Oracle® Communications
Unified Data Repository Provisioning Gateway Guide

Release 1.6
F31310-02
May 2020

Oracle Communications Unified Data Repository Provisioning Gateway Guide, Release 1.6

F31310-02

Contents

1

Introducing Provisioning Gateway

In this chapter, you will get an overview of Provisioning Gateway, its architecture and the operations it supports.

Overview

In this section, you will get an overview on Provisioning Gateway.

Oracle Provisioning Gateway is implemented as a cloud-native function. It offers a rest interface for SLF data provisioning. It relays the request received by the provisioning system to multiple 5G UDRs. It then consolidates the response received from each UDR and sends back the final response to the provisioning system.

With the help of Ingress Gateway, Provisioning Gateway provides an HTTP2 based secured RESTful interface for provisioning clients.

Architecture

In this section, you will learn about Provisioning Gateway architecture.

Provisioning Gateway:

- Interfaces with provisioning system on the northbound and UDR (5G) on the southbound interface.
- Provisioning system uses REST-JSON based interface to send requests to UDR.
- Forks requests to multiple UDRs (multiple segments like UDR1, UDR2). The number of UDRs to which requests are forked is configurable
- Is deployed in a pair. Each one is stateless and do not communicate with each other. If one of the Provisioning Gateway fails, provisioning system can send commands to second Provisioning Gateway to continue provisioning
- Micro services are:
  - **Ingress API Gateway:** receives requests from provisioning system and forwards the same to provGwy service. It also provides TLS for secure request handling and loadbalances among all provGwy service pods.
  - **ProvGwy service:** is the core service, which handles the main business logic of forking requests to multiple UDR/SLF. The service ensures the provisioning operations are atomic across segments i.e. returns success only when requests are successful on all UDR/SLFs.
- CNF is integrated with OCCNE services like EFK, Prometheus/Grafana and Jaegar.

List of Operations Supported

In this section, you will learn about the Network Function (NF) related operations for provisioning.

The NF-group-id related operations for provisioning are:

- **Update SLF data for a subscriber:** Adds or updates the SLF data.
- **Get SLF Data:** Retrieves the nf-group information of a subscriber.
- **Delete SLF Data:** Deletes the nf-group information and related data for a subscriber.

References

In this section, you will learn about Provisioning Gateway references.

Refer to the following documents for more information on Provisioning Gateway usage in 5G cloud native environment.

- Unified Data Repository Installation and Upgrade Guide
- Unified Data Repository User's Guide

2

Installing Provisioning Gateway

In this chapter, you will learn to install Provisioning Gateway.

Planning Your Installation

In this section, you will learn to plan Provisioning Gateway installation.

Pre-installation Tasks:

- Checking the software requirements
- Checking the environment setup

Checking the Software Requirements

In this section, you will learn about softwares required to install provisioning gateway.

Before installing Provisioning Gateway, install the following softwares on your system.

Software	Version
Kubernetes	v1.13.3
HELM	v2.12.3

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

Software	Chart Version	Notes
elasticsearch	1.21.1	Needed for Logging Area
elastic-curator	1.2.1	Needed for Logging Area
elastic-exporter	1.1.2	Needed for Logging Area
logs	2.0.7	Needed for Logging Area
kibana	1.5.2	Needed for Logging Area
grafana	2.2.0	Needed for Metrics Area
prometheus	8.8.0	Needed for Metrics Area
prometheus-node-exporter	1.3.0	Needed for Metrics Area
metallb	0.8.4	Needed for External IP
metrics-server	2.4.0	Needed for Metric Server
tracer	0.8.3	Needed for Tracing Area

Note:

The above softwares are available in the **Oracle Communications Cloud Native Environment (OCCNE)**. If you are deploying Provisioning Gateway in any other environment, then the above softwares must be installed before installing Provisioning Gateway.

To check the installed software items, execute the following command:

```
helm ls
```

Some systems may need to use helm command with **admin.conf** file as follows:

```
helm --kubeconfig admin.conf
```

Checking the Environment Setup

In this section, you will learn to setup an environment to install Provisioning Gateway.

