Genesys Engage cloud private edition

EAP deployment and provisioning instructions

Disclaimer

Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Table of contents

Common prerequisites

> **Disclaimer**
>
> Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Provide an appropriate set of permissions to a select group of users responsible for deploying and managing Genesys software. The permissions should be as limited as possible.

Security Context Constraints (SCC) are the tool provided by OpenShift to control which privilege being requested by a pod is allowed on the platform. The `genesys-restricted` SCC is created in compliance with OpenShift best practices for cluster role management and deployment.

OpenShift comes with eight predefined SCCs, and by default all pods and containers will use the most restrictive "restricted" SCC.

Note: In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs.

The `genesys-restricted` SCC

The `genesys-restricted` SCC:

- Has access linked specifically to user `genesys` and the group `genesys-restricted-group`.
- Expands volume types to allow pods to work with different infrastructure needs.
- Instead of using the default behavior of arbitrary userID, the `genesys-restricted` SCC allows for containers to specify the user/id (genesys /500), which is what Genesys containers assume will be the case.
- Does not allow for privilege escalation but does allow for predictable user and id.
- Is based on the default "restricted" SCC.

Creating and using the SCC

The following steps require ClusterAdmin privileges and should be done prior to deployment of any Genesys services.

1. Create a custom SCC. See Details of the genesys-restricted SCC, below.

   Create SCC

   ```
   $ oc create -f genesys-restricted.yaml
   ```

2. Create ClusterRole to use SCC.

   Create ClusterRole to use SCC

   ```
   $ oc create clusterrole genesys-restricted-scc --verb=use --resource=securitycontextconstraints.security.
   openshift.io --resource-name=genesys-restricted
   ```

3. Assign ClusterRole to User Group.

   Assign ClusterRole to UserGroup

   ```
   $ oc adm policy add-cluster-role-to-group genesys-restricted-scc genesys-restricted-group
   ```

When deploying a specific service, users should execute the following command prior to service deployment.

Add SCC to serviceAccount

```
$ oc adm policy add-scc-to-user genesys-restricted -z <serviceAccount> -n <namespace>
```

Tip: You could define these actions within a single helm chart for ease of use.

Update volumes

Make sure to update volumes to list the objects that your cluster uses (depends on target deployment platform).

Verify the created SCC:

Verify created SCC (oc describe scc genesys-restricted)

```
Name: genesys-restricted
Priority: <none>
Access:
Users: genesys
Groups: genesys-restricted-group
Settings:
Allow Privileged: false
Allow Privilege Escalation: true
Default Add Capabilities: <none>
Required Drop Capabilities: KILL,MKNOD,SETUID,SETGID
Allowed Capabilities: <none>
Allowed Seccomp Profiles: <none>
Allowed Volume Types: configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret, <infrastructure
volume type 1, e.g. azureFile>, <infrastructure volume type 2, e.g. azureDisk>
Allowed Flexvolumes: <all>
Allowed Unsafe Sysctls: <none>
Forbidden Sysctls: <none>
Allow Host Network: false
Allow Host Ports: false
Allow Host PID: false
Allow Host IPC: false
Read Only Root Filesystem: false
Run As User Strategy: MustRunAsRange
UID: <none>
UID Range Min: 500
UID Range Max: 65535
SELinux Context Strategy: MustRunAs
User: <none>
Role: <none>
Type: <none>
Level: <none>
FSGroup Strategy: MustRunAs
Ranges: 500-65535
Supplemental Groups Strategy: MustRunAs
Ranges: 500-65535
```


Details of the `genesys-restricted` SCC

genesys-restricted.yaml

```
kind: SecurityContextConstraints
metadata:
  name: genesys-restricted
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  ranges:
  - max: 65535
    min: 500
  type: MustRunAs
groups:
- genesys-restricted-group
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: MustRunAsRange
  uidRangeMax: 65535
  uidRangeMin: 500
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  ranges:
  - max: 65535
    min: 500
  type: MustRunAs
users:
- genesys
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
- <infrastructure volume type 1, e.g. azureFile>
- <infrastructure volume type 2, e.g. azureDisk>
```

Deploy Genesys Authentication

> **Disclaimer**
>
> Genesys Authentication in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Genesys Authentication provides authentication and authorization capabilities for Genesys Engage cloud private edition services and applications. Genesys Authentication supports OAuth 2.0 authorization flows, single-sign on, and includes a user interface for logging in to Genesys Engage applications.

To deploy Genesys Authentication in your environment, you must:

1. Complete the prerequisites.
2. Prepare your environment.
3. Install the gauth package.

Prerequisites

Install third-party services

The following third-party services are required to deploy Genesys Authentication:

- Redis
  - URL and Port
  - Credentials
- Postgres
  - Address (URL)
  - DB Name
  - User (should have admin rights to create/update/delete schemas)
  - Password

Add JKS support

Create a key store file:

```
keytool -keystore idp_keystore.jks -genkey -alias gws-auth-key -storepass <password> -keypass <password> -
keyalg RSA
```

Get the Base64 encoded key:

```
cat ./idp_keystore.jks | base64
```

The result looks like this:

```
/u3+7QAAAAIAAAABAAAAAQAMZ3dzLWF1dGgta2V5AAABeRmB2Y4AAAUBMIIE/TAOBgorBgEEASoCEQEBBQAEggTpwQQ5aW5CUYAsf4
/IheBuNrlPPyZhUA+NWh3SG52HV3sVjV+p18vKp2k/q12I9NynoM6R
/DW5bFfEWU1zx3cfXH2kNirRUOIbNZpa43NOroyyF1GSdZFlwa8Kq8Xtp8ZBmiJdSb1n12ODaTKGKv1cb5tsfdzkWs99QeTBGJypHMCdnBvdFB0N
vid+yaxdmK3bdQlCwDgDOHZ2RnU0oxXebH3o+jIrrowyo7DidJDyUwIeT2PPPI9F88QsP+CdDRRR1T7t0ojd1hIJ8YYMwJ1wQmwgF4QZRbtrEnwX
qkHinJnwOERcp4FLuceds6YTlcxvWsS+GEv38Jv8YokLwRb/mMACTHk4R9yASsd7fljgNLSn0jhrz9FuxvYgpOVvExiq+sb5YrfbZjtTzZDzFVOu
/2kWzASfZBSiyyxMOr3IhUPkMpIrg+UYkI0tgn
/C3yR1wLr9HElpx8fCu61ORqp8hhp1yvL46K0c6eTa2JcRpO6fmysf2EG0JagG7zNEJHlvtNnt3JpQV06xos2iWsFAtHq+9w8LwvCVbDzx
/UHoCYenIdJ7SBv06mXgKisa3RDIi/y5x5/9T4brgCLUvwI4Z5Rf/oi2Zx5/lXjQXmBPlPAcUVHLr5PvNQUUx5NBr/ooioD7qka4ADF1
/cx8I2bzqTi+U01fiFdMGRlNlCfcGDMI2h82JUeCswRYi4+dMDiSaGgC2MoL2susLxMYa5CTo9Vs0Y2k+6j8fhIO4h8h0JxdXZODU63OM0cDSUHX
fbyKSey/4IhiV3k7W4OHYeXUeDvoNmfo/AriELZl+WgYETiXGsKzxmrsHrBKC0+aT098FwqdY9ACsM/7WoF2+9eftc7fa2jruutrRjmk0A
/BaIqzboJLFiWaUUGV9gsexEmpGszikQsmOYSIRxY8BYF+SYldehcfcsRRxDnhTaGNV8y2ZnwA61FNPAFps3gaFXeaYsUzlxTSi9m70HJJrUp7JD
K6SGg6luiKMG4O7QjsGgOOwGpoLJf7EFOCspN3t4damhH
/KFi9OrEuAdhMJa+iQ21PBZ+iIwxb0y9xMReImoUtoqy6Epre3qMOS6MILLw2bVrxJYo38+hR5uzNdlbsUlpYOoorI1Hp8A
/VEYtG9PDHEhhoqUamdUYUzkFDi9QZfylIgi8Jc4G4PPrPKgMPqqE7sl6bJvoLavU58eHpdWo/Mb9UtdTx+l
/SlulCCE0Xce6M9YE1SyC2B3gd82zNQa81lx+QAY8IaSmX+C2nMz+UeXKngSEzguK6gXg9RwCs8pUavuLQ6uZGkJ+fhDBvDAFgD7hG1XdHs27XGS
UsRq6GiiwmjZXsZ70ETIVmXfuSvGuYYpv4CKzIDvweGdbkUWap2oQqm6Sw+OkJLbim1aW4MSYGCZItLOM36108onELop0wTIMiZBv2cIaQ+WzbyK
DXIi94ebntxu42GSeUn1IGDMGAa+vh4itLDV6yIXCRHCd9GM6JBj+SKXkn+J0FIZ257Kf4k3kg6rm9Ha1800NMB7ILrbJChCZd5bmDmMwUGsNrla
r2Oa4S41Y81vmBQJlMzSvAS37gk3eFX3XLKtfn7+Dxq8aYRA0TQEMln0uUuoiNRZH8iaLwhpI4bEkaoSU/DT/KHRB7AHN5
/vQpj6KOscxqmyPrgPY
/+TseczEeaQLQ6MfjvXY+AAAAAQAFWC41MDkAAAN7MIIDdzCCAl+gAwIBAgIEYxhLHTANBgkqhkiG9w0BAQsFADBsMRAwDgYDVQQGEwdVbmtub3d
uMRAwDgYDVQQIEwdVbmtub3duMRAwDgYDVQQHEwdVbmtub3duMRAwDgYDVQQKEwdVbmtub3duMRAwDgYDVQQLEwdVbmtub3duMRAwDgYDVQQDEwd
Vbmtub3duMB4XDTIxMDQyODE3MjMzMFoXDTIxMDcyNzE3MjMzMFowbDEQMA4GA1UEBhMHVW5rbm93bjEQMA4GA1UECBMHVW5rbm93bjEQMA4GA1U
EBxMHVW5rbm93bjEQMA4GA1UEChMHVW5rbm93bjEQMA4GA1UECxMHVW5rbm93bjEQMA4GA1UEAxMHVW5rbm93bjCCASIwDQYJKoZIhvcNAQEBBQA
DggEPADCCAQoCggEBAK0300E4F3NDKRHYwrkqDgeUPwoIrozyLp6JvkCsOe1nj5L44vwJjnrYw9falOVDQ9ggwquwXoD5RNEf5
/esYcJNEqu1btJLwLvhXb651OyZnsmeNGP2BrNCPXZS6CBReMMKJaZrlCwJQxiSrGPHB/gpxKoAowLwl3V7wB2BHKDhrczQBPdvtsfBAzeqpN
/yRpdKZRAtu2LyGqRZKCglSrwYenJFqROdOeworbNmtIKXfQLiamE4KdhzQdPfnyBC7ZWtCIJUp9Va4LmCYD
/ISOmVyfQ9Xql1rRNQLcVaewCKRM2ffBAkx98d3n79XUZDljOzHh+79tCpheuuYfbMQqMCAwEAAaMhMB8wHQYDVR0OBBYEFNtM8mIEb67VYot5tj
DAdrhiq+31MA0GCSqGSIb3DQEBCwUAA4IBAQBued3SqK+1chAUpNz06Ceak2Ldzlz6MXxBZx6fx50odSY3C4rwywK31RSAAtCe
/Ta4y+B6JcdPjFtII6Pf5W0DDTOa3cHNMeukYn5lBnaMbIKqoxFT7nM7MD3DB+dISvMu8FtVWFwbPzXWhl+Aycuu9ETGlCoJqYfl+vmLyGjJVadc
Ug4E3u7b29woO+bH9pagJErfi7Wq0vaaqMtkKjfOSovMTwz5ruXSiCDZlxb8C8CEr9lkpT3F0G1XFaNNOHoZiZNwXFKLoLrdI2t3pOCYyhwYbGKG
MyitRguYz490dxZ0G5R4kMo/YbN7be2QIJwmucIZzH7fkU90V+rmVZhl9Bo8ixuIJG/vZTxmEBaDqmhiP4w=
```

Prepare your environment

Confirm the cluster is running:

```
oc get clusterversion
```

Create a new project:

```
oc new-project gauth
```

Enable a security context for the default service account:

```
oc adm policy add-scc-to-user genesys-restricted -z default -n gauth
```

Download the Helm charts from the **gauth** folder in the JFrog repository.

Update **values_gauth.yaml:**

```
1. Image details under 'image'

image:
  registry: <docker-repo>

2. Postgres DB details under 'postgres'

postgres:
  deploy: false
  secret:
    name_override:
    create: true
```

```
  configmap:
    name_override:
    create: true
  username: postgres -----> user name of the Postgres DB
  password: <password> -----> password of the Postgres DB
  db: postgres -----> DB name for gauth. It should be already created.
  host: postgres-rw.infra01.svc.cluster.local -----> service URL
  port: 5432 -----> port

3. Redis details under 'redis'

redis:
  secret:
    name_override: gauth-redis
    create: true
  configmap:
    name_override:
    create: true
  image: redis:5-stretch
  cluster_nodes: infra01-redis-redis-cluster.infra01.svc.cluster.local:6379 -----> Redis URL and port
  use_tls: false
  password_required: true
  password: K9QvHnqQUr -----> password

4. Set the replica count and ops admin credential details under 'services'

services:
  replicas: 3
  location: /USW1
  secret:
    name_override:
    create: true
    admin_username: ops -----> user name
    admin_password: $2a$10$7.jOOym2GhhVtpSPXOQmweVqY4gLKOXFbNDRul7UMeP40kQVVx/au -----> encrypted password 'ops'
    client_id: external_api_client
    client_secret: RpV+KfuwZMf+yh7IOJDz+56XgRoBFMExKkycBi7GWNY=

5. Update JKS details under 'auth'

auth:
  jks:
    enabled: true
  sso:
    enabled: false
  secret:
    create: true
  # provide a name secret if should use existing
    name:

  keyStore: jksStorage.jks
  keyStoreFileData: |-
  <Base64 encoded key value from the prerequisite step to add JKS support>
  # Keystore password
  keyStorePassword: <password>
  # SSL key alias
  keyAlias: gws-auth-key
  # SSL key password
  keyPassword: <password>

6. Specify the versions for each service
gws-ui-auth: 9.0.000.35
gws-core-environment: 9.0.000.68
gws-core-auth: 9.0.000.70
```


Copy **values_gauth.yaml** and the Helm package (**gauth-0.1.77.tgz**) to the installation location.

Install gauth

Log in to the OpenShift cluster from the host where you will run the deployment:

```
oc login --token <token> --server <URL of the API server>
```

Select the gauth project you created in Prepare the environment:

```
oc project gauth
```

For debugging purposes, use the following command to render templates without installing so you can check that resources are created properly:

```
helm template --debug /gauth-0.1.77.tgz -f values-gauth.yaml
```

The result shows Kubernetes descriptors. The values you see are generated from Helm templates, and based on settings from **values_gauth.yaml**. Ensure that no errors are displayed; you will later apply this configuration to your Kubernetes cluster.

Deploy gauth:

```
 helm install gauth ./gauth-0.1.77.tgz -f values-gauth.yaml
```

Check the installed Helm release:

```
helm list
```

The results should show the gauth deployment details. For example:

```
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART
APP VERSION
gauth   gauth           1               2021-05-20 11:56:32.5531685 +0530 +0530 deployed        gauth-0.1.77
0.1
```

Check the gauth project status:

```
helm status gauth
```

The result should show the project details with a status of deployed.

```
NAME: gauth
LAST DEPLOYED: Thu May 20 11:56:32 2021
NAMESPACE: gauth
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Check the gauth OpenShift objects created by Helm:

```
oc get all -n gauth
```

The result should show all the created pods, service configmaps, and so on.

Expose gauth services

Make gauth services accessible from outside the cluster, using the standard HTTP port. Make sure to use the same hostname for all three routes. Genesys recommends using the following host format: `gauth.<cluster-subdomain>`. For example, the VCE cluster **(https://console-openshift-console.apps.<yourclusterdomain>.com/)** should have the hostname **gauth.apps.<yourclusterdomain>.com**

```
oc create route edge --service=<env-service> --hostname=<hostname> --path /environment
oc create route edge --service=<gauth-service> --hostname=<hostname> --path /auth
oc create route edge --service=<gauth-auth-ui-service> --hostname=<hostname> --path /ui/auth
```

Verify the new route is created in the gauth project:

```
oc get route -n gauth
```

The result includes the following information about the services:

```
NAME            HOST/PORT                                          PATH                SERVICES
PORT   TERMINATION   WILDCARD
env             gauth.apps.<yourclusterdomain>.com   /environment   gauth-environment   https
None
gauth           gauth.apps.<yourclusterdomain>.com   /auth          gauth-auth          https
None
gauth-auth-ui   gauth.apps.<yourclusterdomain>.com   /ui/auth       gauth-auth-ui       https
None
```

Note: `HOST` is the host name generated by OpenShift.

Verify that you can now access Genesys Authentication at the following URL: https://<hostname>/ui/auth/sign-in.html

Uninstall gauth

To remove gauth:

