Oracle® Communications

Cloud Native Policy and Charging Rules Function Installation Guide

Release 1.6.0
F31311-01
May 2020

ORACLE

Oracle Communications Cloud Native Policy and Charging Rules Function Installation Guide, Release 1.6.0

F31311-01

Contents

1 Introduction

References 1-1
Acronyms and Terminology 1-1

2 Installing Cloud Native Policy and Charging Rules Function Deployment Package

Pre-Installation Tasks 2-1
- Checking the Software Requirements 2-1
- Checking the Environment Setup 2-2

Installation Tasks 2-3

3 Customizing Cloud Native Policy and Charging Rules Function

4 Enabling LoadBalancer with MetalLB

Updating diam-gateway Service 4-1

5 Uninstalling Cloud Native Policy and Charging Rules Function (CNPCRF)

6 Troubleshooting Cloud Native Policy Charging and Rules Function

A Docker Images

B Deployment Service Type Selection

My Oracle Support

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

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

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

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

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

What's New in This Guide

This section introduces the documentation updates for Release 1.6.0 in Oracle Communications Cloud Native Policy and Charging Rules Function (CNPCRF) Installation Guide.

Significant Documentation Updates for Release 1.6.0

For Release 1.6.0, the following sections in this guide have been updated:

- Updated the Checking the Software Requirements
- Updated the Installation procedure to support hooks
- Added the Troubleshooting chapter

1 Introduction

The Oracle Communications Cloud Native Core Policy and Charging Rules Function (cnPCRF) is a micro-services based solution for managing policy and charging functionality in a 3G and 4G network. Oracle cnPCRF's offers operators a highly scalable PCRF that is cloud ready. It helps service providers to create and manage complex network policies in a telecom network.

Oracle Communications cnPCRF has the Oracle's PCRF functionalities with product architecture designed for the cloud. The cnPCRF provides a new policy designer/ configuration & troubleshooting GUI, besides a set of new functionalities and architectural changes, the prominent features are listed below:

- 3-Tier Architecture with N+M Geo-Redundancy model
- Compliant with 3GPP Release 15
- Leverages a common Oracle Communications Cloud Native Environment (CNE)
- Packaged to support both VM-based and container-based cloud infrastructure
- Policy solution handling 4G Policy and Charging Control (PCC) use cases with support to legacy diameter based interfaces
- Supports CI/CD
- Integrated with Kubernetes and 5G/CNE common services
- Integrated with DevOps workflows
- Supports all legacy diameter interfaces

For more information, see *Oracle Communications Cloud Native Policy and Charging Rules Function User's Guide*.

References

Refer to the following documents for more information about 5G Cloud Native Policy and Charging Rules Function (CNPCRF).

- Cloud Native Environment Installation Document
- Cloud Native PCRF User's Guide

Acronyms and Terminology

The following table provides information about the acronyms and the terminology used in the document.

Table 1-1 Acronyms and Terminology

Acronym	Definition
5GC	5G Core Network
5GS	5G System
CUSTOMER_REPO	The docker registry address in customer side, plus Port No. if registry has port attached
IMAGE_TAG	The image tag from release tar file is 1.4.0, You can decide to use any tag No. Then push related docker image with that specific tag to their registry
METALLB_ADDRESS_POOL	The address pool which configured on metallb to provide external IPs
MPS	Multimedia Priority Service
N3IWF	Non-3GPP InterWorking Function
NAI	Network Access Identifier
NF	Network Function
NGAP	Next Generation Application Protocol
NSI ID	Network Slice Instance Identifier
NSSP	Network Slice Selection Policy
NWDAF	Network Data Analytics Function
PCF	Policy Control Function
SBA	Service Based Architecture
SBI	Service Based Interface
SD	Slice Differentiator

2

Installing Cloud Native Policy and Charging Rules Function Deployment Package

This chapter describes how to install Cloud Native Policy and Charging Rules Function (CNPCRF) on a cloud native environment.