Before installing Provisioning Gateway, your system should have the following:

- **Provisioning Gateway Software:** The ProvGateway software consists of:
  - **ProvGw Helm Chart:** It reflects the ProvGateway software version. It comes in the form of a zipped tar file.
  - **Software images of the micro-services**. The images are available in the form of docker images and/or tar file.

  Note:

  For more details about ProvGateway software, see Checking the Software Requirements

- **Network Access:** The Kubernetes cluster hosts must have network access to:
  - Local docker image repository where the ProvGateway images are available.
  - Local helm repository where the ProvGateway helm charts are available.

  Note:

  Execute all the kubectl and helm commands used in this document on a system depending on the infrastructure of the deployment. It may be some client machine like virtual machine, server, local desktop so on).

- **Laptop/Desktop Client Software:** A laptop/desktop where the user executes deployment commands should have:
  - Network access to the helm repository and docker image repository
  - Helm repository configured on the client
  - Network access to the Kubernetes cluster

- Necessary environment settings to run the 'kubectl' commands. The environment should have privileges to create namespace in the Kubernetes cluster.
- 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.

Installation Sequence

In this section, you will learn about ProvGateway installation sequence:

The installation sequence of ProvGateway is as follows:

1. Installation Preparation
2. **Namespace Creation:** Refer to *Unified Data Repository Installation and Upgrade Guide - OCUDR Namespace Creation*.

   Note:

   Modify the namespace as provgw.

3. **Service Account, Role and RoleBinding Creation:** Refer to *Unified Data Repository Installation and Upgrade Guide - OCUDR Service Account, Role and RoleBinding Creation*.

   Note:

   Use provgw in place of ocudr.

4. **Kubernetes Secret Creation** (provgw-ingress-secret) for storing private keys and certificates for https: Refer to *Unified Data Repository Installation and Upgrade Guide - Kubernetes Secret Creation: Private Keys and Certificates for IngressGateway*.

   Note:

   Use provgw in place of ocudr.

5. **provgw-custom-values.yaml File Configuration**: This includes repository path, primary and secondary node and ProvGateway details configuration. Other configurations may change depending on the deployment.

   Note:

   For more details, you can refer to Customizing Provisioning Gateway.

6. **ProvGateway Deployment and Verification:** You can deploy ProvGateway either with **HELM repository** or with **HELM tar** in Kubernetes cluster. Execute the following command to deploy ProvGateway:

```
helm install <helm chart> [--version <ProvGw version>] --name
<release> --namespace <k8s namespace> -f <provgw-custom-
values-1.6.0.yaml>
```

In the above command:

- **<helm chart>:** is the name of the chart which is of the form <helm repo>/ provgw.
- **<ProvGw version>:** is the software version (helm chart version) of the ProvGw. This is optional. If omitted, the default is 'latest' version available in helm repository.
- **<release>:** is a user defined name to identify the helm deployment. From 1.6.0 release onwards, all pod names, service name, deployment name are prepended by this release name.
- **<k8s namespace>:** is a user defined name to identifying the kubernetes namespace of the ProvGw. All the ProvGw micro services are deployed in this kubernetes namespace.
- **<provgw-custom-values-1.6.0.yaml>:** is the customized provgw-custom-values-1.6.0.yaml file. For more details, refer to Customizing Provisioning Gateway.

Note:

If helm3 is used, then execute the following command for installation:
```
helm install -name <release> --namespace <k8s namespace> -f
<provgw-custom-values-1.6.0.yaml> <helm chart>
```

After Provgateway deployment, you need to verify whether all the services and pods are up and running.

Installation Preparation

In this section, you will learn to prepare for ProvGateway installation.