```
helm uninstall gauth -n gauth
```

Deploy Agent Setup

Disclaimer

Agent Setup in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Context

Agent Setup does not have its own container; it is delivered as part of Genesys Web Services and Applications (GWS) and must be enabled when deploying GWS services.

Agent Setup controls your contact center and its resources:

- The **people** who run and operate it – that's the **administrators** who control the technical ins and outs, the **managers** who run the day-to-day operations and administrative aspects of a contact center, the **supervisors** who oversee agents, and the **agents** who communicate with customers.
- The **systems and programs** that make the day-to-day stuff possible – that's the telephony, the software, the servers, the routing and dialing strategies, and so on.
- The **features and capabilities** we use to meet our business needs and requirements - those are things like Caller ID capabilities, voicemail, agent transfers and conferencing, and so on.

This topic describes how to **deploy** Agent Setup in your environment.

Deploy Agent Setup

To deploy Agent Setup, you must:

1. Update the value overrides
2. Update the version overrides

Update the value overrides

From the gws-services helm charts, update the following lines in the value overrides under the `gwsServices` > `appProvisioning` > `context` > `env` section before installing GWS:

- GWS_SERVICE_AUTH_URL: Auth internal service URI from gauth namespace (for example, http://gauth-auth.gauth.svc.cluster.local.:80)
- GWS_SERVICE_ENV_URL: Environment internal service URI from gauth namespace (for example, http://gauth-environment.gauth.svc.cluster.local.:80)
- GWS_SERVICE_CONF_URL: gws internal service URI from gws namespace (for example, http://gws-service-proxy.gws.svc.cluster.local:80)
- GWS_PROVISIONING_SERVICES_AUTH_FOR_REDIRECT : External https ingress URLS from gauth service(ex: https://gauth.<yourclusterdomain.com>)
- GWS_PROVISIONING_OBJECTCACHE_POSTGRES_USER: <Postgres DB user for provisioning service>
- GWS_PROVISIONING_OBJECTCACHE_POSTGRES_PASSWORD: <Postgres DB password for provisioning service>
- GWS_PROVISIONING_OBJECTCACHE_POSTGRES_HOST: <Postgres DB host for provisioning>
- GWS_PROVISIONING_OBJECTCACHE_POSTGRES_PORT: <Postgres DB Port for provisioning >

Example:

```
gwsServices:
   uiProvisioning:
    # Name of deployment
    name: gws-ui-provisioning
    # App type
    appType: ui
    # Service to wait before start
    waitService: gws-core-environment
    # Whether do liveness probe or not
    doLivenessProbe: false
    # Discovery tags
    discoveryTags: ""
```

```
  deployment:
    # Number of pod replicas
    replicaCount: 2
    # Can use a custom postfix other than green/blue/canary
    postfix: "GWS_UI_PROVISIONING"
  image:
    # Name of docker registry repository
    repository: "gws-ui-provisioning"
    # Docker image extra information
    tag: ""
  resources:
    limits:
      # Limits for cpu
      cpu: 1
      # Limits for memory
      memory: 0.5Gi
    requests:
      # Requests for cpu
      cpu: 0.5
      # Requests for memory
      memory: 0.5Gi
  context:
    # Whether create context or not
    create: true
    # Config name
    configname: gws-ui-provisioning-config
    env: {}
  service:
    enabled: true
    port:
      # Srv port
      srv: 50040
      # Mgmt port
      mgmt: 50040
appProvisioning:
  # Name of deployment
  name: gws-app-provisioning
  # App type
  appType: nodejs
  # Service to wait before start
  waitService: gws-core-environment
  # Whether do liveness probe or not
  doLivenessProbe: true
  # Discovery tags
  discoveryTags: contextPath=/ui/wwe
  deployment:
    # Number of pod replicas
    replicaCount: 2
    # Can use a custom postfix other than green/blue/canary
    postfix: "GWS_PROVISIONING"
  image:
    # Name of docker registry repository
    repository: "gws-app-provisioning"
    # Docker image extra information
    tag: ""
  resources:
    limits:
      # Limits for cpu
      cpu: 2
      # Limits for memory
      memory: 2Gi
    requests:
      # Requests for cpu
      cpu: 0.5
      # Requests for memory
      memory: 2Gi
  context:
    # Whether create context or not
    create: true
    # Config name
    configname: gws-app-provisioning-config
```

```
    env:
      GWS_PROVISIONING_ASYNC_IO_ENABLED: true
      GWS_PROVISIONING_COMET_ENGINE: cometd
      GWS_PROVISIONING_COMMON_LOGLEVEL: info
      GWS_PROVISIONING_OBJECTCACHE_DB_TYPE: postgres
      GWS_PROVISIONING_CONSUL_DISCOVERY_CONFIGSERVICE_NAME: gws-platform-configuration
      GWS_SERVICE_AUTH_URL: http://gauth-auth.gauth.svc.cluster.local.:80
      GWS_SERVICE_ENV_URL: http://gauth-environment.gauth.svc.cluster.local.:80
      GWS_SERVICE_CONF_URL: http://gws-service-proxy.gws.svc.cluster.local:80
      GWS_SERVICE_VOICEMAIL_URL: http://voice-voicemail-service.voice.svc.cluster.local.:8081/fs
      GWS_PROVISIONING_SERVICES_AUTH_FOR_REDIRECT: https://gauth.apps.<yourclusterdomain>.com:443
      GWS_PROVISIONING_OBJECTCACHE_POSTGRES_USER: prov
      GWS_PROVISIONING_OBJECTCACHE_POSTGRES_PASSWORD: prov
      GWS_PROVISIONING_OBJECTCACHE_POSTGRES_HOST: postgres-rw.infra01.svc.cluster.local
      GWS_PROVISIONING_OBJECTCACHE_POSTGRES_PORT: 5432
      GWS_PROVISIONING_OBJECTCACHE_POSTGRES_USE_SSL: false
      GWS_SECURE_COOKIE: false
  service:
    enabled: true
    port:
      # Srv port
      srv: 48060
      # Mgmt port
      mgmt: 48061
```

Update the version overrides

From the gws-services helm charts, update the following lines in the version overrides with latest version for provisioning.

Example:

```
gws-app-provisioning: 9.0.000.95.5045

gws-ui-provisioning: 9.0.000.84.8957
```

Next steps

- Proceed to **GWS services installation** so all related services to `gws-app-provisioning` and `gws-ui-provisioning`, as well as `pods` are deployed, then continue to **Provision Agent Setup**.

Deploy Genesys Web Services and Applications

Disclaimer

Genesys Web Services and Applications (GWS) in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Context

Genesys Web Services and Applications (GWS) must be deployed along with Agent Setup and Genesys Authentication.

Genesys Web Services (GWS) is an application cluster composed of several microservices that run together. GWS runs on multiple containers that are categorized as below:

- Data Services: These services use multiple data sources (third-party databases) that you must maintain to store GWS data.
- Platform Services: These services are used to connect to Genesys servers such as Configuration Server, Stat Server, SIP Server, and Interaction Server.
- Core Services: These services are used for Web Services and Applications configuration and authentication.
- UI Services: These services provide user interfaces (Workspace Web Edition and the authentication UI) and the underlying services needed to support them, such as the Workspace Service.
- Client Application: This can be Workspace Web Edition (WWE) Agent Desktop, a custom desktop, or Gplus Adapter for Salesforce.

A reverse proxy service is used as an ingress controller. This works as an internal application load balancer

This topic describes how to **deploy** GWS in your environment.

Deploy GWS

Prerequisites

Install third-party services

The following third-party services are required to deploy GWS:

- Consul
  - URL
  - Consul-gws-token
- Redis
  - URL and Port
  - Credentials
- Postgres
  - Address (URL)
  - DB Name
  - User (should have admin rights to create/update/delete schemas)
  - Password
- Elasticsearch
  - URL

Deploy Genesys Authentication

The common Authentication Service must be deployed first. See Deploy Genesys Authentication.

Secret configuration for pulling image

You might already have you secret created.

One of the way to do it is by using the following command:

```
oc create secret docker-registry <credential-name> --docker-server=<dcker repo> --docker-username=<username> --docker-password=<password> --docker-email=<emailid>
```

You have to execute the following command to map the secret to the default service account:

```
oc secrets link default <credential-name> --for=pull
```

Prepare your environment

Check the cluster

Run the following command to get the version of the cluster:

```
oc get clusterversion
```

Create a new project

Use the following command to create a new project:

```
oc new-project gws
```

Enable security context

Use the following command to enable the security context to the default service account:

```
oc adm policy add-scc-to-user genesys-restricted -z default -n gws
```

Download GWS helm charts

Download the GWS helm charts from JFrog using your credentials.

Create two API clients

Create the following two API clients on Genesys Authentication using the Create a new API Client procedure.

1. API Client for gws

- **name**: gws-app-workspace (Note: Name should not be changed)
- **client_Id**: gws-app-workspace (Note: Client ID should not be changed)
- **client_secret**: <Your password> - default password is 'secret'

Record the 'encrypted_client_secret' as it is used to create your secret.

2. API Client for provisioning (Agent-setup)

- **name**: gws-app-provisioning (Note: Name should not be changed)
- **client_Id**: gws-app-provisioning (Note: Client ID should not be changed)
- **client_secret**: <secret>

Record the 'encrypted_client_secret' as it is used to create your secret.

Create Secrets

Add the following lines to the value override file to have Helm create secrets during deployment:

```
secrets:
        gws-consul-token: <token-from consul>
        gws-postgres-username: <gws postgres DB username>
        gws-postgres-password: <gws postgres DB password>
        ops-user: <ops user>
        ops-pass-encr: <ops password>
        agentsetup-postgres-username: <prov postgres username>
        agentsetup-postgres-password: <prov postgres password>
        gws-app-workspace-encrypted: <secret(encrypted) for gws-app-workspace client>
        gws-app-provisioning-encrypted: <secret(encrypted) for gws-app-provisioning client>
```

Update parameters in values.yaml

In the values.yaml file provided by Genesys, update following parameters:

```
Image repo details:
        REGISTRY: <docker-repo>
Postgres:
        POSTGRES_ADDR: Postgres service DB URL
        POSTGRES_DB: Postgres DB name for gws service
        POSTGRES_USER: Postgres user to access gws DB
        POSTGRES_PASS: Postgres Password

Redis:
        REDIS_ADDR: Address of the Redis cluster
        REDIS_PORT: Redis Port

elastic:
        ELASTICSEARCH_ADDR: Elastic search service master address
        ELASTICSEARCH_PORT: Port of ES service

Authentication service configurations:
        Add/update below variables in env section of all services under 'gwsServices'
                GWS_SERVICE_AUTH_URL: http://gauth-auth.gauth.svc.cluster.local.:80 // Genesys Authentication
variable - pointes to internal auth service URL from gauth namesapce, Example: http://gauth-auth.gauth.svc.
cluster.local.:80
                GWS_SERVICE_ENV_URL: http://gauth-environment.gauth.svc.cluster.local.:80 // Environment
variable pointes to internal environment service URL from gauth namesapce, Example: http://gauth-environment.
gauth.svc.cluster.local.:80
                GWS_WORKSPACE_SERVICES_ENV: http://gauth-environment.gauth.svc.cluster.local.:80 // Environment
variable - pointes to internal environment service URL from gauth namesapce, Example: http://gauth-environment.
gauth.svc.cluster.local.:80
                GWS_WORKSPACE_SERVICES_AUTH: http://gauth-auth.gauth.svc.cluster.local.:80 // Genesys
Authentication variable - should be pointed to internal auth service URL from gauth namesapce, Example:
http://gauth-auth.gauth.svc.cluster.local.:80
                GWS_WORKSPACE_SERVICES_AUTH_FOR_REDIRECT: https://gauth.<yourclusterdomain>.com //Genesys
Authentication redirect variable - pointes to external https ingress URL from gauth namesapce, Example:
https://gauth.apps.<yourclusterdomain>.com
```


For the Agent Setup deployment

Agent Setup is part of the GWS deployment. It needs to be configured before the GWS deployment. You must follow instructions from the Agent Setup section.

Create or update versions.yaml

Create/update the versions.yaml file with the latest docker versions:

gws-app-provisioning: 9.0.000.94
gws-app-workspace: 9.0.000.89
gws-platform-configuration: 9.0.000.77
gws-platform-datacollector: 9.0.000.49
gws-platform-ixn: 9.0.000.42
gws-platform-ocs: 9.0.000.45
gws-platform-setting: 9.0.000.52
gws-platform-statistics: 9.0.000.60
gws-platform-voice: 9.0.000.65
gws-system-nginx: 9.0.000.16
gws-ui-crmworkspace: 9.0.000.63
gws-ui-provisioning: 9.0.000.84
gws-ui-workspace: 9.0.000.82
gws-platform-ucs: 9.0.000.46
gws-platform-chat: 9.0.000.47

GWS services installation

Login to OpenShift cluster

Use the following command to log in to OpenShift cluster from the host where you will run deployment:

```
oc login --token <token> --server <url of api server>
```

Select your GWS Project

Use the following command to select the default GWS project that was created as a prerequisite:

```
oc project gws
```

Render the templates

To ensure that resources are created correctly, you can render the templates for debugging purposes within installing them. Use the following command to render the templates:

```
helm template --debug /gws-services-1.0.18.tgz -f values-gws-new.yaml -f gws-versions_ltst.yaml
```

Kubernetes descriptors are displayed. The values are generated from Helm templates, based on settings from values-gws-new.yaml and gws-versions_ltst.yaml. Ensure that no errors are displayed. This configuration will be applied to your Kubernetes cluster.

Deploy GWS

Use the following command to deploy GWS:

```
helm install gws-services./gws-services-1.0.18.tgz -f values-gws-new.yaml -f gws-versions_ltst.yaml
```

Check the deployment

Use the following command to check the installed Helm release:

```
helm list –n gws
```

The result should show gws-service deployment details similar to the following:

```
NAME                        NAMESPACE        REVISION
UPDATED                                                                    STATUS
CHART                                        APP        VERSION
gws-services        gws                        1                          2021-05-19        11:49:49.2243107
+0530 +0530        deployed        gws-services-1.0.18               1.0
```

To check the GWS project status, use the following command:

```
helm status gws-services
```

The result should display the details with 'STATUS: deployed':

```
NAME: gws-services
LAST DEPLOYED: Wed May 19 11:49:49 2021
NAMESPACE: gws
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

To check the GWS OpenShift objects created by Helm, use the following command:

```
oc get all -n gws
```

The results should show all the created Pods, services, configmaps, and so on.

GWS Ingress

The GWS project should be created and the gws-services should be installed before proceeding with the following installation.

Installation Steps

Specify the gws project

Use the following command to specify the gws project:

```
oc project gws
```

Map credentials

Use the following command to map the credentials to the default service account:

```
oc adm policy add-scc-to-user genesys-restricted -z default -n gws
```

Download the Helm charts

Download the gws-ingress Helm charts from JFrog using your credentials.

Create the Secret for consul token

Use the following command to create the Secret for consul token:

```
oc create secret generic gws-secrets-green -n gws --from-literal='gws-consul-token=<token-from-consul>'
```

Update the Host parameters

Copy the values.yaml file from the gws-ingress folder and update the values for Host parameters:

```
REGISTRY: <docker-repo-url>
entryPoints:
        internal:
                service:
                        type: LoadBalancer
                        annotations: {}
                ingress:
                        path: /
                        annotations: {}
                        tlsEnable: false
                        secretName: gws-secret-int
                        hostName: gws01-int.<yourclusterdomain>.com ------------------------------- http
internal end point for accessing GWS APIs
        internalTest:
                service:
                        type: LoadBalancer
                        annotations: {}
                ingress:
                        path: /
                        annotations: {}
                        tlsEnable: false
                        secretName: gws-secret-int
                        hostName: gws01-test.<yourclusterdomain>.com ------------------------------- http test
end point for accessing GWS APIs

        external:
                service:
                        type: ClusterIP
                        annotations: {}
                        #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
                        #service.beta.kubernetes.io/aws-load-balancer-type: nlb
                ingress:
                        path: /
                        annotations: {}
                        tlsEnable: false
                        secretName: gws-secret-ext
                        hostName: gws01.<yourclusterdomain>.com ------------------------------- http end point
for accessing GWS APIs
                        hostNameTemp: gws-temp.<yourclusterdomain>.com ------------------------------- http
test end point for accessing GWS

        externalTest:
                service:
                        type: ClusterIP
                        annotations: {}
                        #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
                        #service.beta.kubernetes.io/aws-load-balancer-type: nlb
                ingress:
                        path: /
                        annotations: {}
                        #appgw.ingress.kubernetes.io/connection-draining: "true"
                        #appgw.ingress.kubernetes.io/connection-draining-timeout: "30"
                        #appgw.ingress.kubernetes.io/cookie-based-affinity: "true"
                        #appgw.ingress.kubernetes.io/ssl-redirect: "false"
                        #cert-manager.io/cluster-issuer: letsencrypt-prod
                        #ingress.kubernetes.io/ssl-redirect: "false"
                        #kubernetes.io/ingress.class: azure/application-gateway
                tlsEnable: false
                secretName: gws-secret-ext
                hostName: gws.apps.<yourclusterdomain>.com ------------------------------- http end point for
accessing GWS

Version Details:
gws-system-nginx: 9.0.000.14
```

Copy the edited file and the gws-ingress helm package to the installation location.

Install gws-ingress

Use the following command to install the gws-ingress:

```
helm install gws-ingress ./gws-ingress-0.2.7.tgz -f values.yaml
```

Test the installation

Check the installation

To check the ingress installation, run the following commands and test:

```
oc get pod
```

This command should return gws-service-proxy and the status should running. For example:

```
gws-service-proxy-d5997957f-m4kcg 1/1 Running 0 4d13h
```

```
oc get svc
```

This command should display the service name 'gws-service-proxy. For example:

```
y ClusterIP 10.202.55.20 <none> 80/TCP,81/TCP,85/TCP,86/TCP 4d13h
```

Create https routes

The recommended Hostname format is: **gws.<cluster-subdomain>**. For example: VCE cluster https://console-openshift-console.<yourclusterdomain>.com/ will have the **gws.<yourclusterdomain>.com** host name.

Use the following command to create https routes to for external access:

```
oc create route edge --service=gws-service-proxy --hostname=<hostname>
```

Test the routes

Access the **https://<hostname>/ui/wwe/index.html** in a browser. It should navigate to the Workspace login page.