This chapter contains the following:

- Pre-Installation Tasks
- Installation Tasks

Pre-Installation Tasks

Prior to installing the CNPCRF, perform the following tasks:

- Checking the Software Requirements
- Checking the Environment Setup

Checking the Software Requirements

The following softwares must be installed before installing Cloud Native Policy and Charging Rules Function (CNPCRF):

Software	Version
Kubernetes	v1.15.3
HELM	v2.14.3

Additional software that needs to be deployed as per the requirement of the services:

Software	App Version	Notes
elasticsearch	7.4.0	Needed for Logging Area
elastic-curator	2.0.2	Needed for Logging Area
elastic-exporter	1.1.2	Needed for Logging Area
logs	2.7.0	Needed for Logging Area
kibana	7.4.0	Needed for Logging Area
grafana	3.8.4	Needed for Metrics Area
prometheus	9.2.0	Needed for Metrics Area
prometheus-node-exporter	1.6.0	Needed for Metrics Area
metallb	0.8.4	Needed for External IP
metrics-server	2.5.1	Needed for Metric Server
occne-snmp-notifier	0.3.0	Needed for Metric Server
tracer	0.13.3	Needed for Tracing Area

> **Note:**
>
> The above softwares are available if the CNPCRF is deployed in the Oracle Communications Cloud Native Environment (OCCNE). If you are deploying CNPCRF in any other environment, the above softwares must be installed before installing the CNPCRF. To check the installed software items,
>
> ```
> helm ls
> ```
>
> Some of the systems may need to use helm command with **admin.conf** file as follows:
>
> ```
> helm --kubeconfig admin.conf
> ```

Checking the Environment Setup

> **Note:**
>
> This section is applicable only when the Cloud Native Policy and Charging Rules Function (CNPCRF) is deployed in the environment, other than OCCNE.

Network access

The Kubernetes cluster hosts must have network access to:

- Local helm repository, where the CNPCRF helm charts are available.
  To check if the Kubernetes cluster hosts have network access to the local helm repository, execute the following command:

  ```
  helm repo update
  ```

  > **Note:**
  >
  > Some of the systems may need to use helm command with **admin.conf** file as follows:
  >
  > ```
  > helm --kubeconfig admin.conf
  > ```

- Local docker image repository, where the CNPCRF images are available.

To check if the Kubernetes cluster hosts have network access to the local docker image repository, pull any image with tag name to check connectivity by executing the following command:

```
docker pull docker-repo/image-name:image-tag
```

where:

docker-repo is the IP address or host name of the repository.

image-name is the docker image name.

image-tag is the tag the image used for the CNPCRF pod.

> **Note:**
>
> All the kubectl and helm related commands that are used in this guide must be executed on a system depending on the infrastructure/deployment. It could be a client machine, such as, a VM, server, local desktop, and so on.

Client Machine Requirements

Following are the requirements for the client machine where the deployment commands shall be executed:

- It should have network access to the helm repository and docker image repository.
- Helm repository must be configured on the client.
- It should have network access to the Kubernetes cluster.
- It should have necessary environment settings to run the `kubectl` commands. The environment should have privileges to create namespace in the Kubernetes cluster.
- It should have helm client installed with the **push** plugin. The environment should be configured so that the `helm install` command deploys the software in the Kubernetes cluster.

Server or Space Requirements

For information on the server or space requirements, see the *Oracle Communications Cloud Native Environment (OCCNE) Installation Guide*.

Installation Tasks

Downloading CNPCRF package

To download the CNPCRF package from MOS:

1. Login to My Oracle Support with your credentials.
2. Select **Patches and Updates** tab to locate the patch.
3. In **Patch Search** window, click **Product or Family (Advanced)**.