Installation Preparation includes downloading the required files and loading the files to the system. The steps are:

1. Download the ProvGateway package file from Oracle Software Delivery Cloud (OSDC). Execute the following command to download ProvGateway package.
   `<nfname>-pkg-<marketing-release-number>.tgz`
2. Untar the ProvGateway Package File. Execute the following command to untar ProvGateway Package File.
   `tar -xvf provgw-pkg-1.6.0.tgz`

   This command results into `provgw-pkg-1.6.0` directory. The directory consists of following:

   - **ProvGw Docker Images File:** `provgw-images-1.6.0.tar`
   - **Helm File:** `provgw-1.6.0.tgz`
   - **Readme txt File:** The `Readme.txt` contains cksum and md5sum of tarballs.

3. Verify the checksums of tarballs. Execute the following command:
   `Readme.txt`
4. Load the tarballs to docker images. Execute the following command:
   `# docker load --input /root/provgw-images-1.6.0.tar`
5. Check if all the images are loaded. Execute the following command:
   `docker images | grep provgw`
6. Tag the docker images to docker registry. Execute the following command:
   `docker tag <image-name>:<image-tag> <docker-repo>/<image-name>:<image-tag>`

   Sample Commands:

   ```
   # docker tag provgw/prov_gw:1.6.0 <customer repo>/provgw/prov_gw:1.6.0
   # docker tag provgw/ocingress_gateway:1.6.2 <customer repo>/
   ocingress_gateway:1.6.2
   # docker push <customer repo>/ocingress_gateway:1.6.2
   # docker tag provgw/configurationinit:1.1.1 <customer repo>/
   configurationinit:1.1.1
   # docker push <customer repo>/configurationinit:1.1.1
   # docker tag provgw/configurationupdate:1.1.1 <customer repo>/
   configurationupdate:1.1.1
   # docker push <customer repo>/configurationupdate:1.1.1
   ```

7. Push the docker images to docker registry. Execute the following command:
   `docker push <docker-repo>/<image-name>:<image-tag>`
8. Untar Helm Files. Execute the following command:
   `tar -xvzf provgw-1.6.0.tgz`
9. Download the Provisioning Gateway Custom Template ZIP file from OHC. The steps are as follows:
   a. Go to the URL, docs.oracle.com
   b. Navigate to **Industries**->**Communications**->**Cloud Native Core**.
   c. Click the ProvGateway Custom Template link to download the zip file.
   d. Unzip the template to get provgw-custom-configtemplates-1.6.0.0 file that contains the following:
      - **ProvGW_Dashboard.json:** This file is used by grafana.
      - **prov-gw5g-custom-values-1.6.0.yaml:** This file is used during installation.
      - **rollback.py**
      - **upgrade.py**
      - **rollbackPCFschema_15_3.py**
      - **UDR_Dashboard.json**
      - **ProvGw_Dashboard.json**

Following are the ProvGateway Images.

Pod	Image
<helm_release_name>-prov-gw	provgw/prov_gw
<helm_release_name>-ocingress_gateway	provgw/ocingress_gateway
	provgw/configurationinit
	provgw/configurationupdate

3

Customizing Provisioning Gateway

In this section, you will learn to customize Provisioning Gateway deployment. You can customize it by overriding the default values of various configurable parameters.