Provision GWS and Genesys Authentication

Disclaimer

Genesys Web Services and Applications and Genesys Authentication in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

To provision GWS and Genesys Authentication, you must:

1. Meet all the prerequisites.
2. Create a new API Client.
3. Create an authentication token.
4. Add a Genesys tenant/environment.
5. Add a contact center with environment.
6. Update CORS settings (optional).

Prerequisites

- You have installed the Genesys Authentication services and the following URLs are accessible:
  - <auth-url>/auth/v3/oauth/token
  - <auth-url>/environment/v3/environments
- You have the ops credentials (admin_username and admin_password) from the **values_gauth.yaml** file.
- GWS services are accessible.
- You have Configuration Server details such as hostname or IP, port, username, password, and cloud application name.

Create a new API Client

```
curl --location --request POST '<gauth-url>/auth/v3/ops/clients' \
--header 'Content-Type: application/json' \
--user ops:ops \ ---------------------------- Cloud ops credentials (<username:password>) from values_gauth.
yaml. The default value is ops:ops
--data-raw '{"data": {
        "name": "external_api_client", ----------------- <Client Name>
        "clientType": "CONFIDENTIAL",
        "refreshTokenExpirationTimeout": 43200,
        "client_id": "external_api_client", ----------------- <Client ID>
        "client_secret": "", --------------------------<Client Password>
        "authorities": ["ROLE_INTERNAL_CLIENT"],
        "scope": ["*"],
        "authorizedGrantTypes": ["client_credentials", "authorization_code", "refresh_token", "password"],
        "redirectURIs": ["https://gauth.<yourcluster.com>","https://wwe.<yourcluster.com>","https://gws.
<yourcluster.com>","https://prov.<yourcluster.com>"], -----> should add gws/prov external URLS here
        "accessTokenExpirationTimeout": 43200,
        "contactCenterIds": [
        "*" ------------------ <CCID or *>
        ]

        }
}'

Result:
"status": {
                "code": 0
        },
        "data": {
                "clientType": "CONFIDENTIAL",
                "scope": [
                        "*"
                ],
                "internalClient": false,
                "authorizedGrantTypes": [
                        "refresh_token",
                        "client_credentials",
                        "password",
                        "authorization_code",
                        "urn:ietf:params:oauth:grant-type:token-exchange",
                        "urn:ietf:params:oauth:grant-type:jwt-bearer"
                ],
                "authorities": [
                        "ROLE_INTERNAL_CLIENT"
                ],

                "redirectURIs": [
                        "https://gauth.<yourcluster.com>",
            "https://gws.<yourcluster.com>",
            "https://prov.<yourcluster.com>",

                ],
                "contactCenterIds": [
                        "9350e2fc-a1dd-4c65-8d40-1f75a2e080dd"
                ],
                "accessTokenExpirationTimeout": 43200,
                "refreshTokenExpirationTimeout": 43200,
                "createdAt": 1619796576236,
                "name": "external_api_client",
                "client_id": "external_api_client",
                "client_secret": "secret",
                "encrypted_client_secret": "A34BOmXDedZwbTKrwmd4eA=="
        }
}
```

Create an authentication token

```
curl --location --user external_api_client:secret --request POST '<gauth-url>/auth/v3/oauth/token' \ ----- user
is the API client created in the previous step
--data-urlencode 'username=ops' \
--data-urlencode 'client_id=external_api_client' \ ------------------ client ID created in the previous step
--data-urlencode 'grant_type=password' \
--data-urlencode 'password=ops'


Result

{

        "access_token": "5f1ecb33-5c63-4606-8e30-824e494194c6",
        "token_type": "bearer",
        "refresh_token": "f0c7eed6-cc55-426f-9594-7ae14903e749",
        "expires_in": 43199,
        "scope": "*"
}
```

Add a Genesys tenant/environment

> Complete this step after installing the Tenant service.

```
curl --location --request POST '<gauth-url>/environment/v3/environments' \

--header 'Content-Type: application/json' \
--header 'Authorization: Bearer f3aa2109-8889-4182-b2b7-d86917c53e4e' \ ----- access token generated in
previous step
--data-raw '{
        "data": {
                "username": "default", -------------------------- Configuration Server username
                "password": "password", ------------------------- Configuration Server password
                "connectionProtocol": "addp",
                "remoteTimeout": 7,
                "appName": "Cloud", ------------------------- Cloud app
                "traceMode": "CFGTMBoth",
                "tlsEnabled": false,
                "configServers": [{
                        "primaryPort": 2020, ---------------------- Configuration Server port
                        "readOnly": false,
                        "primaryAddress": "172.24.132.84", ------ Configuration Server IP
                        "locations": "/USW1"

                }],
                "localTimeout": 5,
                "tenant": "Environment"
        }
}'



Result

{
        "status": {
                "code": 0
        },
        "path": "/environments/d0fb6386-236c-4739-aec0-b9c1bd6173df" - Environment ID
}
```

Add a contact center with environment

Complete this step after installing the Tenant service.

```
curl --location --request POST '<gauth-url>/environment/v3/contact-centers' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer 9901f8d6-0351-47f8-b718-7db992f53a02' \
--data-raw '{
        "data": {

                "domains": <customer-domain>,
                "environmentId": "343dd264-7c26-4f9e-82c5-26baedbcb797", ------- > Environment ID created in
the previous step
                "auth": "configServer",
                "id" : <CC-id> which is used while deploying tenant service
        }
}'



Result



{
        "status": {
                "code": 0
        },
        "path": "/contact-centers/ed4c03f3-6275-4419-8b2b-11d14af10655" - Contact center ID
```

Make note of the contact center ID (also known as CCID) from the POST request above – you require it to provision other Genesys services. Now, open a web browser, navigate to the GWS URL and try to log in using any agent available in Configuration Server.

Update CORS settings (optional)

This step is only necessary when services need CORS clearance from GWS.

```
curl --location --request POST '<gauth-url>/environment/v3/cors' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer 201ad145-3b79-4d25-b88e-6c3279e00c63' \ --- Bearer Token
--data-raw '{
        "data": {
                "origin": "", ------------ URL to add Origin
                "contactCenterId": "" -------------- CCID
        }
}'
```

Provision Agent Setup

Disclaimer

Agent Setup in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Context

Agent Setup is part of Genesys Web Services and Applications (GWS) and must be enabled when deploying GWS services.

Agent Setup is used to manage the controls and settings that run the contact center and enable the users within it to handle and manage interactions.

This topic describes how to **provision** Agent Setup in your environment.

To provision Agent Setup you must run the following commands in order to create https access routes for both `gws-app-provisioning` and `gws-ui-provisioning`.

```
oc create route edge --service=gws-ui-provisioning-blue-srv --hostname=<hostname> --path /ui/provisioning

oc create route edge --service=gws-app-provisioning-blue-srv --hostname=<hostname> --path /provisioning
```

Tip

Genesys recommends the hostname format `prov.<cluster-subdomain>`

Example:

For VCE cluster (https://console-openshift-console.apps.<yourclusterdomain>.com/), the host name should be `prov.apps.<yourclusterdomain>.com`

Next steps

- Launch Agent Setup using the URL **https://<hostname>/ui/provisioning**

Deploy Genesys Voice Platform

> **Disclaimer**
>
> Genesys Voice Platform (GVP) in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

Genesys Voice Platform (GVP) is a software-only, standards-based voice portal that provides cost-effective customer interactions, 24x7, for businesses using voice, video, the web, and the cloud. Functioning beyond traditional IVR systems, GVP provides touch-tone access to applications and incorporates speech recognition technology and video for conversational exchanges, better to identify and resolve customer requests.

GVP employs a VoiceXML-based media server for network service providers and enterprise customers. GVP is a self-service system that comes with Genesys Media Server and can provide media services simultaneously with VoiceXML self-service applications.

Media services available with GVP are:

- Call parking
- Call qualification
- Call progress detection
- Third-party call recording support
- Call conferencing
- Audio/video streaming

So GVP can be used to provide augmented routing and agent services in addition to self-service applications, proactive contact solutions and outbound calling media.

Prerequisites

- Consul with Service Mesh and DNS
- Availability of shared Postgres for GVP Configuration Server
- Availability AzureSQL Database for Reporting Server
  - DB should be created in advance (Example DB Name: **gvp_rs**)
  - There is a requirement for one user to have admin (dbo) access and a second with read only (ro) access
  - These credentials will be used for creation of Reporting server secrets

Environment setup

- Log in to the OpenShift cluster from the remote host via CLI

```
oc login --token <token> --server <URL of the API server>
```

- Check the cluster version

```
oc get clusterversion
```

- Create gvp project in OpenShift cluster

```
oc new-project gvp
```

- Set default project to GVP

```
oc project gvp
```

- Bind SCC to genesys user using default service account

```
oc adm policy add-scc-to-user genesys-restricted -z default -n gvp
```

- Create secret for docker-registry in order to pull image from JFrog

```
oc create secret docker-registry <credential-name> --docker-server=<docker repo> --docker-username=<username> --docker-password=<API key from jfrog> --docker-email=<emailid>
```

- Link the secret to default service account with pull role

```
oc secrets link default <credential-name> --for=pull
```

Installation Order

Installation order matters with GVP. To deploy without errors, the install order should be:

1. Configuration Server
2. Service Discovery
3. Reporting Server
4. Resource Manager
5. MCP

Helm chart release URLs

Download the GVP Helm charts from JFrog using your credentials.

gvp-configserver : https://<jfrog artifactory/helm location>/gvp-configserver-9.000.16.tgz
gvp-sd : https://<jfrog artifactory/helm location>/gvp-sd-9.000.52.tgz
gvp-rs : https://<jfrog artifactory/helm location>/gvp-rs-9.048.18.tgz
gvp-rm : https://<jfrog artifactory/helm location>/gvp/gvp-rm-9.048.31.tgz
gvp-mcp : https://<jfrog artifactory/helm location>/gvp-mcp-9.048.65.tgz

1. GVP Configuration Server

Provision GVP Configuration Database

You will need a Postgres Database to host the configuration of GVP:

- Database Name must be 'gvp' (limitation to be addressed)
- User must be 'gvp' and have full access to 'gvp' database (limitation to be addressed)

Secrets creation

Create the following secrets which are required for the service deployment.

postgres-secret

db-hostname: Hostname of DB Server

db-name: Database Name

db-password: password for db user

db-username: username for db

server-name: Hostname of DB Server

postgres-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
  namespace: gvp
type: Opaque
data:
  db-username: Z3Zw
  db-password: Z3Zw
  db-hostname: cG9zdGdyZXMtcncuaW5mcmEuc3ZjLmNsdXN0ZXIubG9jYWw=
  db-name: Z3Zw
  server-name: cG9zdGdyZXMtcncuaW5mcmEuc3ZjLmNsdXN0ZXIubG9jYWw=
```

Execute the following command:

```
oc create -f postgres-secret.yaml
```

configserver-secret

password: Password to set for Config DB

username: Username to set for Config DB

configserver-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: configserver-secret
  namespace: gvp
type: Opaque
data:
  username: ZGVmYXVsdA==
  password: cGFzc3dvcmQ=
```

Execute the following command:

```
oc create -f configserver-secret.yaml
```


Install Helm chart

Download the required Helm chart release from the JFrog repository and install. Refer to Helm chart release URLs.

Install Helm Chart

```
helm install gvp-configserver ./<gvp-configserver-helm-artifact> -f gvp-configserver-values.yaml
```

At minimum following values will need to be updated in your values.yaml:

- <critical-priority-class> - Set to a priority class that exists on cluster (or create it instead)
- <docker-repo> - Set to your Docker Repo with Private Edition Artifacts
- <credential-name> - Set to your pull secret name

gvp-configserver-values.yaml

```
# Default values for gvp-configserver.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
```

```
## Global Parameters
## Add labels to all the deployed resources
##
podLabels: {}

## Add annotations to all the deployed resources
##
podAnnotations: {}

serviceAccount:
  # Specifies whether a service account should be created
  create: false
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name:

## Deployment Configuration
## replicaCount should be 1 for Config Server
replicaCount: 1

## Base Labels. Please do not change these.
serviceName: gvp-configserver
component: shared
# Namespace
partOf: gvp

## Container image repo settings.
image:
  confserv:
    registry: <docker-repo>
    repository: gvp/gvp_confserv
    pullPolicy: IfNotPresent
    tag: "{{ .Chart.AppVersion }}"
  serviceHandler:
    registry: <docker-repo>
    repository: gvp/gvp_configserver_servicehandler
    pullPolicy: IfNotPresent
    tag: "{{ .Chart.AppVersion }}"
  dbInit:
    registry: <docker-repo>
    repository: gvp/gvp_configserver_configserverinit
    pullPolicy: IfNotPresent
    tag: "{{ .Chart.AppVersion }}"

## Config Server App Configuration
configserver:
  ## Settings for liveness and readiness probes
  ## !!! THESE VALUES SHOULD NOT BE CHANGED UNLESS INSTRUCTED BY GENESYS !!!
  livenessValues:
    path: /cs/liveness
    initialDelaySeconds: 30
    periodSeconds: 60
    timeoutSeconds: 20
    failureThreshold: 3
    healthCheckAPIPort: 8300

  readinessValues:
    path: /cs/readiness
    initialDelaySeconds: 30
    periodSeconds: 30
    timeoutSeconds: 20
    failureThreshold: 3
    healthCheckAPIPort: 8300

  alerts:
    cpuUtilizationAlertLimit: 70
    memUtilizationAlertLimit: 90
    workingMemAlertLimit: 7
    maxRestarts: 2
```

```
## PVCs defined
#  none

## Define service(s) for application
service:
  type: ClusterIP
  host: gvp-configserver-0
  port: 8888
  targetPort: 8888

## Service Handler configuration.
serviceHandler:
  port: 8300

## Secrets storage related settings - k8s secrets only
secrets:
  # Used for pulling images/containers from the repositories.
  imagePull:
    - name: <credential-name>

  # Config Server secrets. If k8s is false, csi will be used, else k8s will be used.
  # Currently, only k8s is supported!
  configServer:
    secretName: configserver-secret
    secretUserKey: username
    secretPwdKey: password
    #csiSecretProviderClass: keyvault-gvp-gvp-configserver-secret

  # Config Server Postgres DB secrets and settings.
  postgres:
    dbName: gvp
    dbPort: 5432
    secretName: postgres-secret
    secretAdminUserKey: db-username
    secretAdminPwdKey: db-password
    secretHostnameKey: db-hostname
    secretDbNameKey: db-name
    #secretServerNameKey: server-name

## Ingress configuration
ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

## App resource requests and limits
## ref: http://kubernetes.io/docs/user-guide/compute-resources/
##
resources:
  requests:
    memory: "512Mi"
    cpu: "500m"
  limits:
    memory: "1Gi"
    cpu: "1"

## App containers' Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-
a-container
##
## Containers should run as genesys user and cannot use elevated permissions
##
```

```
securityContext:
  runAsUser: 500
  runAsGroup: 500
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

podSecurityContext: {}
  # fsGroup: 2000

## Priority Class
## ref: https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
## NOTE: this is an optional parameter
##
priorityClassName: <critical-priority-class>

## Affinity for assignment.
## Ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
##
affinity: {}

## Node labels for assignment.
## ref: https://kubernetes.io/docs/user-guide/node-selection/
##
nodeSelector: {}

## Tolerations for assignment.
## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
##
tolerations: []

## Service/Pod Monitoring Settings
## Whether to create Prometheus alert rules or not.
prometheusRule:
  create: true

## Grafana dashboard Settings
## Whether to create Grafana dashboard or not.
grafana:
  enabled: true

## Enable network policies or not
networkPolicies:
  enabled: false

## DNS configuration options
dnsConfig:
  options:
    - name: ndots
      value: "3"
```


Verify the deployed resources

Verify the deployed resources from OpenShift console/CLI.

2. GVP Service Discovery

NOTE: After GVP-SD pod gets deployed, you will notice a few errors. Please ignore them and move on to next deployment. This will start working once RM & MCP are deployed.