4. Enter *Oracle Communications Cloud Native Core - 5G* in **Product** field, select *Oracle Communications Cloud Native Core Policy and Charging Rules Function 1.6.0.0.0* from **Release** drop-down.
5. Click **Search**. The **Patch Advanced Search Results** displays a list of releases.
6. Click the required patch from the search results. A window opens and click **Download**.
7. Click the zip file to download the package. Package is named as follows:
   `ReleaseName-pkg-Releasenumber.tgz`

   where:

   ReleaseName is a name which is used to track this installation instance.

   Releasenumber is the release number.
   For example, `occnpcrf-pkg-1.6.0.tgz`

Pushing the Images to Customer Docker Registry

To Push the images to customer docker resgistry:

1. Untar the CNPCRF package file to get CNPCRF docker image tar file.
   `tar -xvzf ReleaseName-pkg-Releasenumber.tgz`

   The directory consists of the following:

   - **CNPCRF Docker Images File:**
     `occnpcrf-images-1.6.0.tar`
   - **Helm File:**
     `occnpcrf-1.6.0.tgz`
   - **Readme txt File:**
     `Readme.txt` (Contains cksum and md5sum of tarballs)
2. Load the **occnpcrf-images-1.6.0.tar** file into the Docker system

   ```
   docker load --input /IMAGE_PATH/occnpcrf-images-1.6.0.tar
   ```

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

   ```
   docker images
   ```

   ReferDocker Images for more information on docker images available in CNPCRF.
4. Create a new tag for each imported image and push the image to the customer docker registry by entering this command:

   ```
   docker tag occnpcrf/diam-gateway:1.6.0 CUSTOMER_REPO/diam-gateway:1.6.0
   docker push CUSTOMER_REPO/diam-gateway:1.6.0

   docker tag occnpcrf/pcrf_core:1.6.0 CUSTOMER_REPO/pcrf_core:1.6.0
   docker push CUSTOMER_REPO/pcrf_core:1.6.0

   docker tag occnpcrf/ocpm_cm_service:1.6.0 CUSTOMER_REPO/ocpm_cm_service:
   1.6.0
   docker push CUSTOMER_REPO/ocpm_cm_service:1.6.0

   docker tag occnpcrf/ocpm_config_server:1.6.0 CUSTOMER_REPO/
   ```

```
ocpm_config_server:1.6.0
docker push CUSTOMER_REPO/ocpm_config_server:1.6.0

docker tag occnpcrf/ocpm_pre:1.6.0 CUSTOMER_REPO/ocpm_pre:1.6.0
docker push CUSTOMER_REPO/ocpm_pre:1.6.0

docker tag occnpcrf/readiness-detector:1.6.0 CUSTOMER_REPO/readiness-
detector:latest
docker push CUSTOMER_REPO/readiness-detector:latest

docker tag occnpcrf/soapconnector:1.6.0 CUSTOMER_REPO/soapconnector:
1.6.0
docker push CUSTOMER_REPO/soapconnector:1.6.0

docker tag occnpcrf/policyds:1.6.0 CUSTOMER_REPO/policyds:1.6.0
docker push CUSTOMER_REPO/policyds:1.6.0

docker tag occnpcrf/ldap-gateway:1.6.0 CUSTOMER_REPO/ldap-gateway:1.6.0
docker push CUSTOMER_REPO/ldap-gateway:1.6.0

docker tag occnpcrf/ocingress_gateway:1.5.1 <CUSTOMER_REPO>/
ocingress_gateway:1.5.1
docker push <CUSTOMER_REPO>/ocingress_gateway:1.5.1

docker tag occnpcrf/ocpm_queryservice:1.6.0 <CUSTOMER_REPO>/
ocpm_queryservice:1.6.0
docker push <CUSTOMER_REPO>/ocpm_queryservice:1.6.0
```

where:
CUSTOMER_REPO is the docker registry address having Port Number, if registry has port attached.

> **Note:**
>
> For OCCNE, copy the package to bastion server and use **localhost: 5000** as CUSTOMER_REPO to tag the images and push to bastion docker registry.

> **Note:**
>
> You may need to configure the Docker certificate before the push command to access customer registry via HTTPS, otherwise, docker push command may fail.