A ProvGateway Customization file is given below:

```
# Copyright 2019 (C), Oracle and/or its affiliates. All rights reserved.

global:
  dockerRegistry: reg-1:5000
  serviceAccountName:
  prefix:
    container:
    configmap:
    hpa:

prov-gw:
  image:
    name: provgw/prov_gw
    tag: 1.6.0
    pullPolicy: Always

  service:
    type: ClusterIP
    port:
      https: 5002
      http: 5001
      management: 9000

  deployment:
    replicaCount: 2

  logging:
    level:
      root: "WARN"

  resources:
    limits:
      cpu: 3
      memory: 4Gi
    requests:
      cpu: 3
      memory: 4Gi
    target:
      averageCpuUtil: 80

  server:
    redirect:
      http: false
    http2enabled: true

  udr:
    segs:
      - ocudr-ingressgateway.ocudr
      - ocudr-ingressgateway.ocudr

  minReplicas: 2
  maxReplicas: 4

prov-ingressgateway:
 global:
   # Docker registry name
   # dockerRegistry: reg-1:5000/ocudr

   # Specify type of service - Possible values are :- ClusterIP, NodePort,
LoadBalancer and ExternalName
   type: LoadBalancer

   # Enable or disable IP Address allocation from Metallb Pool
   metalLbIpAllocationEnabled: true

   # Address Pool Annotation for Metallb
   metalLbIpAllocationAnnotation: "metallb.universe.tf/address-pool:
signaling"

   # If Static node port needs to be set, then set staticNodePortEnabled
flag to true and provide value for staticNodePort
   #   # Else random node port will be assigned by K8
   staticNodePortEnabled: false
   staticHttpNodePort: 30075
   staticHttpsNodePort: 30043

 image:
   # image name
   name: provgw/ocingress_gateway
   # tag name of image
   tag: 1.6.2
   # Pull Policy - Possible Values are:- Always, IfNotPresent, Never
   pullPolicy: Always

 initContainersImage:
   # inint Containers image name
   name: provgw/configurationinit
   # tag name of init Container image
   tag: 1.1.1
   # Pull Policy - Possible Values are:- Always, IfNotPresent, Never
   pullPolicy: Always

 updateContainersImage:
   # update Containers image name
   name: provgw/configurationupdate
   # tag name of update Container image
   tag: 1.1.1
   # Pull Policy - Possible Values are:- Always, IfNotPresent, Never
  pullPolicy: Always

service:
  ssl:
    tlsVersion: TLSv1.2

    privateKey:
      k8SecretName: provgw-ingress-secret
      k8NameSpace: provgw
      rsa:
        fileName: rsa_private_key_pkcs1.pem
      ecdsa:
        fileName: ssl_ecdsa_private_key.pem

    certificate:
      k8SecretName: provgw-ingress-secret
      k8NameSpace: provgw
      rsa:
        fileName: tmp.cer
      ecdsa:
        fileName: ssl_ecdsa_certificate.crt

    caBundle:
      k8SecretName: provgw-ingress-secret
      k8NameSpace: provgw
      fileName: caroot.cer

    keyStorePassword:
      k8SecretName: provgw-ingress-secret
      k8NameSpace: provgw
      fileName: key.txt

    trustStorePassword:
      k8SecretName: provgw-ingress-secret
      k8NameSpace: provgw
      fileName: trust.txt

    initialAlgorithm: RSA256

# Resource details
resources:
  limits:
    cpu: 3
    memory: 4Gi
  requests:
    cpu: 3
    memory: 4Gi
  target:
    averageCpuUtil: 80

log:
  level:
    root: WARN
    ingress: INFO
    oauth: INFO
```

```
# enable jaeger tracing
jaegerTracingEnabled: false

openTracing :
  jaeger:
    udpSender:
      # udpsender host
      host: "occne-tracer-jaeger-query.occne-infra"
      # udpsender port
      port: 6831
    probabilisticSampler: 0.5


# Number of Pods must always be available, even during a disruption.
minAvailable: 2
# Min replicas to scale to maintain an average CPU utilization
minReplicas: 2
# Max replicas to scale to maintain an average CPU utilization
maxReplicas: 5

# label to override name of api-gateway micro-service name
#fullnameOverride: provgw-endpoint

# To Initialize SSL related infrastructure in init/update container
initssl: false

# Cipher suites to be enabled on server side
ciphersuites:
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
  - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_DHE_RSA_WITH_AES_256_CCM
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

#OAUTH CONFIGURATION
oauthValidatorEnabled: false
nfType: SMF
nfInstanceId: 6faf1bbc-6e4a-4454-a507-a14ef8e1bc11
producerScope: nsmf-pdusession,nsmf-event-exposure
allowedClockSkewSeconds: 0
nrfPublicKeyKubeSecret: nrfpublickeysecret
nrfPublicKeyKubeNamespace: ingress
validationType: strict
producerPlmnMNC: 123
producerPlmnMCC: 346

#Server Configuration for http and https support
#Server side http support
enableIncomingHttp: true
#Server side https support
enableIncomingHttps: false
#Client side https support
```

```
enableOutgoingHttps: false

maxRequestsQueuedPerDestination: 5000
maxConnectionsPerIp: 10

#Service Mesh (Istio) to take care of load-balancing
serviceMeshCheck: false
# configuring routes
routesConfig:
- id: traffic_mapping_rest_group_prov
  uri: http://{{ .Release.Name }}-prov-gw:5001
  path: /**
```