Secrets creation

Create the following secrets which are required for the service deployment.

shared-consul-consul-gvp-token

shared-consul-consul-gvp-token-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: shared-consul-consul-gvp-token
  namespace: gvp
type: Opaque
data:
  consul-consul-gvp-token: ZmU2NjFkNWYtYzVmNi1mZTJlLTgyM2MtYTAyZGQwN2JlMzll
```

Execute the following command:

```
oc create -f shared-consul-consul-gvp-token-secret.yaml
```

ConfigMap creation

Creation of a **tenant-inventory** ConfigMap is required for service discovery deployment.

Caveat

If the tenant has not been deployed yet then you will not have the information needed to populate the config map. An empty config-map can be created using:

```
oc create configmap tenant-inventory -n gvp
```

Provisioning a new tenant

Create a file (t100.json in the example) containing at minimum: **name, id, gws-ccid,** and **default-application** (should be set to **IVRAppDefault**) from your tenant deployment.

t100.json

```
{
    "name": "t100",
    "id": "80dd",
    "gws-ccid": "9350e2fc-a1dd-4c65-8d40-1f75a2e080dd",
    "default-application": "IVRAppDefault"
}
```

Execute the following command to create ConfigMap on cluster:

Add Config Map

```
oc create configmap tenant-inventory --from-file t100.json -n gvp
```

Updating a tenant

Delete the **tenant-inventory** ConfigMap using:

Delete Config Map

```
oc delete configmap tenant-inventory -n gvp --ignore-not-found
```

Update the **t100.json** file and execute the following to create the new ConfigMap:

Add Config Map

```
oc create configmap tenant-inventory --from-file t100.json -n gvp
```

GVP Service Discovery looks for changes to the config map every 60 seconds.

Provisioning process details

For additional details on the provisioning process, please see Provision GVP.

Install Helm chart

Download the required Helm chart release from the JFrog repository and install. Refer to Helm chart release URLs.

Install Helm Chart

```
helm install gvp-sd ./<gvp-sd-helm-artifact> -f gvp-sd-values.yaml
```

At minimum following values will need to be updated in your values.yaml:

- <critical-priority-class> - Set to a priority class that exists on cluster (or create it instead)
- <docker-repo> - Set to your Docker Repo with Private Edition Artifacts
- <credential-name> - Set to your pull secret name

gvp-sd-values.yaml

```
# Default values for gvp-sd.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

## Global Parameters
## Add labels to all the deployed resources
##
podLabels: {}

## Add annotations to all the deployed resources
##
podAnnotations: {}

serviceAccount:
  # Specifies whether a service account should be created
  create: false
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name:

## Deployment Configuration
replicaCount: 1
smtp: allowed

## Name overrides
nameOverride: ""
fullnameOverride: ""

## Base Labels. Please do not change these.
component: shared
partOf: gvp

image:
  registry: <docker-repo>
  repository: gvp/gvp_sd
  tag: "{{ .Chart.AppVersion }}"
  pullPolicy: IfNotPresent
```

```
## PVCs defined
# none

## Define service for application.
service:
  name: gvp-sd
  type: ClusterIP
  port: 8080

## Application configuration parameters.
env:
  MCP_SVC_NAME: "gvp-mcp"
  EXTERNAL_CONSUL_SERVER: ""
  CONSUL_PORT: "8501"
  CONFIG_SERVER_HOST: "gvp-configserver"
  CONFIG_SERVER_PORT: "8888"
  CONFIG_SERVER_APP: "default"
  HTTP_SERVER_PORT: "8080"
  METRICS_EXPORTER_PORT: "9090"
  DEF_MCP_FOLDER: "MCP_Configuration_Unit\\MCP_LRG"
  TEST_MCP_FOLDER: "MCP_Configuration_Unit_Test\\MCP_LRG"
  SYNC_INIT_DELAY: "10000"
  SYNC_PERIOD: "60000"
  MCP_PURGE_PERIOD_MINS: "0"
  EMAIL_METERING_FACTOR: "10"
  RECORDINGS_CONTAINER: "ccerp-recordings"
  TENANT_KV_FOLDER: "tenants"
  TENANT_CONFIGMAP_FOLDER: "/etc/config"
  SMTP_SERVER: "smtp-relay.smtp.svc.cluster.local"

## Secrets storage related settings
secrets:
  # Used for pulling images/containers from the repositories.
  imagePull:
    - name: <credential-name>

  # If k8s is true, k8s will be used, else vault secret will be used.
  configServer:
    k8s: true
    k8sSecretName: configserver-secret
    k8sUserKey: username
    k8sPasswordKey: password
    vaultSecretName: "/configserver-secret"
    vaultUserKey: "configserver-username"
    vaultPasswordKey: "configserver-password"

  # If k8s is true, k8s will be used, else vault secret will be used.
  consul:
    k8s: true
    k8sTokenName: "shared-consul-consul-gvp-token"
    k8sTokenKey: "consul-consul-gvp-token"
    vaultSecretName: "/consul-secret"
    vaultSecretKey: "consul-consul-gvp-token"

  # GTTS key, password via k8s secret, if k8s is true.  If false, this data comes from tenant profile.
  gtts:
    k8s: false
    k8sSecretName: gtts-secret
    EncryptedKey: encrypted-key
    PasswordKey: password

ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
```

```
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

resources:
  requests:
    memory: "2Gi"
    cpu: "1000m"
  limits:
    memory: "2Gi"
    cpu: "1000m"

## App containers' Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-
a-container
##
## Containers should run as genesys user and cannot use elevated permissions
## Pod level security context
podSecurityContext:
  fsGroup: 500
  runAsUser: 500
  runAsGroup: 500
  runAsNonRoot: true

## Container security context
securityContext:
  runAsUser: 500
  runAsGroup: 500
  runAsNonRoot: true

## Priority Class
## ref: https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
## NOTE: this is an optional parameter
##
priorityClassName: <critical-priority-class>

## Affinity for assignment.
## Ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
##
affinity: {}

## Node labels for assignment.
## ref: https://kubernetes.io/docs/user-guide/node-selection/
##
nodeSelector: {}

## Tolerations for assignment.
## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
##
tolerations: []

## Service/Pod Monitoring Settings
prometheus:
  # Enable for Prometheus operator
  podMonitor:
    enabled: true

## Enable network policies or not
networkPolicies:
  enabled: false

## DNS configuration options
dnsConfig:
  options:
    - name: ndots
      value: "3"
```


Verify the deployed resources

Verify the deployed resources from OpenShift console/CLI.

3. GVP Reporting Server

Secrets creation

Create the following secrets which are required for the service deployment.

rs-dbreader-password

db_hostname:
db_name:
db_password:
db_username:

rs-dbreader-password-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: rs-dbreader-password
  namespace: gvp
type: Opaque
data:
  db_username: b3BlbnNoaWZ0YWRtaW4=
  db_password: R2VuZXN5c0AxMjM=
  db_hostname: bXNzcWxzZXJ2ZXJvcGVuc2hpZnQuZGF0YWJhc2Uud2luZG93cy5uZXQ=
  db_name: cnNfZ3Zw
```

Execute the following command:

```
oc create -f rs-dbreader-password-secret.yaml
```

shared-gvp-rs-sqlserer-secret

db-admin-password:
db-reader-password:

shared-gvp-rs-sqlserer-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: shared-gvp-rs-sqlserver-secret
  namespace: gvp
type: Opaque
data:
  db-admin-password: R2VuZXN5c0AxMjM=
  db-reader-password: R2VuZXN5c0AxMjM=
```

Execute the following command:

```
oc create -f shared-gvp-rs-sqlserer-secret.yaml
```

Persistent Volumes creation

Create the following PVs which are required for the service deployment.

Note Regarding Persistent Volumes

If your OpenShift deployment is capable of self-provisioning of Persistent Volumes, then this step can be skipped. Volumes will be created by provisioner.

gvp-rs-0

gvp-rs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-rs-0
  namespace: gvp
spec:
  capacity:
    storage: 30Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/rs-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-rs-pv.yaml
```

Install Helm chart

Download the required Helm chart release from the JFrog repository and install. Refer to Helm chart release URLs.

Install Helm Chart

```
helm install gvp-rs ./<gvp-rs-helm-artifact> -f gvp-rs-values.yaml
```

At minimum following values will need to be updated in your values.yaml:

- <docker-repo> - Set to your Docker Repo with Private Edition Artifacts
- <credential-name> - Set to your pull secret name

gvp-rs-values.yaml

```
## Global Parameters
## Add labels to all the deployed resources
##
labels:
  enabled: true
  serviceGroup: "gvp"
  componentType: "shared"

serviceAccount:
  # Specifies whether a service account should be created
  create: false
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name:
```

```
## Primary App Configuration
##
# primaryApp:
# type: ReplicaSet
# Should include the defaults for replicas
deployment:
  replicaCount: 1
  strategy: Recreate
  namespace: gvp
nameOverride: ""
fullnameOverride: ""

image:
  registry: <docker-repo>
  gvprsrepository: gvp/gvp_rs
  snmprepository: gvp/gvp_snmp
  rsinitrepository: gvp/gvp_rs_init
  rstag:
  rsinittag:
  snmptag: v9.0.040.07
  pullPolicy: Always
  imagePullSecrets:
    - name: "<credential-name>"

## liveness and readiness probes
## !!! THESE OPTIONS SHOULD NOT BE CHANGED UNLESS INSTRUCTED BY GENESYS !!!
livenessValues:
  path: /ems-rs/components
  initialDelaySeconds: 30
  periodSeconds: 120
  timeoutSeconds: 3
  failureThreshold: 3

readinessValues:
  path: /ems-rs/components
  initialDelaySeconds: 10
  periodSeconds: 60
  timeoutSeconds: 3
  failureThreshold: 3

## PVCs defined
volumes:
  pvc:
    storageClass: managed-premium
    claimSize: 20Gi
    activemqAndLocalConfigPath: "/billing/gvp-rs"

## Define service(s) for application.  Fields many need to be modified based on `type`
service:
  type: ClusterIP
  restapiport: 8080
  activemqport: 61616
  envinjectport: 443
  dnsport: 53
  configserverport: 8888
  snmpport: 1705

## ConfigMaps with Configuration
## Use Config Map for creating environment variables
context:
  env:
    CFGAPP: default
    GVP_RS_SERVICE_HOSTNAME: gvp-rs.gvp.svc.cluster.local
    #CFGPASSWORD: password
    #CFGUSER: default
    CFG_HOST: gvp-configserver.gvp.svc.cluster.local
    CFG_PORT: '8888'
    CMDLINE: ./rs_startup.sh
    DBNAME: gvp_rs
    #DBPASS: 'jbIKfoS6LpfgaU$E'
    DBUSER: openshiftadmin
```

```
    rsDbSharedUsername: openshiftadmin
    DBPORT: 1433
    ENVTYPE: ""
    GenesysIURegion: ""
    localconfigcachepath: /billing/gvp-rs/data/cache
    HOSTFOLDER: Hosts
    HOSTOS: CFGRedHatLinux
    LCAPORT: '4999'
    MSSQLHOST: mssqlserveropenshift.database.windows.net
    RSAPP: azure_rs
    RSJVM_INITIALHEAPSIZE: 500m
    RSJVM_MAXHEAPSIZE: 1536m
    RSFOLDER: Applications
    RS_VERSION: 9.0.032.22
    STDOUT: 'true'
    WRKDIR: /usr/local/genesys/rs/
    SNMPAPP: azure_rs_snmp
    SNMP_WORKDIR: /usr/sbin
    SNMP_CMDLINE: snmpd
    SNMPFOLDER: Applications

  RSCONFIG:
    messaging:
      activemq.memoryUsageLimit: "256 mb"
      activemq.dataDirectory: "/billing/gvp-rs/data/activemq"
    log:
      verbose: "trace"
      trace: "stdout"
    dbmp:
      rs.db.retention.operations.daily.default: "40"
      rs.db.retention.operations.monthly.default: "40"
      rs.db.retention.operations.weekly.default: "40"
      rs.db.retention.var.daily.default: "40"
      rs.db.retention.var.monthly.default: "40"
      rs.db.retention.var.weekly.default: "40"
      rs.db.retention.cdr.default: "40"

# Default secrets storage to k8s secrets with csi able to be optional
secret:
  # keyVaultSecret will be a flag to between secret types(k8's or CSI). If keyVaultSecret was set to false k8's
secret will be used
  keyVaultSecret: false
  #RS SQL server secret
  rsSecretName: shared-gvp-rs-sqlserver-secret
  # secretProviderClassName will not be used used when keyVaultSecret set to false
  secretProviderClassName: keyvault-gvp-rs-sqlserver-secret-00
  dbreadersecretFileName: db-reader-password
  dbadminsecretFileName: db-admin-password
  #Configserver secret
  #If keyVaultSecret set to false the below parameters will not be used.
  configserverProviderClassName: gvp-configserver-secret
  cfgSecretFileNameForCfgUsername: configserver-username
  cfgSecretFileNameForCfgPassword: configserver-password
  #If keyVaultSecret set to true the below parameters will not be used.
  cfgServerSecretName: configserver-secret
  cfgSecretKeyNameForCfgUsername: username
  cfgSecretKeyNameForCfgPassword: password

## Ingress configuration
ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
  # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
```

```
networkPolicies:
  enabled: false

## primaryAppresource requests and limits
## ref: http://kubernetes.io/docs/user-guide/compute-resources/
##
resourceForRS:
  # We usually recommend not to specify default resources and to leave this as a conscious
  # choice for the user. This also increases chances charts run on environments with little
  # resources, such as Minikube. If you do want to specify resources, uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  requests:
    memory: "500Mi"
    cpu: "200m"
  limits:
    memory: "1Gi"
    cpu: "300m"

resoueceForSnmp:
  requests:
    memory: "500Mi"
    cpu: "100m"
  limits:
    memory: "1Gi"
    cpu: "150m"

## primaryApp containers' Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-
a-container
##
## Containers should run as genesys user and cannot use elevated permissions
securityContext:
  runAsNonRoot: true
  runAsUser: 500
  runAsGroup: 500

podSecurityContext:
  fsGroup: 500

## Priority Class
## ref: https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
##
priorityClassName: ""

## Affinity for assignment.
## Ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
##
affinity: {}

## Node labels for assignment.
## ref: https://kubernetes.io/docs/user-guide/node-selection/
##
nodeSelector: {}

## Tolerations for assignment.
## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
##
tolerations: []

## Extra labels
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
##
#  labels: {}

## Extra Annotations
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
##
#  annotations: {}

## Service/Pod Monitoring Settings
```

```
monitoring:
  podMonitorEnabled: true
  prometheusRulesEnabled: true
  grafanaEnabled: true

monitor:
  prometheusPort: 9116
  monitorName: gvp-monitoring
  module: [if_mib]
  target: [127.0.0.1:1161]

##DNS Settings
dnsConfig:
  options:
    - name: ndots
      value: "3"
```

Verify the deployed resources

Verify the deployed resources from OpenShift console/CLI.

4. GVP Resource Manager

> **Note**
>
> RM and forward will not pass readiness checks until an MCP has registered properly. This is because service is not available without MCPs.

Persistent Volumes creation

Create the following PVs which are required for the service deployment.

> **Note Regarding Persistent Volumes**
>
> If your OpenShift deployment is capable of self-provisioning of Persistent Volumes, then this step can be skipped. Volumes will be created by provisioner.

gvp-rm-01

gvp-rm-01-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-rm-01
spec:
  capacity:
    storage: 30Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/rm-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-rm-01-pv.yaml
```

gvp-rm-02

gvp-rm-02-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-rm-02
spec:
  capacity:
    storage: 30Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/rm-02
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-rm-02-pv.yaml
```

gvp-rm-logs-01

gvp-rm-logs-01-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-rm-logs-01
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/rm-logs-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-rm-logs-01-pv.yaml
```

gvp-rm-logs-02

gvp-rm-logs-02-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-rm-logs-02
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/rm-logs-02
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-rm-logs-02-pv.yaml
```

Install Helm chart

Download the required Helm chart release from the JFrog repository and install. Refer to Helm chart release URLs.

Install Helm Chart

```
helm install gvp-rm ./<gvp-rm-helm-artifact> -f gvp-rm-values.yaml
```

At minimum following values will need to be updated in your values.yaml:

- <docker-repo> - Set to your Docker Repo with Private Edition Artifacts
- <credential-name> - Set to your pull secret name

gvp-rm-values.yaml

```
## Global Parameters
## Add labels to all the deployed resources
##
labels:
  enabled: true
  serviceGroup: "gvp"
  componentType: "shared"

## Primary App Configuration
##
# primaryApp:
# type: ReplicaSet
# Should include the defaults for replicas
deployment:
  replicaCount: 2
  deploymentEnv: "UPDATE_ENV"
  namespace: gvp
  clusterDomain: "svc.cluster.local"
nameOverride: ""
fullnameOverride: ""

image:
  registry: <docker-repo>
  gvprmrepository: gvp/gvp_rm
  cfghandlerrepository: gvp/gvp_rm_cfghandler
  snmprepository: gvp/gvp_snmp
  gvprmtestrepository: gvp/gvp_rm_test
  cfghandlertag:
```

```
  rmtesttag:
  rmtag:
  snmptag: v9.0.040.07
  pullPolicy: Always
  imagePullSecrets:
    - name: "<credential-name>"

dnsConfig:
  options:
    - name: ndots
      value: "3"

# Pod termination grace period 15 mins.
gracePeriodSeconds: 900

## liveness and readiness probes
## !!! THESE OPTIONS SHOULD NOT BE CHANGED UNLESS INSTRUCTED BY GENESYS !!!
livenessValues:
  path: /rm/liveness
  initialDelaySeconds: 60
  periodSeconds: 90
  timeoutSeconds: 20
  failureThreshold: 3

readinessValues:
  path: /rm/readiness
  initialDelaySeconds: 10
  periodSeconds: 60
  timeoutSeconds: 20
  failureThreshold: 3

## PVCs defined
volumes:
  billingpvc:
    storageClass: managed-premium
    claimSize: 20Gi
    mountPath: "/rm"
  logpvc:
    EnablePVForLogStorage: true
    storageClass: managed-premium
    claimSize: 5Gi
    accessMode: ReadWriteOnce
    mountPath: "/mnt/log"
    # If PV is not used for log storage by disabling the flag EnablePVForLogStorage: false, the given host path
will be used for log storage.
    LogStorageHostPath: /mnt/log

## Define service(s) for application.  Fields many need to be modified based on `type`
service:
  type: ClusterIP
  port: 5060
  rmHealthCheckAPIPort: 8300

## ConfigMaps with Configuration
## Use Config Map for creating environment variables
context:
  env:
    cfghandler:
      CFGSERVER: gvp-configserver.gvp.svc.cluster.local
      CFGSERVERBACKUP: gvp-configserver.gvp.svc.cluster.local
      CFGPORT: "8888"
      CFGAPP: "default"
      RMAPP: "azure_rm"
      RMFOLDER: "Applications\\RM_MicroService\\RM_Apps"
      HOSTFOLDER: "Hosts\\RM_MicroService"
      MCPFOLDER: "MCP_Configuration_Unit\\MCP_LRG"
      SNMPFOLDER: "Applications\\RM_MicroService\\SNMP_Apps"
      EnvironmentType: "prod"
      CONFSERVERAPP: "confserv"
      RSAPP: "azure_rs"
      SNMPAPP: "azure_rm_snmp"
```

```
      STDOUT: "true"
      VOICEMAILSERVICEDIDNUMBER: "55551111"

  RMCONFIG:
    rm:
      sip-header-for-dnis: "Request-Uri"
      ignore-gw-lrg-configuration: "true"
      ignore-ruri-tenant-dbid: "true"
    log:
      verbose: "trace"
    subscription:
      sip.transport.dnsharouting: "true"
      sip.headerutf8verification: "false"
      sip.transport.setuptimer.tcp: "5000"
      sip.threadpoolsize: "1"
    registrar:
      sip.transport.dnsharouting: "true"
      sip.headerutf8verification: "false"
      sip.transport.setuptimer.tcp: "5000"
      sip.threadpoolsize: "1"
    proxy:
      sip.transport.dnsharouting: "true"
      sip.headerutf8verification: "false"
      sip.transport.setuptimer.tcp: "5000"
      sip.threadpoolsize: "16"
      sip.maxtcpconnections: "1000"
    monitor:
      sip.transport.dnsharouting: "true"
      sip.maxtcpconnections: "1000"
      sip.headerutf8verification: "false"
      sip.transport.setuptimer.tcp: "5000"
      sip.threadpoolsize: "1"
    ems:
      rc.cdr.local_queue_path: "/rm/ems/data/cdrQueue_rm.db"
      rc.ors.local_queue_path: "/rm/ems/data/orsQueue_rm.db"

# Default secrets storage to k8s secrets with csi able to be optional
secret:
  # keyVaultSecret will be a flag to between secret types(k8's or CSI). If keyVaultSecret was set to false k8's
secret will be used
  keyVaultSecret: false
  #If keyVaultSecret set to false the below parameters will not be used.
  configserverProviderClassName: gvp-configserver-secret
  cfgSecretFileNameForCfgUsername: configserver-username
  cfgSecretFileNameForCfgPassword: configserver-password
  #If keyVaultSecret set to true the below parameters will not be used.
  cfgServerSecretName: configserver-secret
  cfgSecretKeyNameForCfgUsername: username
  cfgSecretKeyNameForCfgPassword: password

## Ingress configuration
ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
  # kubernetes.io/tls-acme: "true"
  paths: []
  hosts:
    - chart-example.local
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
networkPolicies:
  enabled: false
sip:
  serviceName: sipnode

## primaryAppresource requests and limits
## ref: http://kubernetes.io/docs/user-guide/compute-resources/
##
```

```
resourceForRM:
  # We usually recommend not to specify default resources and to leave this as a conscious
  # choice for the user. This also increases chances charts run on environments with little
  # resources, such as Minikube. If you do want to specify resources, uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  requests:
    memory: "1Gi"
    cpu: "200m"
    ephemeral-storage: "10Gi"
  limits:
    memory: "2Gi"
    cpu: "250m"

resoueceForSnmp:
  requests:
    memory: "500Mi"
    cpu: "100m"
  limits:
    memory: "1Gi"
    cpu: "150m"

## primaryApp containers' Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-
a-container
##
## Containers should run as genesys user and cannot use elevated permissions
securityContext:
  fsGroup: 500
  runAsNonRoot: true
  runAsUserRM: 500
  runAsGroupRM: 500
  runAsUserCfghandler: 500
  runAsGroupCfghandler: 500

## Priority Class
## ref: https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
##
priorityClassName: ""

## Affinity for assignment.
## Ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
##
affinity: {}

## Node labels for assignment.
## ref: https://kubernetes.io/docs/user-guide/node-selection/
##
nodeSelector:


## Tolerations for assignment.
## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
##
tolerations: []

## Service/Pod Monitoring Settings
monitoring:
  podMonitorEnabled: true
  prometheusRulesEnabled: true
  grafanaEnabled: true

monitor:
  monitorName: gvp-monitoring
  prometheusPort: 9116
  prometheusPortlogs: 8200
  logFilePrefixName: RM
  module: [if_mib]
  target: [127.0.0.1:1161]
```