Installing the CNPCRF Package

To install the CNPCRF package:

1. Login to the server where the ssh keys are stored and SQL nodes are accessible.
2. Connect to the SQL nodes.

3. Login to the database as a root user.
4. Create a privileged user and grant all the permissions to the privileged user by executing the following command:

   ```
   CREATE USER 'username'@'%' IDENTIFIED BY 'password';
   GRANT ALL PRIVILEGES ON *.* to 'username'@'%';
   FLUSH PRIVILEGES;
   ```

   where:
   username is the username and *password* is the password for MYSQL privileged user.

   For Example: In the below example "pcrfprivilegedusr" is used as username, "pcrfprivilegedpasswd" is used as password and granting all the permissions to "pcrfprivilegedusr".

   ```
   CREATE USER 'pcrfprivilegedusr'@'%' IDENTIFIED BY
   'pcrfprivilegedpasswd';
   GRANT ALL PRIVILEGES ON *.* to 'pcrfprivilegedusr'@'%';
   FLUSH PRIVILEGES;
   ```

5. Create a user and grant the necessary permissions to access the tables on all SQL nodes by executing the following command:

   ```
   CREATE USER 'username'@'%' IDENTIFIED BY 'password';
   ```

   where:

   username is the username and *password* is the password for MYSQL database user.

   For Example: In the below example "pcrfusr" is used as username, "pcrfpasswd" is used as password and granting the necessary permissions to "pcrfusr".

   ```
   CREATE USER 'pcrfusr'@'%' IDENTIFIED BY 'pcrfpasswd';
   GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'pcrfusr'@'%';
   ```

   > **Note:**
   >
   > For multiple CNPCRF deployments pointing to the same MySQL database, ensure that the database names has to be unique for each CNPCRF deployment.The database name can be specified in the **envMysqlDatabase** parameter in the custom-value.yaml file.

6. Execute the command, `show grants for pcrfprivilegedusr`, to confirm that privileged user has all the permission.
7. Exit from database and logout from MYSQL node.
8. Create namespace if already does not exists by entering the command:

   ```
   kubectl create namespace release_namespace
   ```

where:
release_namespace is the deployment CNPCRF namespace used by helm command.

9. Create a kubernetes secret for a privileged and an application user. In CNPCRF, there are two type of users, one is application user who can perform INSERT, UPDATE, DELETE and SELECT operations on database created. Another one is privileged deployment user who can use helm hooks to perform DDL operations to perform install/upgrade/rollback or delete operations.
To create a kubernetes secret for storing database username and password for these users :

   a. Create a yaml file with the application user's username and password. Below is a sample of yaml file, where, "pcrfusr" is used as username, "pcrfpasswd" is used as password.

```
apiVersion: v1
kind: Secret
metadata:
  name: pcrf-db-pass
type: Opaque
data:
  mysql-username: cGNyZnVzcg==
  mysql-password: cGNyZnBhc3N3ZA==
```

> **Note:**
>
> The values for **mysql-username** and **mysql-password** should be base64 encoded.

   b. Create a yaml file with the privileged user's username and password. Below is a sample of yaml file, where, "pcrfprivilegedusr" is used as username, "pcrfprivilegedpasswd" is used as password.

```
apiVersion: v1
kind: Secret
metadata:
  name: pcrf-privileged-db-pass
type: Opaque
data:
  mysql-username: cGNyZnByaXZpbGVnZWR1c3I=
  mysql-password: cGNyZnByaXZpbGVnZWRwYXNzd2Q=
```

> **Note:**
>
> The values for **mysql-username** and **mysql-password** should be base64 encoded.

c. Execute the following commands to add the kubernetes secrets in a namespace:

```
kubectl create -f yaml_file_name1 -n release_namespace
kubectl create -f yaml_file_name2 -n release_namespace
```

where:
release_namespace is the deployment namespace used by the helm command.

yaml_file_name1 is a name of the yaml file that is created in step 9a.

yaml_file_name2 is a name of the yaml file that is created in step 9b.