The configurable parameters of Provisioning Gateway are:

Default Helm Release Name :- provgw

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
dockerRegistry	Docker registry from where the images will be pulled	reg-1:5000	Not applicable	
serviceAccountName	Service account name	null	Not Applicable	The serviceaccount, role and rolebindings required for deployment should be done prior to the installation. Use the created serviceaccountname here.
prefix.container	Container configurable prefix	null	Not Applicable	If this is configured with some value, the same will be used as prefix for container names on different pods of ProvGw deployment. If Not configured, release name will be used as preifx.
prefix.configmap	Configmap configurabe prefix	null	Not Applicable	If this is configured with some value, the same will be used as prefix for configmap names. if Not configured, release name will be used as preifx.

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
prefix.hpa	HPA configurable prefix	null	Not Applicable	If this is configured with some value, the same will be used as prefix for HPA names. If Not configured, release name will be used as preifx.

Following table provides parameters for **provgw-service** micro service.

Parameter	Description	Default Value	Range of possible values(if applicable	Notes
image.name	Image name	provgw/prov_gw	Not Applicable	
image.tag	Tag of Image	1.6.0	Not Applicable	
image.pullPolicy	This setting will tell if image needs to be pulled or not	Always	Always IfNotPresent Never	
service.type	ProvGw service type	ClusterIP	ClusterIP NodePort LoadBalancer	The Kubernetes service type for exposing ProvGw deployment Note: Suggested to be set as LoadBalancer (default value) always
service.port.http	HTTP port	5001	Not Applicable	The http port to be used in provGw service
service.port.https	HTTPS port	5002	Not Applicable	The https port to be used in provgw service
service.port.management	Management port	9000	Not Applicable	The Prometheus management port to be used for ProvGw service
deployment.replicaCount	Replicas of provgw pod	2	Not applicable	Number of provgw pods to be maintained by replica set created with deployment

Parameter	Description	Default Value	Range of possible values(if applicable	Notes
logging.level.root	Log Level	WARN	WARN INFO DEBUG ERROR	Log level of the Provisioning gateway pod
server.redirect.http	Enable redirecting HTTP mesagases	false	true/false	
server.http2enabled	Enabled HTTP2 support flag	true	true/false	

*udr.segs	FQDNs of UDR	**Not Applicable	Not Applicable	To be used to send SLF requests to UDRs. This accepts yaml array.
				e.g.
				`udr:`
				` segs:`
				`   -`
				`ocudr1.ingressgateway.ocudr1`
				`   -`
				`ocudr2.intressgateway.ocudr2`

resources.requests.cpu	Cpu Allotment for nudr-drservice pod	3	Not applicable	The cpu to be allocated for prov-gw pod during deployment
resources.requests.memory	Memory allotment for nudr-drservice pod	4Gi	Not applicable	The memory to be allocated for prov-gw pod during deployment
resources.limits.cpu	Cpu allotment limitation	3	Not applicable	
resources.limits.memory	Memory allotment limitation	4Gi	Not applicable	
resources.target.averageCpuUtil	CPU utilization limit for autoscaling	80	Not Applicable	CPU utilization limit for creating HPA
minReplicas	Minimum Replicas	2	Not Applicable	Minimum number of pods
maxReplicas	Maximum Replicas	4	Not Applicable	Maximum number of pods

Following table provides parameters for provgw-ingressgateway micro service (API Gateway).

Note:

(*) - This configuration is mandatory before starting the service.