Verify the deployed resources

Verify the deployed resources from OpenShift console/CLI.

5. GVP MCP

Persistent Volumes creation

Create the following PVs which are required for the service deployment.

> **Note Regarding Persistent Volumes**
>
> If your OpenShift deployment is capable of self-provisioning of Persistent Volumes, then this step can be skipped. Volumes will be created by provisioner.

gvp-mcp-logs-01

gvp-mcp-logs-01-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-logs-01
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-logs-01-pv.yaml
```

gvp-mcp-logs-02

gvp-mcp-logs-02-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-logs-02
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-02
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-logs-02-pv.yaml
```

gvp-mcp-rup-volume-01

gvp-mcp-rup-volume-01-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-rup-volume-01
spec:
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: disk-premium
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-rup-volume-01-pv.yaml
```

gvp-mcp-rup-volume-02

gvp-mcp-rup-volume-02-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-rup-volume-02
spec:
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: disk-premium
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-02
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-rup-volume-02-pv.yaml
```

gvp-mcp-recording-volume-01

gvp-mcp-recordings-volume-01-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-recording-volume-01
spec:
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-01
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-recordings-volume-01-pv.yaml
```

gvp-mcp-recording-volume-02

gvp-mcp-recordings-volume-02-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gvp-mcp-recording-volume-02
spec:
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: gvp
  nfs:
    path:  /export/vol1/PAT/gvp/mcp-logs-02
    server: 192.168.30.51
```

Execute the following command:

```
oc create -f gvp-mcp-recordings-volume-02-pv.yaml
```

Install Helm chart

Download the required Helm chart release from the JFrog repository and install. Refer to Helm chart release URLs.

Install Helm Chart

```
helm install gvp-sd ./<gvp-sd-helm-artifact> -f gvp-sd-values.yaml
```

At minimum following values will need to be updated in your values.yaml:

- <critical-priority-class> - Set to a priority class that exists on cluster (or create it instead)
- <docker-repo> - Set to your Docker Repo with Private Edition Artifacts
- <credential-name> - Set to your pull secret name
- Set **logicalResourceGroup: "MCP_Configuration_Unit"** to add MCPs to the Real Configuration Unit (rather than test)

gvp-mcp-values.yaml

```
## Default values for gvp-mcp.
## This is a YAML-formatted file.
## Declare variables to be passed into your templates.

## Global Parameters
## Add labels to all the deployed resources
##
podLabels: {}

## Add annotations to all the deployed resources
##
podAnnotations: {}

serviceAccount:
  # Specifies whether a service account should be created
  create: false
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name:

## Deployment Configuration
deploymentEnv: "UPDATE_ENV"
replicaCount: 2
terminationGracePeriod: 3600

## Name and dashboard overrides
nameOverride: ""
fullnameOverride: ""
dashboardReplicaStatefulsetFilterOverride: ""

## Base Labels. Please do not change these.
serviceName: gvp-mcp
component: shared
partOf: gvp

## Command-line arguments to the MCP process
args:
  - "gvp-configserver"
  - "8888"
  - "default"
  - "/etc/mcpconfig/config.ini"

## Container image repo settings.
image:
  mcp:
    registry: <docker-repo>
    repository: gvp/multicloud/gvp_mcp
    tag: "{{ .Chart.AppVersion }}"
    pullPolicy: IfNotPresent
  serviceHandler:
    registry: <docker-repo>
    repository: gvp/multicloud/gvp_mcp_servicehandler
    tag: "{{ .Chart.AppVersion }}"
    pullPolicy: IfNotPresent
  configHandler:
    registry: <docker-repo>
    repository: gvp/multicloud/gvp_mcp_confighandler
    tag: "{{ .Chart.AppVersion }}"
    pullPolicy: IfNotPresent
  snmp:
    registry: <docker-repo>
    repository: gvp/multicloud/gvp_snmp
    tag: v9.0.040.21
    pullPolicy: IfNotPresent
  rup:
    registry: <docker-repo>
    repository: cce/recording-provider
    tag: 9.0.000.00.b.1432.r.ef30441
    pullPolicy: IfNotPresent
```

```
## MCP specific settings
mcp:
  ## Settings for liveness and readiness probes of MCP
  ## !!! THESE VALUES SHOULD NOT BE CHANGED UNLESS INSTRUCTED BY GENESYS !!!
  livenessValues:
    path: /mcp/liveness
    initialDelaySeconds: 30
    periodSeconds: 60
    timeoutSeconds: 20
    failureThreshold: 3
    healthCheckAPIPort: 8300

  # Used instead of startupProbe.  This runs all initial self-tests, and could take some time.
  # Timeout is < 1 minute (with reduced test set), and interval/period is 1 minute.
  readinessValues:
    path: /mcp/readiness
    initialDelaySeconds: 30
    periodSeconds: 60
    timeoutSeconds: 50
    failureThreshold: 3
    healthCheckAPIPort: 8300
  # Location of configuration file for MCP
  # initialConfigFile is the default template
  # finalConfigFile is the final configuration after overrides are applied (see mcpConfig section for overrides)
  initialConfigFile: "/etc/config/config.ini"
  finalConfigFile: "/etc/mcpconfig/config.ini"

  # Dev and QA deployments will use MCP_Configuration_Unit_Test LRG and shared deployments will use
MCP_Configuration_Unit LRG
  logicalResourceGroup: "MCP_Configuration_Unit"

  # Threshold values for the various alerts in podmonitor.
  alerts:
    cpuUtilizationAlertLimit: 70
    memUtilizationAlertLimit: 90
    workingMemAlertLimit: 7
    maxRestarts: 2
    persistentVolume: 20
    serviceHealth: 40
    recordingError: 7
    configServerFailure: 0
    dtmfError: 1
    dnsError: 6
    totalError: 120
    selfTestError: 25
    fetchErrorMin: 120
    fetchErrorMax: 220
    execError: 120
    sdpParseError: 1
    mediaWarning: 3
    mediaCritical: 7
    fetchTimeout: 10
    fetchError: 10
    ngiError: 12
    ngi4xx: 10
    recPostError: 7
    recOpenError: 1
    recStartError: 3
    recCertError: 7
    reportingDbInitError: 1
    reportingFlushError: 1
    grammarLoadError: 1
    grammarSynError: 1
    dtmfGrammarLoadError: 1
    dtmfGrammarError: 1
    vrmOpenSessError: 1
    wsTokenCreateError: 1
    wsTokenConfigError: 1
    wsTokenFetchError: 1
    wsOpenSessError: 1
```

```
    wsProtoError: 1
    grpcConfigError: 1
    grpcSSLRootCertError: 1
    grpcGoogleCredentialError: 1
    grpcRecognizeStartError: 7
    grpcWriteError: 7
    grpcRecognizeError: 7
    grpcTtsError: 7
    streamerOpenSessionError: 1
    streamerProtocolError: 1
    msmlReqError: 7
    dnsResError: 6
    rsConnError: 150

## RUP (Recording Uploader) Settings
rup:
  ## Settings for liveness and readiness probes of RUP
  ## !!! THESE VALUES SHOULD NOT BE CHANGED UNLESS INSTRUCTED BY GENESYS !!!
  livenessValues:
    path: /health/live
    initialDelaySeconds: 30
    periodSeconds: 60
    timeoutSeconds: 20
    failureThreshold: 3
    healthCheckAPIPort: 8080

  readinessValues:
    path: /health/ready
    initialDelaySeconds: 30
    periodSeconds: 30
    timeoutSeconds: 20
    failureThreshold: 3
    healthCheckAPIPort: 8080

  ## RUP PVC defines
  rupVolume:
    storageClass: "genesys"
    accessModes: "ReadWriteOnce"
    volumeSize: 40Gi

  ## Other settings for RUP
  recordingsFolder: "/pvolume/recordings"
  recordingsCache: "/pvolume/recording_cache"
  rupProvisionerEnabled: "false"
  decommisionDestType: "WebDAV"
  decommisionDestWebdavUrl: "http://gvp-central-rup:8180"
  decommisionDestWebdavUsername: ""
  decommisionDestWebdavPassword: ""
  diskFullDestType: "WebDAV"
  diskFullDestWebdavUrl: "http://gvp-central-rup:8180"
  diskFullDestWebdavUsername: ""
  diskFullDestWebdavPassword: ""
  cpUrl: "http://cce-conversation-provider.cce.svc.cluster.local"
  unrecoverableLostAction: "uploadtodefault"
  unrecoverableDestType: "Azure"
  unrecoverableDestAzureAccountName: "gvpwestus2dev"
  unrecoverableDestAzureContainerName: "ccerp-unrecoverable"
  logJsonEnable: true
  logLevel: INFO
  logConsoleLevel: INFO

  ## RUP resource requests and limits
  ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
  resources:
    requests:
      memory: "128Mi"
      cpu: "100m"
      ephemeral-storage: "1Gi"
    limits:
      memory: "2Gi"
      cpu: "1000m"
```

```
## PVCs defined.  RUP one is under "rup" label.
recordingStorage:
  storageClass: "genesys"
  accessModes: "ReadWriteOnce"
  volumeSize: 40Gi

# If PVC is not used by setting flag enablePV to false, the path in hostPath will be used for log storage.
logStorage:
  enablePV: true
  storageClass: genesys
  accessModes: ReadWriteOnce
  volumeSize: 5Gi
  hostPath: /mnt/log

## Service Handler configuration. Note, the port values CANNOT be changed here alone, and should not be
changed.
serviceHandler:
  serviceHandlerPort: 8300
  mcpSipPort: 5070
  consuleExternalHost: ""
  consulPort: 8501
  registrationInterval: 10000
  mcpHealthCheckInterval: 30s
  mcpHealthCheckTimeout: 10s

## Config Server values passed to RUP, etc. These should not be changed.
configServer:
  host: gvp-configserver
  port: "8888"

## Secrets storage related settings - k8s secrets or csi
secrets:
  # Used for pulling images/containers from the respositories.
  imagePull:
    - name: <credential-name>

  # Config Server secrets. If k8s is false, csi will be used, else k8s will be used.
  configServer:
    k8s: true
    secretName: configserver-secret
    dbUserKey: username
    dbPasswordKey: password
    csiSecretProviderClass: keyvault-gvp-gvp-configserver-secret

  # Consul secrets. If k8s is false, csi will be used, else k8s will be used.
  consul:
    k8s: true
    secretName: shared-consul-consul-gvp-token
    secretKey: consul-consul-gvp-token
    csiSecretProviderClass: keyvault-consul-consul-gvp-token

## Ingress configuration
ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

## App resource requests and limits
## ref: http://kubernetes.io/docs/user-guide/compute-resources/
## Values for MCP, and the default ones for other containers. RUP ones are under "rup" label.
##
resourcesMcp:
```

```
  requests:
    memory: "200Mi"
    cpu: "250m"
    ephemeral-storage: "1Gi"
  limits:
    memory: "2Gi"
    cpu: "300m"

resourcesDefault:
  requests:
    memory: "128Mi"
    cpu: "100m"
  limits:
    memory: "128Mi"
    cpu: "100m"
  # We usually recommend not to specify default resources and to leave this as a conscious
  # choice for the user. This also increases chances charts run on environments with little
  # resources, such as Minikube. If you do want to specify resources, uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  # limits:
  #   cpu: 100m
  #   memory: 128Mi
  # requests:
  #   cpu: 100m
  #   memory: 128Mi

## App containers' Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-
a-container
##
## Containers should run as genesys user and cannot use elevated permissions
## Pod level security context
podSecurityContext:
  fsGroup: 500
  runAsUser: 500
  runAsGroup: 500
  runAsNonRoot: true

## Container security context
securityContext:
  # fsGroup: 500
  runAsUser: 500
  runAsGroup: 500
  runAsNonRoot: true

## Priority Class
## ref: https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
## NOTE: this is an optional parameter
##
priorityClassName: <critical-priority-class>

# affinity: {}

## Node labels for assignment.
## ref: https://kubernetes.io/docs/user-guide/node-selection/
##
nodeSelector:
  #genesysengage.com/nodepool: realtime

## Tolerations for assignment.
## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
##
tolerations: []
  # - key: "kubernetes.azure.com/scalesetpriority"
  #   operator: "Equal"
  #   value: "spot"
  #   effect: "NoSchedule"
  # - key: "k8s.genesysengage.com/nodepool"
  #   operator: "Equal"
  #   value: "compute"
  #   effect: "NoSchedule"
```

```
  # - key: "kubernetes.azure.com/scalesetpriority"
  #   operator: "Equal"
  #   value: "compute"
  #   effect: "NoSchedule"
  #- key: "k8s.genesysengage.com/nodepool"
  #  operator: Exists
  #  effect: NoSchedule

## Extra labels
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
##
## Use podLabels
#labels: {}

## Extra Annotations
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
##
## Use podAnnotations
#annotations: {}

## Autoscaling Settings
## Keda can be used as an alternative to HPA only to scale MCPs based on a cron schedule (UTC).
## If this is set to true, use Keda for scaling, or use HPA directly.
useKeda: true

## If Keda is enabled, only the following parameters are supported, and default HPA settings
## will be used within Keda.
keda:
  preScaleStart: "0 14 * * *"
  preScaleEnd: "0 2 * * *"
  preScaleDesiredReplicas: 4
  pollingInterval: 15
  cooldownPeriod: 300

## HPA Settings
# GVP-42512: PDB issue
# Alaways keep the following:
# minReplicas >= 2
# maxUnavailable = 1
hpa:
  enabled: false
  minReplicas: 2
  maxUnavailable: 1
  maxReplicas: 4
  podManagementPolicy: Parallel
  targetCPUAverageUtilization: 20
  scaleupPeriod: 15
  scaleupPods: 4
  scaleupPercent: 50
  scaleupStabilizationWindow: 0
  scaleupPolicy: Max
  scaledownPeriod: 300
  scaledownPods: 2
  scaledownPercent: 10
  scaledownStabilizationWindow: 3600
  scaledownPolicy: Min

## Service/Pod Monitoring Settings
prometheus:
  mcp:
    name: gvp-mcp-snmp
    port: 9116

  rup:
    name: gvp-mcp-rup
    port: 8080

  podMonitor:
    enabled: true

grafana:
```

```
  enabled: false

  #log:
  #  name: gvp-mcp-log
  #  port: 8200

## Pod Disruption Budget Settings
podDisruptionBudget:
  enabled: true

## Enable network policies or not
networkPolicies:
  enabled: false

## DNS configuration options
dnsConfig:
  options:
    - name: ndots
      value: "3"

## Configuration overrides
mcpConfig:
  # MCP config overrides
  mcp.mpc.numdispatchthreads: 4
  mcp.log.verbose: "interaction"
  mcp.mpc.codec: "pcmu pcma telephone-event"
  mcp.mpc.transcoders: "PCM MP3"
  mcp.mpc.playcache.enable: 1
  mcp.fm.http_proxy: ""
  mcp.fm.https_proxy: ""

  #MRCP v2 ASR config overrides
  mrcpv2_asr.provision.vrm.client.connectpersetup: true
  mrcpv2_asr.provision.vrm.client.disablehotword: false
  mrcpv2_asr.provision.vrm.client.hotkeybasepath: "/usr/local/genesys/mcp/grammar/nuance/hotkey"
  mrcpv2_asr.provision.vrm.client.noduplicatedgramuri: true
  mrcpv2_asr.provision.vrm.client.sendswmsparams: false
  mrcpv2_asr.provision.vrm.client.transportprotocol: "MRCPv2"
  mrcpv2_asr.provision.vrm.client.sendloggingtag: true
  mrcpv2_asr.provision.vrm.client.resource.name: "NuanceASRv2"
  mrcpv2_asr.provision.vrm.client.resource.uri: "sip:mresources@speech-server-clusterip:5060"
  mrcpv2_asr.provision.vrm.client.tlscertificatekey: "/usr/local/genesys/mcp/config/x509_certificate.pem"
  mrcpv2_asr.provision.vrm.client.tlsprivatekey: "/usr/local/genesys/mcp/config/x509_certificate.pem"
  mrcpv2_asr.provision.vrm.client.tlspassword: ""
  mrcpv2_asr.provision.vrm.client.tlsprotocoltype: "TLSv1"
  mrcpv2_asr.provision.vrm.client.confidencescale: 1
  mrcpv2_asr.provision.vrm.client.sendsessionxml: true
  mrcpv2_asr.provision.vrm.client.supportfornuance11: true
  mrcpv2_asr.provision.vrm.client.uniquegramid: true

  #MRCP v2 TTS config overrides
  mrcpv2_tts.provision.vrm.client.connectpersetup: true
  mrcpv2_tts.provision.vrm.client.speechmarkerencoding: "UTF-8"
  mrcpv2_tts.provision.vrm.client.transportprotocol: "MRCPv2"
  mrcpv2_tts.provision.vrm.client.sendloggingtag: true
  mrcpv2_tts.provision.vrm.client.resource.name: "NuanceTTSv2"
  mrcpv2_tts.provision.vrm.client.resource.uri: "sip:mresources@speech-server-clusterip:5060"
  mrcpv2_tts.provision.vrm.client.tlscertificatekey: "/usr/local/genesys/mcp/config/x509_certificate.pem"
  mrcpv2_tts.provision.vrm.client.tlsprivatekey: "/usr/local/genesys/mcp/config/x509_certificate.pem"
  mrcpv2_tts.provision.vrm.client.tlspassword: ""
  mrcpv2_tts.provision.vrm.client.tlsprotocoltype: "TLSv1"
  mrcpv2_tts.provision.vrm.client.nospeechlanguageheader: true
  mrcpv2_tts.provision.vrm.client.sendsessionxml: true
  mrcpv2_tts.provision.vrm.client.supportfornuance11: true
```