10. Create the customize *custom-values-cnpcrf.yaml* file with the required input parameters. To customize the file, see Customizing Cloud Native Policy and Charging Rules Function.

> **Note:**
>
> The values of the parameters mentioned in the custom values yaml file overrides the defaults values yaml file specified in the helm chart. If the **envMysqlDatabase** parameter is modified, then you should modify the **envMysqlDatabaseConfigServer** parameter with the same value in values.yaml file of policycds.

11. Install CNPCRF by entering this command:

```
helm install HELM_CHART --name release_name --namespace
release_namespace -f CUSTOM_VALUES_YAML_FILE
--atomic
```

where:
HELM_CHART is the location of the helm chart extracted from occnpcrf-pkg-1.6.0.tgz file

release_name is the release name used by helm command.

release_namespace is the deployment namespace used by helm command.

CUSTOM_VALUES_YAML_FILE - is the name of the custom values yaml file (including location).

For example:

```
helm install /home/cloud-user/occnpcrf/pcrf --name occnpcrf --namespace
occnpcrf -f ocpcrf-custom-values-1.6.0.yaml --atomic
```

> **Note:**
>
> Do not exit from `helm install` manually. After running `helm install` command, it takes some time to install all the services. In the meantime, you must not press "ctrl+c" to come out from `helm install` command. It leads to some anomalous behavior.

Parameters in `helm install` command:

- **atomic**: If this parameter is set, installation process purges chart on failure. The --wait flag will be set automatically.
- **wait**: If this parameter is set, installation process will wait until all pods, PVCs, Services, and minimum number of pods of a deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout.
- **timeout *duration***: Specifies the time to wait for any individual kubernetes operation (like Jobs for hooks). Default value is 5m0s. If the `helm install` command fails at any point to create a kubernetes object, it will internally call the purge to delete after timeout value (default: 300s). Here timeout value is not for overall install, but it is for automatic purge on installation failure.

12. Check the installation status by entering this command

```
helm status release_name
```

where:
release_name is the release name used by helm command.

You will see the status as **DEPLOYED** if the installation has been done successfully.
Execute the following command to get status of jobs and pods:

```
kubectl get jobs,pods -n release_namespace
```

You will see the status as **Running** for all the nodes if the installation has been done successfully.
If any pod is found with error status, execute this command to describe the pod events:

```
kubectl describe pod pod-name -n release_namespace
```

where, *pod-name* is the name of a pod with an error status.
If any pod is found with error status, execute this command for log details:

```
kubectl logs -f pod pod-name -n release_namespace
```

3
Customizing Cloud Native Policy and Charging Rules Function

This chapter describes how to customize the Cloud Native Policy and Charging Rules Function (CNPCRF) deployment in a cloud native environment.

The CNPCRF deployment is customized by overriding the default values of various configurable parameters in the **custom-values-cnpcrf.yaml** file.

To customize the **custom-values-cnpcrf.yaml** file as per the required parameters:

1. Go to the Oracle Help Center (OHC) Web site: https://docs.oracle.com
2. Navigate to **Industries**->**Communications**->**Cloud Native Core**
3. Click the **CNPCRF Custom Templates** link to download the zip file.
4. Unzip the file to get the `occnpcrf-custom-values-1.6.0.yaml`. This file is used during installation.
5. Customize the **occnpcrf-custom-values-1.6.0.yaml** file.
6. Save the updated **occnpcrf-custom-values-1.6.0.yaml** file in the helm chart directory.