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
type	provgw-prov-ingressgateway service type	LoadBalancer	Possbile Values- ClusterIP NodePort LoadBalancer	
metalLbIpAllocationEnabled	Enable or disable Address Pool for Metallb	true	true/false	
metalLbIpAllocationAnnotation	Address Pool for Metallb	"metallb.universe.tf/address-pool: signaling"	Not applicable	
staticNodePortEnabled	If Static node port needs to be set, then set staticNodePortEnabled flag to true and provide value for staticNodePort	false	Not applicable	
staticHttpNodePort	static http node port value needs to be provided	30075	can be changed based on user requirement.	
staticHttpsNodePort	static https node port value needs to be provided	30043	can be changed based on user requirement.	
image.name	Docker image name	provgw/ocingress_gateway	Not applicable	
image.tag	Image version tag	1.6.2	Not applicable	
image.pullPolicy	This setting will tell if image need to be pulled or not	Always	Possible Values - Always IfNotPresent Never	
initContainersImage.name	Docker image name	provgw/configurationinit	Not applicable	
initContainersImage.tag	Image version tag	1.1.1	Not applicable	
initContainersImage.pullPolicy	This setting will tell if image need to be pulled or not	Always	Possible Values - Always IfNotPresent Never	

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
updateContainersImage.name	Docker image name	provgw/configurationupdate	Not applicable	
updateContainersImage.tag	Image version tag	1.1.1	Not applicable	
updateContainersImage.pullPolicy	This setting will tell if image need to be pulled or not	Always	Possible Values - Always IfNotPresent Never	
service.ssl.privateKey.k8SecretName	name of the secret which stores keys and certificates	provgw-gateway-secret	Not applicable	
service.ssl.privateKey.k8NameSpace	namespace in which secret is created	provgw	Not applicable	
service.ssl.privateKey.rsa.fileName	rsa private key stored in the secret	rsa_private_key_pkcs1.pem	Not applicable	
service.ssl.privateKey.ecdsa.fileName	ecdsa private key stored in the secret	ecdsa_private_key_pkcs8.pem	Not applicable	
service.ssl.certificate.k8SecretName	name of the secret which stores keys and certificates	`provgw-ingress-secret`	Not applicable	
service.ssl.certificate.k8NameSpace	namespace in which secret is created	provgw	Not applicable	
service.ssl.certificate.rsa.fileName	rsa certificate stored in the secret	apigatewayrsa.cer	Not applicable	
service.ssl.certificate.ecdsa.fileName	ecdsa certificate stored in the secret	apigatewayecdsa.cer	Not applicable	
service.ssl.caBundle.k8SecretName	name of the secret which stores keys and certificates	provgw-ingress-secret	Not applicable	
service.ssl.caBundle.k8NameSpace	namespace in which secret is created	provgw	Not applicable	
service.ssl.caBundle.fileName	ca Bundle stored in the secret	caroot.cer	Not applicable	
service.ssl.keyStorePassword.k8SecretName	name of the secret which stores keys and certificates	provgw-ingress-secret	Not applicable	

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
service.ssl.keyStorePassword.k8NameSpace	namespace in which secret is created	provgw	Not applicable	
service.ssl.keyStorePassword.fileName	keyStore password stored in the secret	key.txt	Not applicable	
service.ssl.trustStorePassword.k8SecretName	name of the secret which stores keys and certificates	`provgw-ingress-secret`	Not applicable	
service.ssl.trustStorePassword.k8NameSpace	namespace in which secret is created	provgw	Not applicable	
service.ssl.trustStorePassword.fileName	trustStore password stored in the secret	trust.txt	Not applicable	
resources.limits.cpu	Cpu allotment limitation	3	Not applicable	
resources.limits.memory	Memory allotment limitation	4Gi	Not applicable	
resources.requests.cpu	Cpu allotment for provgw-prov-ingressgateway pod	3	Not Applicable	
resources.requests.memory	Memory allotment for provgw-prov-ingressgateway pod	4Gi	Not Applicable	
resources.target.averageCpuUtil	CPU utilization limit for autoscaling	80	Not Applicable	
minAvailable	Number of pods always running	2	Not Applicable	
minReplicas	Min replicas to scale to maintain an average CPU utilization	2	Not applicable	
maxReplicas	Max replicas to scale to maintain an average CPU utilization	5	Not applicable	
log.level.root	Logs to be shown on provgw-prov-ingressgateway pod	WARN	valid level	
log.level.ingress	Logs to be shown on provgw-prov-ingressgateway pod for ingress related flows	INFO	valid level	

