "Accessing the ECE Composable Service Logs".
- To change the default logging levels, see "Changing the Log Levels".

Accessing the ECE Composable Service Logs

You access the logs by using the **kubectl** command in the ECE composable services namespace.

To access the logs:

1. Retrieve the names of the ECE composable service pods by entering this command:

   ```
   kubectl -n EceCompServicesNameSpace get pods
   ```

   where *EceCompServicesNameSpace* is the namespace in which Kubernetes objects for the Helm chart reside.

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

   ```
   NAME                                    READY   STATUS    RESTARTS   AGE
   cgf-0                                   1/1     Running   0          7h23m
   cgf-coordinator-69f7d75d75-rml48        1/1     Running   0          7h23m
   ```

```
nchf-converged-charging-65cfd79c6c-nvqb8   1/1    Running    0          7h23m
nchf-nrf-7598464577-k8w25                  1/1    Running    0          7h23m
```

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

   kubectl -n *EceCompServicesNameSpace* **logs** *PodName*

   For example, to access the cgf-0 logs, enter:

   kubectl -n *EceCompServicesNameSpace* logs cgf-0

The following is an example of the logs for the cgf-0 pod:

```
2026-05-19T17:58:20,591+00:00 | INFO | | main | r.hikari.HikariDataSource | oracle -
Starting...
2026-05-19T17:58:20,993+00:00 | INFO | | main | er.hikari.pool.HikariPool | oracle -
Added connection com.mysql.cj.jdbc.ConnectionImpl@620c8641
2026-05-19T17:58:21,687+00:00 | INFO | | main | r.hikari.HikariDataSource | oracle -
Start completed.
2026-05-19T17:58:21,797+00:00 | INFO | | main | oracle.cagbu.ccs.cgf.Main | Configured
Kafka subscriber lifecycle for 2 messaging service(s).
2026-05-19T17:58:22,102+00:00 | INFO | | main | gging.log4j.Log4jProvider | Logging
initialization.
```

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

Changing the Log Levels

You can change the logging level for each pod at the root level and 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 pod:

1. Create an **override-values.yaml** file for your ECE composable services Helm chart.
2. To set a pod's root-level logging, set the **logging.rootLevel** key to the desired logging level.
3. To set a pod's package-level logging, set the following keys:
   - **logging.packageLogging.name**: The name of the logging package, such as io.helidon 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 Helm release:

   helm upgrade *EceCompServicesReleaseName* **oc-ccs-**_version_ **--values override-values.yaml -n** *EceCompServicesNameSpace*

The following shows sample logging settings for the charging-gateway pod.

```
cgf:
   chargingGateway:
      logging:
         format:
            type: TEXT
            pattern: *loggingPattern
         rootLevel: DEBUG
         packageLogging:
            - name: com.oracle.cagbu
              level: INFO
            - name: io.helidon
              level: INFO
```