Following is a sample **occnpcrf-custom-values-1.6.0.yaml** file .

```
#section: global
global:
  # docker registry name
  dockerRegistry: '<>'
  imageServiceDetector: readiness-detector:1.6.0
  # Tag name of image
  imageTag: '<>'
  # Primary MYSQL Host IP or Hostname
  envMysqlHost: '<>'
  # Jaegar hostname
  envJaegerAgentHost: ''
  # System name
  envSystemName: 'PCRF'
  releaseDbName: 'pcrf_release'
  # K8s secret object name containing cnPCRF MYSQL UserName and Password
  dbCredSecretName: 'pcrf-db-pass'
  privilegedDbCredSecretName: 'pcrf-privileged-db-pass'
# section: for core service
pcrf-core:
  # database name core service will connect to
  envMysqlDatabase: pcrf
# section: for config server
config-server:
  # database name config service will connect to
```

```
envMysqlDatabase: ocpm_config_server
```

4
Enabling LoadBalancer with MetalLB

Oracle Communications Cloud Native Environment (OCCNE) have MetalLB installed, and free external IPs are already configured under MetalLB.

Perform the following steps to enable LoadBalancer to specific services.

> **Note:**
>
> MetalLB configuration is supported only from OCCNE 1.4.

> **Note:**
>
> In the CNPCRF namespace, only diam-gateway service and cm service with GUI page requires loadbalancer setting with accessible external IP.

Updating diam-gateway Service

To update diam-gateway service:

1. Login to Kubernetes cluster master node using ssh command.
2. Run the following command to edit svc yaml file for diam-gateway:

   ```
   kubectl edit svc diam-gateway-service -n PCRF_NAME_SPACE
   ```

Table 4-1 Variables

Variable Name	Description
diam-gateway-service	The name of diam-gateway service in setup.
PCRF_NAME_SPACE	The --namespace value used in helm install command.

Following is an sample content that displays in diam-gateway edit window.

```
1 # Please edit the object below. Lines beginning with a '#' will be
ignored,
 2 # and an empty file will abort the edit. If an error occurs while
saving this file will be
 3 # reopened with the relevant failures.
 4 #
 5 apiVersion: v1
 6 kind: Service
 7 metadata:
 8   creationTimestamp: 2019-06-02T13:06:11Z
```

```
 9   labels:
10     category: common
11     io.kompose.service: <PCRF_NAME>-pcrf-diam-gateway-service
12   name: <PCRF_NAME>-pcrf-diam-gateway-service
13   namespace: <PCRF_NAME_SPACE>
14   resourceVersion: "21624671"
15   selfLink: /api/v1/namespaces/<PCRF_NAME_SPACE>/services/
<PCRF_NAME>-pcrf-diam-gateway-service
16   uid: 31a4b13f-8537-11e9-81c8-0010e08b3a8e
17 spec:
18   clusterIP: 10.20.37.37
19   externalTrafficPolicy: Cluster
20   ports:
21   - name: diameter
22     nodePort: 32592
23     port: 3868
24     protocol: TCP
25     targetPort: 3868
26   - name: http
27     nodePort: 31301
28     port: 8080
29     protocol: TCP
30     targetPort: 8080
31   selector:
32     io.kompose.service: <PCRF_NAME>-pcrf-diam-gateway-service
33   sessionAffinity: None
34   type: NodePort
35 status:
36   loadBalancer: {}
```

3. Add two new lines after line 7, after "metadata":

   `annotations:`

   `metallb.universe.tf/address-pool: ADDRESS_POOL_NAME`

   > **Note:**
   >
   > - As per user MetalLB setting, you should select an appropriate pool name to replace the variable, *ADDRESS_POOL_NAME*
   > - *annotation:* line must be kept vertical align with line 16, while following line, *metallb.universe.tf/address-pool: ADDRESS_POOL_NAME* must be kept vertical align with line 10. If vertical align restriction failed to follow this rule, the svc yaml file update may fail.