Verify the deployed resources

Verify the deployed resources from OpenShift console/CLI.

Provision GVP

> **Disclaimer**
>
> Genesys Voice Platform (GVP) in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

GVP Service Discovery (SD) container is used for provisioning tenant object in GVP Configuration Server (CS) and creating application objects for GVP Configuration Server and Media Control Platform (MCP) app objects under Resource Manager (RM) logical resource group.

GVP Service Discovery (SD) container (running in K8s) does the following:

1. Runs a timer that gets invoked every **`1 min`** by default (this is configurable).
2. Checks **Consul** for the registered MCPs and then checks **GVP Configuration Server (CS)** for the MCPs present there and does the necessary addition/removal of MCPs from CS to sync with Consul data.
3. Checks the **`tenant-inventory`** configmap in the **gvp** namespace of the **K8s cluster**, and if configmap is updated from the last run, then based on the new data creates/updates tenant information.

Note: SD processes one tenant at a time.

The following **objects** are created in CS as part of GVP tenant creation and, unless specified, SD uses default values for those objects:

- The **Tenant** object itself with properties set in the **Annex** section
- The following IVR Profiles:
  - IVRAppDefault
  - conference
  - cpd
  - record
  - media

For tenant creation, the following parameters **SHOULD** be specified:

- As part of tenant provisioning, the tenant is registered to **GWS** and you get a **GWS-CCID**. This parameter is **mandatory** for GVP tenant creation.
- The **id** for the tenant is a **mandatory** parameter. This can be an arbitrary string, but the preferable value is the **last 4-digits of the GWS-CCID**.
- The tenant **name** parameter is **preferred** to be populated.
- The **default-application** parameter should be set to **IVRAppDefault** always.

Note: GVP Configuration Server comes pre-loaded with the **Environment** tenant. It can also be used as your tenant if the plan is to use a single tenant only. In that case, it can't be recreated, but can be updated.

You deploy the Service Discovery container and provision a new tenant as part of the overall GVP deployment. as described on Deploy Genesys Voice Platform.

Deploy Designer

Disclaimer

Designer and Designer Application Server (DAS) in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

1 About this chapter

This document guides you through the process of deploying and configuring Designer and Designer Application Server (DAS) as a service in a Kubernetes (K8s) cluster.

Information on the following topics is provided:

- Overview of Designer and DAS
- Configuration
- Deployment process
- Enabling optional features
- Cleanup
- Known limitations

1.1 Intended audience

This document is intended primarily for system engineers and other members of an implementation team who will be configuring and installing the Designer and DAS components and for system administrators who will maintain Designer and DAS. To successfully deploy and implement applications in Designer and DAS, you must have a basic understanding of and familiarity with:

- Network design and operation.
- Your own network configurations.
- Kubernetes
- Genesys Framework architecture and functions

1.2 Before you begin

- Install Kubernetes according to the installation instructions on the Kubernetes documentation site. You can also refer to the Genesys Docker Deployment Guide for information about Kubernetes and High Availability.
- Install Helm according to the instructions outlined on the Helm documentation site.

Once you have completed these mandatory procedures, return to this manual to learn how to complete deployment of Designer and DAS.

2 Overview

This section provides an overview of the Designer and DAS components.

2.2 Designer

The Designer service provides a web UI to build and manage VXML and SCXML based self-service and assisted service applications for a number of media types. It stores data on the local file system which is synchronized across instances by using services like NFS. Genesys customers can build applications using a drag and drop interface, and assign contact points(Route Points and other media endpoints) to the applications from the Designer user interface. Insights into runtime behavior of applications and troubleshooting aid is provided by Designer Analytics which includes a rich set of dashboards based on session detail records (SDR) data stored in Elasticsearch.

Designer provides these features:

- Applications for working with phone, chat, email, sms, Facebook, Twitter and Open Media types
- Bots, ASR, TTS capabilities for self service
- Assisted service or routing
- Callback
- Business Controls
- Audio, message management
- Grammars management
- Contact points management - Route points, Chat End points, Email pop-client/mailboxes
- Analytics Dashboards via embedded Kibana

Designer is an Express/Node.js application. The UI is developed using angular/bootstrap. Application data (SCXML and VXML) is stored as a filesystem. Designer Analytics and Audit data are stored in Elasticsearch.

2.2 Designer Application Server (DAS)

Designer Application Server (DAS) hosts and serves the Designer generated application files (VXML/SCXML), audio, and grammars. It also provides,

- Runtime evaluation of Business Controls (Busines hours, special days, emergency flags and data tables).
- Callback interface to GES.

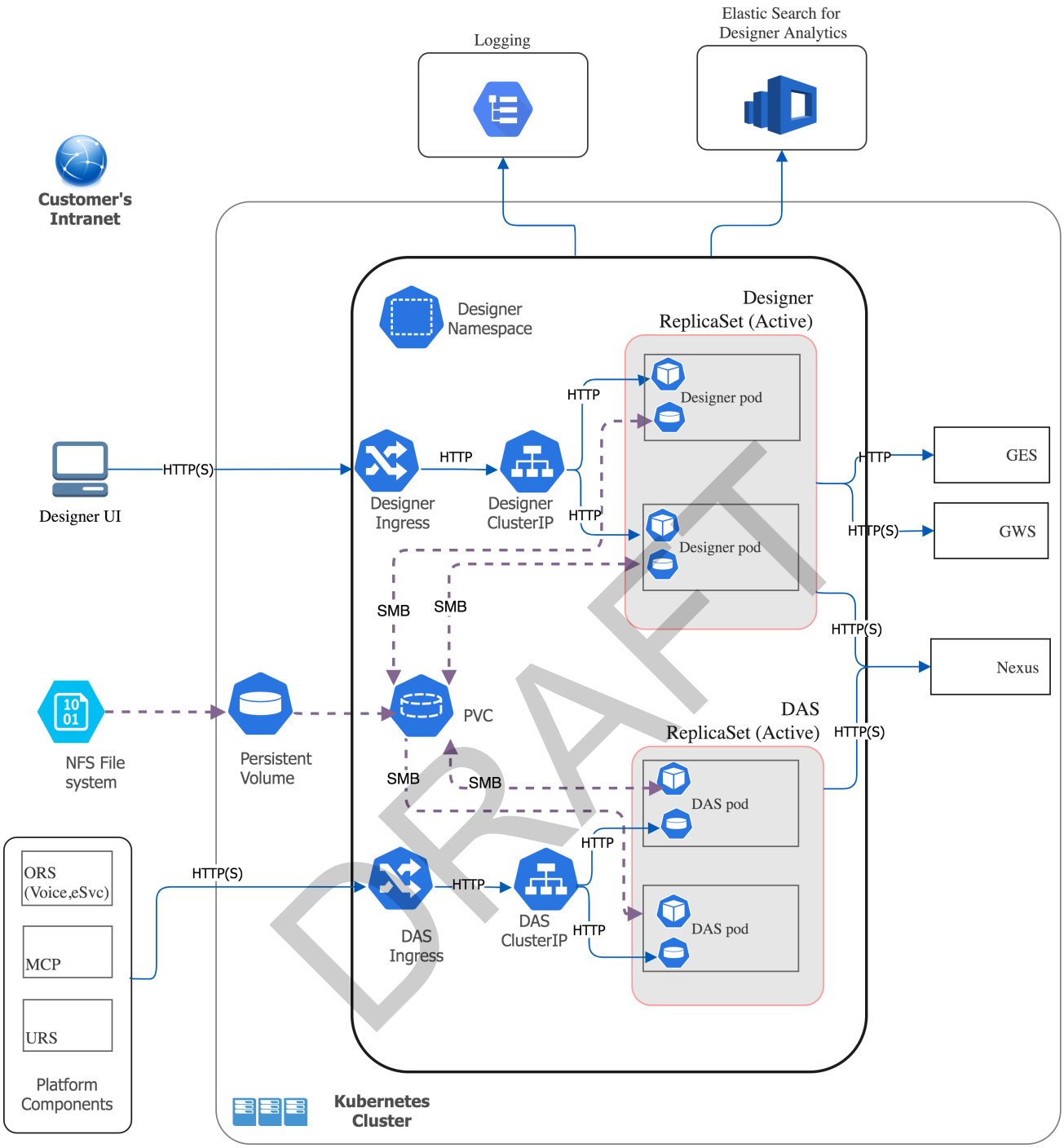
DAS uses built-in NGINX to front requests. It contains 3 modules NGINX, PHP, and Node.js.

- Requests for static workspace content(scxml, vxml, js, audio, grammar, etc) are handled by the NGINX module.
- Requests for PHP are processed by the FastCGI PHP module.
- SDR(Analytics) processing requests are handled by the DAS Node.js module.

Designer generated files can be served only by DAS, i.e. Designer will work only with DAS.

2.3 Deployment architecture

The below architecture diagram illustrates a sample deployment of Designer and DAS:



2.4 High Availability(HA)/DR and Scalability

Designer and DAS should be deployed as High Availability deployment to avoid a single point of failure. There should be a minimum of 2 replicas of each service to be deployed to achieve HA.

The Designer and DAS service pods can be auto-scaled up/down based on metrics such as CPU and memory utilization

The Configuration section explains how to configure HA and auto-scaling.

This document provides general HA recommendation for Kubernetes - https://docs.genesys.com/Documentation/System/8.5.x/DDG/HAandDRNotes.

3 Prerequisites

The table below outlines all the prerequisites for Designer and DAS deployment using Kubernetes:

Component	Description	Mandatory or Optional
Kubernetes	1.12+	Mandatory
Helm	3.0	Mandatory
Docker	Docker to store Designer and DAS Docker images to the local Docker registry.	Mandatory
Ingress Controller	• The Designer and DAS are accessed from outside of K8s cluster, it is recommended to deploy/configure an ingress controller(fo example, NGINX). Also the blue-green deployment strategy works based on the ingress rules. • Designer UI requires Session Stickiness, Configure session stickiness in *annotations* of values.yaml during Designer installation.	Mandatory
Persistent volumes(PVs)	• Persistent volumes should be created for workspace storage (5 GB minimum) and logs (minimum 5GB). • The Persistent volumes should be created with ReadWriteMany accessModes. • Designer manifest package includes a sample YAML file to create Persistent Volumes required for Designer and DAS. • Persistent volumes should be shared across multiple K8s nodes, it is recommended to use NFS to create Persistent Volumes.	Mandatory
Shared file system	NFS • For production, the NFS server should be deployed as High availability(HA) deployment in order to avoid a single point of failure. It is also recommended for NFS storage to be deployed in Disaster Recovery(DR) topology to achieve continues availability incase one region fails. • By Default, Designer and DAS containers run as a Genesys user (uid:gid 500:500). For this reason, the shared volume should have permissions that will allow write access to uid:gid 500:500, the best way is to change the NFS server host path to the Genesys user: `chown -R genesys:genesys.` • Designer manifest package includes a sample YAML file to create NFS server, it can be used only for demo/lab setup purpose. Azure Files Storage If you opt for Cloud storage, then Azure Files Storage is an option to consider and has the following requirements: A Zone-Redundant Storage for RWX volumes replicated data in zone redundant (check this), shared across multiple pods. Provisioned capacity : 1 TiB Baseline IO/s : 1424 Burst IO/s : 4000 Egress Rate : 121.4 MiBytes/s Ingress Rate : 81.0 MiBytes/s	Mandatory
Genesys Web Services (GWS)	v.9.x • Configure to work with a compatible version of Configuration Server.	Mandatory
Other Genesys components	• Authentication Service • Voice Microservices	Mandatory
Elasticsearch	v7.8.0 It is used for Designer Analytics and audit trails.	Optional
Redis	v3.2.x Used by Designer for resource index caching and multi-user collaboration locks on Designer resources.	Optional for lab only.

4 Configuration

This section explains various settings configured in Designer/DAS.

4.1 Deployment settings (Helm values)

The below deployment settings will be used during the deployment. ie. it will be used during the initial deployment/upgrade. These settings can be configured in helm values.yaml

4.1.1 Designer deployment settings

The below table provides a reference to the Designer deployment settings. This will be configured in designer-values.yaml

Parameter	Description	Mandatory	Default
`designer.deployment.replicaCount`	Number of service instances to be created.	Optional	2
`designer.deployment.maxreplicaCount`	The maximum number of replicas created. It is recommended to configure if auto-scaling is used.	Optional	10
`designer.deployment.strategy`	The strategy to select which type of resources to deploy, Valid values are: • *rollingupdate* - default kubernetes update strategy where resources will be updated using the rolling upgrade strategy. • *blue-green* - for deploying and upgrading designer service using blue/green strategy. • *blue-green-volume* - for blue/green upgrade, this is to create persistent volume claim (pvc) for the very first time. • *blue-green-ingress* - for the blue/green upgrade, this is to create ingress for the first time and update ingress during service cutover.	Mandatory	rollingupdate
`designer.deployment.color`	This is to deploy/upgrade the Designer service in blue-green upgrade strategy. Valid values : *blue/green.*	Mandatory for *blue-green* and *blue-green- ingress*	
`designer.deployment.type`	This is to specify the type of deployment. Valid values: *Deployment.*	Optional	Deployment
`designer.image.registry`	The registry that the organization uses for storing images.	Mandatory	
`designer.image.repository`	Docker repository that contains the images for Designer.	Mandatory	
`designer.image.tag`	Designer Image version.	Mandatory	9.0.110.07.7
`designer.image.pullPolicy`	Designer Image pull policy(imagePullPolicy), Valid values : Always / IfNotPresent / Never. • Always - Always pull the image. • IfNotPresent - Only pull the image if it does not already exist on the node • Never - Never pull the image	Mandatory	IfNotPresent
`designer.image.imagePullSecrets`	Secret name containing credentials for authenticating to the Docker repository.	Mandatory	
`designer.volumes.workspacePvc.create`	If the volume is to be created, this value has to be true or else false.	Mandatory	true
`designer.volumes.workspacePvc.mountPath`	The path where the workspace volume to be mounted inside designer container.	Mandatory	/designer/workspace
`designer.volumes.workspacePvc.claim`	Persistent volume claim name for the workspace.	Mandatory	designer-managed-disk
`designer.volumes.workspacePvc.claimSize`	Size of the Persistent volume claim for the workspace. The persistent volume should be created equivalent or greater size.	Mandatory	
`designer.volumes.workspacePvc.storageClass`	storageClassName provided in the Persistent volume that is created for the Designer workspace (e.g. nfs).	Mandatory	
`designer.volumes.logsPvc.create`	If the volume is to be created, this value has to be true or else false.	Mandatory	true
`designer.volumes.logsPvc.mountPath`	The path where the Designer logs volume to be mounted inside designer container.	Mandatory	/designer/logs
`designer.volumes.logsPvc.claim`	Persistent volume claim name for logs.	Mandatory	designer-logs