Parameter	Description	Default value	Range or Possible Values (If applicable)	Notes
log.level.oauth	Logs to be shown on provgw-prov-ingressgateway pod for oauth related flows	INFO	valid level	
initssl	To Initialize SSL related infrastructure in init/update container	true	Not Applicable	
jaegerTracingEnabled	Enable/Disable Jaeger Tracing	false	true/false	
openTracing.jaeger.udpSender.host	Jaeger agent service FQDN	jaeger-agent.cne-infra	Valid FQDN	
openTracing.jaeger.udpSender.port	Jaeger agent service UDP port	6831	Valid Port	
openTracing.jaeger.probabilisticSampler	Probablistic Sampler on Jaeger	0.5	Range: 0.0 - 1.0	Sampler makes a random sampling decision with the probability of sampling. For example, if the value set is 0.1, approximately 1 in 10 traces will be sampled.
oauthValidatorEnabled	OAUTH Configuration	false	Not Applicable	
enableIncomingHttp	Enabling for accepting http requests	true	Not Applicable	
enableIncomingHttps	Enabling for accepting https requests	true	true or false	
enableOutgoingHttps	Enabling for sending https requests	false	true or false	
maxRequestsQueuedPerDestination	Queue Size at the provgw-prov-ingressgateway pod	5000	Not Applicable	
maxConnectionsPerIp	Connections from endpoint to other microServices	10	Not Applicable	
routesConfig	Routes configured to connect to ProvGw	`- id: traffic_mapping_rest_group_prov uri: http://{{ .Release.Name }}-prov-gw:5001 path: /**`	Not Applicable	

4

Upgrading an Existing ProvGateway Deployment

In this section, you will learn to upgrade an existing ProvGateway Deployment.

Upgrading an existing deployment replaces the running containers and pods with new ones. If there is no change in the pod configuration, the pods are not replaced. Unless there is a change in the service configuration of a microservice, the service endpoints remain unchanged (ClusterIP etc.).

Note:

You should stop the provisioning traffic while upgrading the ProvGateway and then, perform the helm upgrade.

Helm Upgrade
To upgrade ProvGateway via Helm:

- Follow the deployment instructions provided in the Installation Sequence section for extracting the required ProvGateway software components and if required re-tag and push the images to customer's repository.
- Take a backup of the provgw-custom-values-1.6.0.yaml file and modify the parameters as per site requirement.
- Execute the following command to upgrade an existing ProvGateway deployment. For the parameters that are configurable, see Customizing Provisioning Gateway.

  ```
  $ helm upgrade <release> <helm chart> [--version <ProvGw version>] -f
  <provgw-custom-values-1.6.0.yaml>
  ```

  In the above command:

  – **<release>** information is available in the output of 'helm list' command.

  – **<chart>** is the name of the chart in the form of <repository/provgw> .
  Example: reg-1/provgw or cne-repo/provgw

Rollback Instructions
As it is an initial release of ProvGateway, rollback feature is not applicable. However, you can use `helm del` on the newly created task.

5
Uninstalling Provisioning Gateway

In this section, you will learn to uninstall Provisioning Gateway.

To uninstall or completely delete the provisioning gateway deployment, execute the following command:

```
helm del --purge <helm_release_name_for_provgw>
```

Note:

If helm3 is used, then execute the following command:

```
helm uninstall <helm_release_name_for_provgw> --namespace
<provgw_namespace>
```