4. Replace line 34 text, **type: NodePort** with **type: LoadBalancer**.
   Following is the sample content after replacing the line 29:

```
1 # Please edit the object below. Lines beginning with a '#' will be
ignored,
 2 # and an empty file will abort the edit. If an error occurs while
saving this file will be
```

```
 3 # reopened with the relevant failures.
 4 #
 5 apiVersion: v1
 6 kind: Service
 7 metadata:
 8   creationTimestamp: 2019-06-02T13:06:11Z
 9   labels:
10     category: common
11     io.kompose.service: <PCRF_NAME>-pcrf-diam-gateway-service
12   name: <PCRF_NAME>-pcrf-diam-gateway-service
13   namespace: <PCRF_NAME_SPACE>
14   resourceVersion: "21624671"
15   selfLink: /api/v1/namespaces/<PCRF_NAME_SPACE>/services/
<PCRF_NAME>-pcrf-diam-gateway-service
16   uid: 31a4b13f-8537-11e9-81c8-0010e08b3a8e
17 spec:
18   clusterIP: 10.20.37.37
19   externalTrafficPolicy: Cluster
20   ports:
21   - name: diameter
22     nodePort: 32592
23     port: 3868
24     protocol: TCP
25     targetPort: 3868
26   - name: http
27     nodePort: 31301
28     port: 8080
29     protocol: TCP
30     targetPort: 8080
31   selector:
32     io.kompose.service: <PCRF_NAME>-pcrf-diam-gateway-service
33   sessionAffinity: None
34   type: LoadBalancer
35 status:
36   loadBalancer: {}
```

5. Quit vim editor and save changes. A new diam-gateway pod starts up.
   a. In the new service, following sample content displays. Note that if the EXTERNAL-IP is available, then the load balancer setting for diam-gateway service works.

```
NAME                          TYPE           CLUSTER-IP      EXTERNAL-
IP    PORT(S)                                                    AGE
<PCRF_NAME>-diam-gateway-service       LoadBalancer
10.xxx.xx.xx    10.xxx.xxx.xx   3868:32592/TCP,8080:31301/TCP 4d
```

Updating cm-service

Follow the same process to update svc yaml for *PCRF_NAME* -pcrf-cm-service.

5 Uninstalling Cloud Native Policy and Charging Rules Function (CNPCRF)

When you uninstall a Helm chart from your CNPCRF deployment, it removes only the Kubernetes objects that it created during installation.

To uninstall, enter this command:

```
helm delete release_name
```

where *release_name* is the release name used by helm command.

Helm keeps a record of its releases, so you can still re-activate the release after you uninstall it.

To completely remove the release from the cluster, add the --purge option to the command:

```
helm delete --purge release_name
```

For example, to completely remove a release named "occnpcrf", you'd enter this command:

```
helm delete --purge occnpcrf
```

Deleting Kubernetes namespace

To delete kubernetes namespace, enter this command:

```
kubectl delete namespace release_namespace
```

where *release_namespace* is the deployment namespace used by helm command.

For example, to delete a kubernetes namespace named "occnpcrf", you'd enter this command:

```
kubectl delete namespace occnpcrf
```

Cleaning Up Database

To clean up database, Login to MYSQL as a root user, enter this command:

```
DROP DATABASE IF EXISTS database_name;
```

where *database_name* is the database name.

6 Troubleshooting Cloud Native Policy Charging and Rules Function

This section provides information to troubleshoot the common error which can be encountered during the installation and upgrade of Cloud Native Policy Charging and Rules Function.

If `helm install` Command Fails

This section covers the reasons and troubleshooting procedures if the `helm install` command fails.

Reasons for `helm install` failure:

- **Chart syntax issue [This issue could be shown in the few seconds]**
  Please resolve the chart specific things and reinstall the `helm install` command, because in this case, no hooks should have begun.
- **Most possible reason [TIMEOUT]**
  If any job stuck in a pending/error state and not able to execute, it will result in the timeout after 5 minutes. As default timeout for `helm install` is 5 minutes. In this case, follow the below troubleshooting steps to troubleshoot.
- **All pods/deployment/replicasets/statefulsets/services are up and running but ExternalIP for LoadBalancer type service is in <pending> state**
  If ExternalIP is in the pending state for LoadBalancer type services, then `helm install` will fail and purge the charts with the following error

  ```
  helm install /home/cloud-user/occnpcrf/pcrf --name occnpcrf --namespace
  occnpcrf -f custom-values-cnpcrf.yaml --atomic



  INSTALL FAILED
  PURGING CHART


  Error: release occnpcrf failed: timed out waiting for the
  conditionSuccessfully purged a chart!
  Error: release occnpcrf failed: timed out waiting for the condition
  ```