`designer.volumes.logsPvc.claimSize`	Size of the Persistent volume claim for the Designer logs. The persistent volume should be created equivalent or greater size.	Mandatory	
`designer.volumes.logsPvc.storageClass`	storageClassName provided in the Persistent volume that is created for the Designer logs (e.g. nfs).	Mandatory	

`designer.podVolumes`	Log and workspace persistent volume claim names and name of the volumes attached to the pod.	Mandatory	`designer:`
			`  podVolumes:`
			`    - name: designer-pv-volume`
			
			`persistentVolumeClaim:`
			
			`claimName: designer-managed-disk`
			`    - name: designer-log-volume`
			
			`persistentVolumeClaim:`
			
			`claimName: designer-logs`

`designer.volumeMounts`	Name and mount path of the volumes to be attached to the Designer pods.	Mandatory	**designer.**
			volumeMounts
			
			`volumeMounts:`
			`    - name: designer-pv-volume`
			
			`mountPath: /designer/workspace`
			`    - name: designer-log-volume`
			
			`mountPath: /designer/logs`

`designer.livenessProbe.path`	Designer liveness probe API path.	Mandatory	/health
`designer.livenessProbe.containerPort`	Container running port.	Mandatory	8888
`designer.livenessProbe.startupDelay`	Liveness Probe will be started after a given delay.	Mandatory	20
`designer.livenessProbe.checkInterval`	The interval between each liveness probe requests.	Mandatory	5
`designer.livenessProbe.failureCount`	No of liveness probe failures to mark the container as unstable or restart.	Mandatory	5
`designer.readinessProbe.path`	Designer readiness probe API path for readiness probe.	Mandatory	/health

`designer.readinessProbe.containerPort`	Container running port.	Mandatory	8888
`designer.readinessProbe.startupDelay`	readinessProbe will be started after a given delay.	Mandatory	20
`designer.readinessProbe.checkInterval`	The interval between each health check requests.	Mandatory	5
`designer.readinessProbe.failureCount`	No of readiness probe failure to mark the container as unstable or restart.	Mandatory	5
`designer.designerSecrets.enabled`	This enables to provide GWS client id/secret as an input to Designer pods. It uses Kubernetes Secrets to store the GWS client credentials.	Mandatory	true
`designer.designerSecrets.secrets`	GWS Client id and GWS Client secret, create a new GWS client if it doesn't exist, steps are explained in the platform settings section.	Mandatory	
`designer.service.enabled`	True if service needs to be created or else false.	Optional	true
`designer.service.type`	Service type either ClusterIP/NodePort/LoadBalancer.	Mandatory	NodePort
`designer.service.port`	Designer service port to be exposed in the cluster.	Mandatory	8888
`designer.service.targetPort`	Designer application port running inside the container.	Mandatory	http
`designer.service.nodePort`	Port to be exposed in case service.type=NodePort.	Mandatory for designer.service.type=Nodeport	30180
`designer.service.termination_grace_period`	The period after which Kubernetes starts to delete the pods in case of deletion.	Optional	30 seconds
`designer.ingress.enabled`	Enable/Disable ingress. Ingress should be enabled for all cases except lab/demo setup.	Mandatory	true
`designer.ingress.annotations`	Annotations added for Ingress. Designer UI requires Session Stickiness if the replica count is more than 1, Configure session sickness based on the ingress controller type. Ingress configuration like session stickiness can be configured here.	Optional	
`designer.ingress.paths`	Ingress path.	Mandatory	["/"]
`designer.ingress.hosts`	Hostnames to be configured in ingress for Designer service.	Mandatory	- ssdev1.genhtcc.com
`designer.ingress.tls`	TLS config for ingress.	Optional	[]
`designer.resources.limits.cpu`	Maximum amount of CPU K8s allocates for container.	Mandatory	600m
`designer.resources.limits.memory`	Maximum amount of Memory K8s allocates for container.	Mandatory	1Gi
`designer.resources.requests.cpu`	Guaranteed CPU allocation for container.	Mandatory	500m
`designer.resources.requests.memory`	Guaranteed Memory allocation for container.	Mandatory	512Mi
`designer.securityContext.runAsUser`	Controls which user ID the containers are run with. This can be configured to run Designer as a non-root user. Currently, only a **Genesys** user is supported by the Designer base image. 500 is the ID of the Genesys user and cannot be modified. The file system must reside within the Genesys user in order to run Designer as a Genesys user. Change the NFS server host path to the Genesys user: `chown -R genesys:genesys`	Optional	

`designer.securityContext.runAsGroup`	Controls which primary group ID the containers are run with. This can be configured to run Designer as a non-root user. Currently, only a **Genesys** userGroup(Gid -500) is supported by the Designer base image.	Optional	

`designer.nodeSelector`	To allow Pods to be scheduled on the nodes based labels assigned to nodes.	Optional	Default value:
			`nodeSelector: {}`
			Sample value:
			`nodeSelector:`
			`  <label_key>:`
			`<label_value>`

`designer.affinity`	The K8s standard node Affinity and anti-affinity configurations can be added here. Refer K8s doc for sample values. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/	Optional	{}
`designer.tolerations`	Tolerations work with taints to ensure that pods are not scheduled onto inappropriate nodes. Refer K8s doc for sample values. https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	Optional	[]
`designer.podDisruptionBudget.enabled`	True if Pod disruption budget is to be created.	Optional	false
`designer.podDisruptionBudget.minAvailable`	The number of pods that should always be available during a disruption.	Optional	1
`designer.dnsPolicy`	Should contain the DNS policy that should be applied to the Designer pods.	Optional	
`designer.dnsConfig`	DNS Config that should be applied to the Designer Pods.	Optional	
`designer.priorityClassName`	The priority Class Name that the pods should belong to.	Optional	
`designer.networkPolicies.enabled`	True if network policies are to be created, currently, Designer doesn't support Network Policies, this value is reserved for future use.	Optional	false
`designer.hpa.enabled`	Enables K8s Horizontal Pod Autoscaler. It automatically scales the number of Pods based on average CPU utilization, average memory utilization. More info about hpa - https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/	Optional	false
`designer.hpa.targetCPUPercent`	K8s HPA controller will scale up/down pods based on the target CPU utilization percentage specified. It scales up/down pods between the range - `deployment.replicaCount to deployment.maxReplicas.`	Optional	70
`designer.hpa.targetMemoryPercent`	K8s HPA controller will scale up/down pods based on the target memory utilization percentage specified. It scales up/down pods between the range - `deployment.replicaCount to deployment.maxReplicas.`	Optional	70
`designer.labels`	Labels that will be added to the Designer pods.	Optional	{}
`designer.annotations`	Annotations added to the Designer pods.	Optional	{}
`designer.prometheus.enabled`	True if Prometheus metrics are to be enabled or else false.	Optional	false
`designer.prometheus.instance`		Optional	`"{{instance}}"`
`designer.prometheus.tagName`		Optional	service
`designer.prometheus.tagValue`			designer

`designer.prometheus.serviceMonitor.enabled`	True if service monitor resource is needed to monitor the pods through the Kubernetes service.	Optional	false
`designer.prometheus.serviceMonitor.path`	The path in which the metrics are exposed.	Optional	/metrics
`designer.prometheus.serviceMonitor.interval`	The scrape interval specified for the prometheus server i.e., the time interval in which the prometheus server will fetch metrics from the service.	Optional	10
`designer.prometheus.serviceMonitor.labels`	Labels to be specified for the service monitor resource.	Optional	
`designer.prometheus.alerts.enabled`	True if prometheus alerts need to be created.	Optional	false
`designer.prometheus.alerts.labels`	Labels to be specified for the alerts resource.	Optional	

`designer.prometheus.alerts.<alert_names>`	Scenarios for which alerts need to be created.	Optional	**designer.prometheus.alerts**
			`containerRestartAlert:`
			`interval: 3m`
			`threshold: 5`
			`AlertPriority: CRITICAL`
			`MemoryUtilization:`
			`interval: 1m`
			`threshold: 70`
			`AlertPriority: CRITICAL`
			`endpointAvailable:`
			`interval: 1m`
			`AlertPriority: CRITICAL`
			`CPUUtilization:`
			`interval: 1m`
			`threshold: 70`
			`AlertPriority: CRITICAL`
			`containerReadyAlert:`
			`interval: 1m`
			`readycount: 1`

			AlertPriority: CRITICAL WorkspaceUtilization: interval: 3m threshold: 80 workspaceClaim: designer-managed-disk AlertPriority: CRITICAL AbsentAlert: interval: 1m AlertPriority: CRITICAL Health: interval: 3m AlertPriority: CRITICAL WorkspaceHealth: interval: 3m AlertPriority: CRITICAL ESHealth: interval: 3m AlertPriority: CRITICAL GWSHealth: interval: 3m AlertPriority: CRITICAL
annotations	Enables Kubernetes annotations and adds it to all created resources. https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/	Optional	{}
labels	Any custom labels can be configured. It is a key and value, for e.g. key:"value". These Labels are added to all resources.	Optional	{}
podLabels	Labels that would be added to all application pods.	Optional	{}
podAnnotations	Annotations that would be added to all application pods.	Optional	{}

4.1.2 Designer config map settings

Parameter	Description	Mandatory	Default
designer.designerConfig.create	This enables to provide environment variables as an input to Designer pods. It uses ConfigMap to store the environment variables	Mandatory	true

`designer.designerConfig.envs.DES_PORT`	Designer port for container ("port" in flowsettings.json).	Mandatory	8888
`designer.designerConfig.envs.DES_APPSERVER_HOST`	DAS hostname ("applicationHost" in flowsettings.json).	Mandatory	das
`designer.designerConfig.envs.DES_APPSERVER_PORT`	DAS port ("applicationPort" in flowsettings.json).	Mandatory	80
`designer.designerConfig.envs.DES_DEPLOY_URL`	This is normally not changed. It is the relative path to the workspace on DAS. The default value "/workspaces" should be used always("deployURL" in flowsettings.json).	Mandatory	/workspaces
`designer.designerConfig.envs.DES_USE_HTCC`	Set to "true" so Designer works with GWS. If set to "false" Designer defaults to a local mode and may be used temporarily if GWS is unavailable ("usehtcc" in flowsettings.json).	Mandatory	false
`designer.designerConfig.envs.DES_HTCC_SERVER`	GWS server host ("htccserver" in flowsettings.json) for e.g. gws.genhtcc.com.	Mandatory	
`designer.designerConfig.envs.DES_HTCC_PORT`	GWS server port ("htccport" in flowsettings.json)or e.g. 80.	Mandatory	80
`designer.designerConfig.envs.DES_ENABLE_ANALYTICS`	To enable or disable Designer analytics ("enableAnalytics" in flowsettings.json).	Optional	false
`designer.designerConfig.envs.DES_ES_URL`	Elasticsearch URL (for ex. http://es-service:9200), "esUrl" in flowsettings.json.	Optional	
`designer.designerConfig.envs.DES_ES_SERVER`	Elasticsearch Server HostName (for ex. es-service), "esServer" in flowsettings.json.	Optional	
`designer.designerConfig.envs.DES_ES_PORT`	Elasticsearch port (for ex. 9200), "esPort" in flowsettings.json.	Optional	80
`designer.designerConfig.envs.DES_FILE_LOGGING_ENABLED`	Enable file logging. If not enabled, Designer will create verbose logs.	Mandatory	false

4.1.3 Designer Application Server (DAS) deployment settings

These settings are configured in das-values.yaml.

Parameter	Description	Mandatory	Default
`das.deployment.replicaCount`	No. of pods to be created.	Mandatory	2
`das.deployment.maxreplicaCount`	The maximum number of replicas created. It is recommended to configure if auto-scaling is used.	Optional	10
`das.deployment.strategy`	The strategy to select which type of resources to deploy, Valid values are: • *rollingupdate* - default kubernetes update strategy where resources will be updated using rolling upgrade strategy. • *blue-green* - for deploying and upgrading DAS service using blue/green strategy. • *blue-green-ingress* - for the blue/green upgrade, this is to create ingress for the first time. • *blue-green-service* - for the blue/green upgrade, this is to create service for the first time and update service during service cutover. • *canary* - to deploy canary pods along with blue-green pods.	Mandatory	rollingupdate
`das.deployment.color`	This is to deploy/upgrade DAS service in blue-green upgrade strategy. Valid values : *blue /green.*	Mandatory for *blue-green* and *blue-green-service* strategies	
`das.deployment.type`	Type of Kubernetes controller. Valid values: Deployment/StatefulSet. • StatefulSet - If the Designer workspace is stored in a remote cloud storage. For instance, Azure Files.	Optional	Deployment
`das.image.registry`	The registry that the organisation uses for storing images.	Mandatory	
`das.image.repository`	Docker repository for DAS.	Mandatory	
`das.image.tag`	DAS Image version.	Mandatory	9.0.106.03.7
`das.image.pullPolicy`	DAS Image pull policy(imagePullPolicy), Valid values: Always / IfNotPresent / Never. • Always - Always pull the image. • IfNotPresent - Only pull the image if it does not already exist on the node. • Never - Never pull the image.	Optional	IfNotPresent
`das.image.imagePullSecrets`	Secret name containing credentials for authenticating to the Docker repository.	Mandatory	

`das.podVolumes`	Provides the name of the volume and name of the persistent volume claim to be attached to the pods.	Mandatory	`das:`
			`  podVolumes:`
			`  - name: workspace`
			`  persistentVolumeClaim:`
			`  claimName: designer-managed-disk`
			`  - name: logs`
			`  persistentVolumeClaim:`
			`  claimName: designer-logs`

`das.volumes.podPvc.create`	If the volume is to be created, this value has to be true or else false.	Optional	false
`das.volumes.podPvc.mountPath`	The path where the volume to be mounted inside DAS container.	Optional	
`das.volumes.podPvc.claim`	Persistent volume claim name for the volume.	Optional	
`das.volumes.podPvc.claimSize`	Size of the Persistent volume claim. The persistent volume should be equivalent to or greater than the value specified here.	Optional	
`das.volumes.podPvc.storageClass`	storageClassName provided in the Persistent volume that is created for DAS (e.g. nfs).	Optional	
`das.volumes.podPvc.accessModes`	The read/write priveleges and mount priveleges of the volume claim with respect to nodes. Valid types: • ReadWriteOnce -- the volume can be mounted as read-write by a single node. • ReadOnlyMany -- the volume can be mounted read-only by many nodes. • ReadWriteMany -- the volume can be mounted as read-write by many nodes. https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes	Optional	ReadWriteOnce

`das.volumeMounts`	The name of the volume and the mount path to be used by the pods.	Mandatory	`volumeMounts:`
			`- mountPath: /das/www/workspaces`
			`name: workspace`
			`- mountPath: /das/log`
			`name: logs`

`das.dasSecrets.enabled`	True if Kubernetes secrets need to be created to store any keys/credentials/token or else false.	Optional	false
`das.dasSecrets.secrets`	Key value pairs having the secret such as username and password etc.	Optional	
`das.livenessProbe.path`	DAS Liveness Probe API path.	Mandatory	/health
`das.livenessProbe.containerPort`	Container running port.	Mandatory	8081
`das.livenessProbe.startupDelay`	Liveness Probe will be started after a given delay.	Mandatory	10
`das.livenessProbe.checkInterval`	The interval between each Liveness Probe requests.	Mandatory	5

`das.livenessProbe.failureCount`	No of Liveness Probe failure to mark the container as instable or restart.	Mandatory	3
`das.readinessProbe.path`	DAS Readiness probe API path.	Mandatory	/health
`das.readinessProbe.containerPort`	Container running port.	Mandatory	8081
`das.readinessProbe.startupDelay`	Readiness probe will be started after a given delay.	Mandatory	10
`das.readinessProbe.checkInterval`	The interval between each readiness probe requests.	Mandatory	5
`das.readinessProbe.failureCount`	No of readiness probe failure to mark the container as instable or restart.	Mandatory	3
`das.service.enabled`	Set to true if service is to be created and false if not.	Optional	true
`das.service.type`	Service port either ClusterIP/NodePort/LoadBalancer.	Mandatory	NodePort
`das.service.port`	DAS service to be exposed in the cluster.	Mandatory	80
`das.service.targetPort`	DAS application port running inside the container.	Mandatory	http
`das.service.nodePort`	Port to be exposed in case service.type=NodePort.	Mandatory if das.service.type is NodePort	30280
`das.service.termination_grace_period`	The period after which Kubernetes starts to delete the pods in case of deletion.	Optional	30 seconds
`das.ingress.enabled`	Enable/Disable ingress. Ingress should be enabled for all cases except lab/demo setup.	Mandatory	true
`das.ingress.annotations`	Annotations added for Ingress resources.	Optional	
`das.ingress.paths`	Ingress path.	Mandatory	["/"]
`das.ingress.hosts`	Hostnames to be configured in ingress for DAS service.	Mandatory	
`das.ingress.tls`	TLS config for ingress.	Optional	[]
`das.resources.limits.cpu`	Maximum amount of CPU K8s allocates for container.	Mandatory	600m
`das.resources.limits.memory`	Maximum amount of Memory K8s allocates for container.	Mandatory	1Gi
`das.resources.requests.cpu`	Guaranteed CPU allocation for container.	Mandatory	400m
`das.resources.requests.memory`	Guaranteed Memory allocation for container.	Mandatory	512Mi
`das.securityContext.runAsUser`	Controls which user ID the containers are run with. This can be configured to run DAS as a non-root user. Currently, only a **Genesys** user is supported by the DAS base image. 500 is the ID of the Genesys user and cannot be modified. For other security context, visit: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/	Optional	
`das.securityContext.runAsGroup`	Controls which primary group ID the containers are run with. This can be configured to run DAS as a non-root user. Currently, only a **Genesys** userGroup(Gid -500) is supported by the DAS base image.	Optional	