  In this case, Don't use **--atomic** option while executing the `helm install`
- **Helm install failed in case of duplicated chart**

  ```
  helm install <helm-chart> --name occnpcrf --namespace occnpcrf -f
  custom-values-cnpcrf.yaml
  Error: release occnpcrf failed: configmaps "perfinfo-config-occnpcrf"
  already exists
  ```

Here, configmap 'perfinfo-config-occnpcrf' exists multiple times while creating kubernetes objects after pre-install hooks, the installation will fail. In this case also please go through the below troubleshooting steps.

Troubleshooting Steps:

1. Execute the below command to cleanup the databases created by the "helm install" command :

   ```
   DROP DATABASE IF EXISTS `pcrf`;
   DROP DATABASE IF EXISTS `ocpm_config_server`;
   DROP DATABASE IF EXISTS `pcrf_release`;
   ```

2. Execute the below command to get kubernetes objects:

   ```
   kubectl get all -n <release_namespace>
   ```

   This gives a detailed overview of which objects are stuck or in a failed state.

3. Execute the below command to delete all kubernetes objects:

   ```
   kubectl delete all --all -n <release_namespace>
   ```

4. Execute the below command :

   ```
   helm ls --all
   ```

   If this is in a failed state, perform purging using the command

   ```
   helm delete --purge <release_name>
   ```

   > **Note:**
   >
   > If the execution of this command is taking more time, run the below command parallelly in another session to clear all the delete jobs.
   >
   > ```
   > while true; do kubectl delete jobs --all -n
   > <release_namespace>; sleep 5;done
   > ```

   Monitor the `helm delete --purge <release_name>` command. Once that is succeeded, press "ctrl+c" to stop the above script.

A
Docker Images

Cloud Native Policy and Charging Rules Function (CNPCRF) deployment package includes ready-to-use images and Helm charts to help you orchestrate containers in Kubernetes.

You can use the Docker images and Helm chart to help you deploy and manage Pods of CNPCRF product services in Kubernetes. Communication between Pods of services of CNPCRF products are preconfigured in the Helm charts.

Table A-1 lists the docker images for CNPCRF.

Table A-1 Docker Images for CNPCRF

Service Name	Docker Image Name	Service Category
CM Service	ocpm_cm_service	Common
Config Server Service	ocpm_config_server	Common
Diameter Gateway	diam-gateway	PCRF
Ingress Gateway	ocingress_gateway	Common
Ldap Gateway Service	ldap-gateway	Common
PolicyDS service	policyds	Common
Soap Connector Service	soapconnector	Common
Pcrf core Service	pcrf_core	PCRF
Policy Runtime Service	ocpm_pre	Common
Query Service	ocpm_queryservice	Common
Readiness Detector	readiness-detector	Common

Table A-2 Service Category Details

Service Category	Description
common	Common service module which would be consumed by CNPCRF as supporting service, such as GUI, policy engine
PCRF	CNPCRF core services which contains the core business logic of NF

B

Deployment Service Type Selection

S.No.	Service Type	Description
1.	ClusterIP	Exposes the service on a cluster-internal IP. Specifying this value makes the service only reachable from within the cluster. This is the default ServiceType.
2.	NodePort	Exposes the service on each Node's IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service will route, is automatically created. You'll be able to contact the NodePort service, from outside the cluster, by requesting *MasterNodeIP*:*NodePort*
3.	LoadBalancer	Exposes the service externally using a cloud provider's load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created. For CM Service, API gateway, Diameter Gateway service, it's recommended to use LoadBalancer type. Given that the CNE already integrated with a load balancer (METALLB, for OCCNE deployed on baremetal).