`das.nodeSelector`	To allow Pods to be scheduled on the nodes-based labels assigned to nodes.	Optional	Default value:
			`nodeSelector: {}`
			Sample value:
			`nodeSelector:`
			`<label_key>: <label_value>`

`das.affinity`	The K8s standard node Affinity and anti-affinity configurations can be added here. Refer K8s doc for sample values. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/	Optional	{}
`das.tolerations`	Tolerations works with taints to ensure that pods are not scheduled onto inappropriate nodes. Refer K8s doc for sample values. https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	Optional	[]
`das.podDisruptionBudget.enabled`	True if Pod disruption budget is to be created.	Optional	false
`das.podDisruptionBudget.minAvailable`	The number of pods that should always be available during a disruption.	Optional	1
`das.dnsPolicy`	Should contain the DNS policy that should be applied to the Designer pods.	Optional	
`das.dnsConfig`	DNS Config that should be applied to the Designer Pods.	Optional	
`das.priorityClassName`	The priority Class Name that the pods should belong to.	Optional	
`das.networkPolicies.enabled`	True if network policies are to be created, currently, DAS doesn't support Network Policies, this value is reserved for future use.	Optional	false
`das.hpa.enabled`	True if HPA is to be created or else false.	Optional	false
`das.hpa.targetCPUPercent`	K8s HPA controller will scale up/down pods based on the target CPU utilization percentage specified. It scale up/down pods between the range - `deployment.replicaCount to deployment.maxReplicas.`	Optional	75
`das.hpa.targetMemoryPercent`	K8s HPA controller will scale up/down pods based on the target memory utilization percentage specified. It scale up/down pods between the range - `deployment.replicaCount to deployment.maxReplicas.`	Optional	70
`das.labels`	Labels that will be added to DAS pods.	Optional	{}
`das.annotations`	Annotations added to DAS pods.	Optional	{}
`das.prometheus.enabled`	True if Prometheus metrics are to be enabled or else false.	Optional	false
`das.prometheus.instance`		Optional	`"{{instance}}"`
`das.prometheus.tagName`		Optional	service
`das.prometheus.tagValue`		Optional	designer
`das.prometheus.serviceMonitor.enabled`	True if service monitor resource is needed to monitor the pods through the Kubernetes service.	Optional	false
`das.prometheus.serviceMonitor.path`	The path in which the metrics are exposed.	Optional	/metrics
`das.prometheus.serviceMonitor.interval`	The scrape interval specified for the prometheus server i.e., the time interval in which the prometheus server will fetch metrics from the service.	Optional	10

`das.prometheus.serviceMonitor.labels`	Labels to be specified for the service monitor resource.	Optional	
`das.prometheus.alerts.enabled`	True if prometheus alerts need to be created.	Optional	false
`das.prometheus.alerts.labels`	Labels to be specified for the alerts resource.	Optional	

`das.prometheus.alerts.<alert_names>`	Scenarios for which alerts need to be created.	Optional	`containerRestartAlert:`
			`interval: 3m`
			`threshold: 5`
			`AlertPriority: CRITICAL`
			`MemoryUtilization:`
			`interval: 1m`
			`threshold: 75`
			`AlertPriority: CRITICAL`
			`endpointAvailable:`
			`interval: 1m`
			`AlertPriority: CRITICAL`
			`CPUUtilization:`
			`interval: 1m`
			`threshold: 75`
			`AlertPriority: CRITICAL`
			`containerReadyAlert:`
			`interval: 5m`
			`readycount: 1`
			`AlertPriority: CRITICAL`
			`rsyncContainerReadyAlert:`
			`interval: 5m`
			`readycount: 1`
			`AlertPriority: CRITICAL`
			`WorkspaceUtilization:`
			`interval: 3m`
			`threshold: 70`
			`workspaceClaim: designer-managed-disk`
			`AlertPriority: CRITICAL`
			`AbsentAlert:`
			`interval: 1m`
			`AlertPriority: CRITICAL`
			`LocalWorkspaceUtilization:`
			`interval: 3m`
			`threshold: 70`
			`AlertPriority: CRITICAL`
			`Health:`
			`interval: 3m`
			`AlertPriority: CRITICAL`
			`WorkspaceHealth:`
			`interval: 3m`
			`AlertPriority`

			`: CRITICAL PHPHealth: interval: 3m AlertPriority : CRITICAL ProxyHealth: interval: 3m AlertPriority : CRITICAL`
`annotations`	Enables Kubernetes annotations and adds to all resources created. https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/	Optional	{}
`labels`	Any custom labels can be configured. It is a key and value, for e.g. key:"value". These labels are added to all the resources created.	Optional	{}
`podLabels`	Labels that would be added to all application pods.	Optional	{}
`podAnnotations`	Annotations that would be added to all application pods.	Optional	{}

4.1.4 DAS config map settings

Parameter	Description	Mandatory	Default
`das.dasConfig.create`	This enables to provide environment variables as an input to DAS pods. It uses ConfigMap to store the environment variables	Mandatory	true
`das.dasConfig.envs.DAS_FILE_LOGGING_ENABLED`	Enable file logging. DAS supports only std out logging, this should be false always.	Mandatory	false
`das.dasConfig.envs.DAS_LOG_LEVEL`	Enable log levels, valid values : FATAL, ERROR, WARN, INFO, DEBUG, TRACE.	Optional	DEBUG
`das.dasConfig.envs.` `DAS_STDOUT_LOGGING_ENABLE`	Enable standard output console logging.	Mandatory	true
`das.dasConfig.envs.` `DAS_SERVICES_ELASTICSEARCH_ENABLED`	To enable or disable Designer analytics. This config is required for DAS to initialize ES templates .	Optional	false
`das.dasConfig.envs.` `DAS_SERVICES_ELASTICSEARCH_HOST`	Elasticsearch Server HostName with http:// prefix(for ex. http://es-service).	Optional	
`das.dasConfig.envs.` `DAS_SERVICES_ELASTICSEARCH_PORT`	Elasticsearch port (for ex. 80).	Optional	9200

4.1.5 Designer-Sync settings

If the Designer workspace is stored in cloud storage then DAS should use the 'StatefulSet' Deployment type. In this case, designer-sync containers will be used to sync the designer workspace from cloud storage to high-performance disks attached to DAS pods.

Parameter	Description	Mandatory	Default
Parameter	**Description**	**Mandatory**	**Default**
designerSync.image.registry	Image registry of designer-sync container	Yes	
designerSync.image.repository	Image repository of designer-sync container	Yes	
designerSync.image.tag	designer-sync image version	Yes	
designerSync.resources.requests.cpu	CPU resource requests needed for designer-sync container	Yes	
designerSync.resources.requests.memory	Memory resource requests needed for designer-sync container	Yes	
designerSync.resources.limits.cpu	CPU resource limit set for designer-sync container	Yes	
designerSync.resources.limits.memory	Memory resource limit set for designer-sync container	Yes	
designerSync.livenessProbe.healthScript	Script path needs to be executed to check the health of the container	Yes	/workspace_sync/health.sh
designerSync.livenessProbe.timeoutSeconds	Timeout for the livenessprobe check	Yes	5
designerSync.livenessProbe.checkInterval	Interval of executing successive liveness probe check	Yes	10
designerSync.livenessProbe.failureCount	No of failures to be accounted before deciding container health is not good	Yes	3
designerSync.volumeMounts	volumes and mounts configuration for designer-sync image	Yes	

4.1.6 Designer-Sync config map settings

Parameter	Description	Mandatory	Default
designerSync.designerSyncConfig.create	Whether to create configmap for designer-sync container	Yes	true
designerSync.designerSyncConfig.envs.MODE	designer-sync mode, there are 2 modes 1. initial - imports the designer workspace from cloud file system to the disks attached to DAS pods. 2. periodic - periodically syncing the incoming changes in the designer workspace to disks attached to the DAS pods.	Yes	periodic
designerSync.designerSyncConfig.envs.LOCATION	Location name like westus2	No	
designerSync.designerSyncConfig.envs. RSYNC_INTERVAL	Interval perodic rsync execution.	No	60
designerSync.designerSyncConfig.envs. WORKSPACE_RSYNC_MOUNT_0	Persistent volume mount name from where the data needs to be synced	Yes	
designerSync.initialDesignerSyncConfig.create	Whether to create configmap for designer-sync container for initial import	Yes	True
designerSync.initialDesignerSyncConfig.envs.MODE	initial mode	Yes	initial
designerSync.initialDesignerSyncConfig.envs. LOCATION	location of the cloud deployment	No	
designerSync.initialDesignerSyncConfig.envs. WORKSPACE_RSYNC_MOUNT_0	Persistent volume mount name from where the data needs to be imprted	Yes	

4.2 Designer post deployment settings

Post deployment, Designer configuration is managed in the following 3 locations:

4.2.1 Flowsettings

This is used for controlling global Designer settings that are applicable to all tenants and it contains the bootstap config like port, GWS info, DAS URL etc.

Config path - <workspace>/designer/flowsettings.json.

This will be configured using helm install, the deployment section "6.8 Flowsettings.json update" explains the steps to update flowsettings.json.

4.2.2 Tenantsettings

These are the tenant specific settings if the Designer is configured with multi-tenancy.

Config path - <workspace>/<contactcenter-Id>/config/tenantsettings.json.

The user should logout and log back in. Otherwise the Designer UI will continue to show the older features.

This will be configured by editing the file in the above path.

4.2.3 DesignerEnv

DesignerEnv transaction list in Configuration Server (Tenant/Transactions/Internal/DesignerEnv). This is mostly used to control the run time settings, any change in the DesignerEnv list does't require application publish/new build.

The user should be logout and log back in otherwise the Designer UI will continue to show the older features.

This will be configured by login to Agent Setup.

4.2.4 Configuration reference table

Category	Setting Name	Mandatory	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample value	Default value
Analytics	enableAnalytics	optional	yes	yes			This flag enables or disables analytics	true	false
	esUrl	optional	yes	yes			Elasticsearch URL	http://es-spot.usw1.genhtcc.com:80	

	esServer	optional	yes	yes			Elasticsearch Server HostName (for ex. es-service)	es-spot.usw1.genhtcc.com	
	esPort	optional	yes	yes			Elasticsearch port	80	
	ReportingURL	optional			yes	reporting	URL of Elasticsearch where Designer applications will report data	http://es-spot.usw1.genhtcc.com:80	
	esMaxQueryDuration	optional	yes	yes			The maximum time range (in days) to query in Designer Analytics. Each day's data is stored in a separate index in Elasticsearch.	90	90
	sdrMaxObjCount	optional	yes	yes			The maximum count of nested type objects will be captured in SDRs. When set to -1, which is the default value, meaning no objects will be trimmed, all the *milestones* or *activities* visited in runtime are expected to be captured in SDR.	20	-1
	SdrTraceLevel	optional	yes	yes			It controls the level of SDR detail that is recorded by the `blocks` array for each application. Currently, the valid values are: • 100 — Debug level and up. Currently, there are no Debug messages. • 200 — Standard level and up. This setting will show all blocks that are entered during a call in the `blocks` array. • 300 — Important level and up. This setting filters out all blocks from the `blocks` array, except those containing data that will change from call to call (such as the Menu block and User Input block).	300	300
Audit	enableESAuditLogs	optional	yes	yes			Enable or Disable Audit logs captured in Elasticsearch	false	false
	enableFSAuditLogs	optional	yes	yes			Enable or Disable Audit logs captured in file system under logs directory or in standard output	true	true
	maxAppSizeCompare	optional	yes	yes			The maximum size of data object for which a diff will be captured in audit logs, value in bytes. I.e. The difference between old Designer object value and new value	1000000	1000000
	enableReadAuditLogs	optional	yes	yes			Control whether reading of Designer objects is captured in audit trails. If enabled any designer objects viewed in the UI will be recorded in audit logs	false	false
Authorization	disableRBAC	optional	yes	yes			Controls if Designer reads and enforces permissions associated with the logged-in user's roles	false	false
	rbacSection	optional	yes	yes			In a Role object, the name of the section within the Annex where the privileges are stored.	CfgGenesysAdministratorServer	CfgGenesysAdministratorServer
	disablePBAC	optional	yes	yes			Controls if Designer allows partitioning of Designer workspace and restricts a user's access to Designer objects in the user's partitions	false	false

Collaboration	locking	optional	yes				The type of locking used, for an editing session of applications, modules, or data tables. Valid values : "file", "redis", "none". • none - resources are not locked and can be edited simultaneously by multiple users which can result in one user overwriting another user's changes • file - uses files to keep track of locks and relies on shared storage (e.g. NFS) to make lock files available to each pod of Designer. Lock files are stored in the same location as the customer's Designer workspace • redis - uses Redis to for storing resource locks and is recommended for production environments	file	file
DAS	application Host	Mandatory	yes				The server name Designer uses to generate the URL to the application. ORS and MCP fetch application code and other resources from this URL	das.usw1.genhtcc.com	localhost
	application Port		yes				The corresponding port to be used with applicationHost	80	80
	deployURL		yes				This is normally not changed. It is the relative path to the workspace on DAS	/workspace	/workspace
Digital	rootsSRL	optional	yes	yes			If specified, this is used to filter which Root Categories to display when selecting Standard Responses.	A REGular EXpression (REGEX)	
	maxFlowEntryCount	optional	yes		yes	flowsettings	Specify how many times the same application can process a specific digital interaction.	20	20
External APIs	httpProxy	optional	yes	yes	yes	flowsettings	Specify the proxy used for external request and nexus API calls (if enable_proxy is true)	http://vpcproxy-000-int.geo.genprim.com:8080	
	redundantHttpProxy	optional	yes	yes	yes	flowsettings	Specify the backup proxy used for external request and nexus API calls (if enable_proxy is true), when httpProxy is down	http://vpcproxy-001-int.geo.genprim.com:8080	
Features	features		yes	yes			This is an object. See the features section for a list of support features.		{ "nexus": true, "enableBulkAudioImport: true }
GWS	usehtcc		yes				Set to "true" so Designer works with GWS. If set to "false" Designer defaults to a local mode and may be used temporarily if GWS is unavailable	true	false
	htccServer		yes				GWS Server	gws-usw1-int.genhtcc.com	gws-usw1-int.genhtcc.com
	htccport		yes				GWS Port	80	80
	ssoLoginUrl		yes				URL of GWS authentication UI. Designer redirects to this URL for authentication.	https://gws-usw1.genhtcc.com	https://gws-usw1.genhtcc.com
	maxConcurrentHTCCRequest	optional	yes				For batch operations to GWS, the max number of concurrent requests that Designer will send to GWS.	5	5
	batchOperationResultTTL	optional	yes				For batch operations to GWS, the time, in milliseconds, that Designer stores results of a batch operation on the server, before deleting it.	100000	100000

Help	docsMicroserviceURL	optional	yes				URL for Designer Documentation		https://docs.genesys.com/Documentation/PSAAS/Public/Administrator/Designer"
IVR	recordingType	optional	yes	yes			Specify the recording type to be used in Record block. Set as 'GIR'. If the option is missing or blank, "Full Call Recording" type will be used.	GIR	GIR
Logging	"logging": { "designer": { "level": "debug" }, "audit": { "level": "trace" }, "auditdebug": { "level": "debug" }, "cli": { "level": "debug" } }	optional	yes				Specify Designer log levels, each filed has valid values - trace, debug, info, warn, error, fatal designer - log level of Designer audit - log level of audit auditdebug - log level of audit debug, this will log detailed audit information cli- Log level for cli commands executed on Designer	"logging": { "designer": { "level": "debug" }, "audit": { "level": "trace" }, "auditdebug": { "level": "debug" }, "cli": { "level": "debug" } }	"logging": { "designer": { "level": "debug" }, "audit": { "level": "trace" }, "auditdebug": { "level": "debug" }, "cli": { "level": "debug" } }
Nexus	url	optional			yes	nexus	URL of Nexus that typically includes API version path e.g. https://nexus-server/nexus/api/v3		http://nex-dev.usw1.genhtcc.com
	password	optional			yes	nexus	nexus x-api-key created by Nexus deployment		dc4qeiro13nsof569dfn234smf
	enable_proxy	optional			yes	nexus	boolean value on whether httpProxy is used to reach Nexus		false
	profile	optional			yes	nexus	Enable Contact Identification via Nexus (e.g. to enable Last Called Agent routing)		
Process	port		yes				Designer process port in the container. Normally it should be left to the default value.	8888	3000
Provisioning	primarySwitch	optional	yes	yes			Specify the primary switch name if more than one switch is defined for the tenant. Designer fetches and works with route points from this switch.		us-west-1
Routing	ewtRefreshTimeout	optional			yes	flowsettings	Specify the interval (in seconds) at which to refresh the Estimated Waiting Time when routing an interaction.	5	1
redis	"redis": { "host": "", "port": "", "tlsEnabled": true, "lockTimeout": 120, "listTimeout": 1800 }	optional					Used by Designer for resource index caching and multi user collaboration locks on Designer resources. It is a separate object contains host - redis host name port - redis port tlsEnabled - TLS enabled or not lockTimeout - Timeout, in seconds, before a resource lock is released, for an editing session of applications, modules, or data tables. listTimeout - The cache expiry timeout(in seconds) of the application list and shared modules list. By default, it is 30 minutes, i.e any new application /modules created in the UI will be seen in the listing page after 30 mins, this can be reduced to smaller value. This is to improve page loading performance of Applications and Shared Modules page, more performance will be achieved with the higher value	"redis": { "host": "", "port": "", "tlsEnabled": true, "lockTimeout": 120, "listTimeout": 1800 }	"redis": { "host": "redis.server.genhtcc.com", "port": "6379", "tlsEnabled": true, "lockTimeout": 120, "listTimeout": 1800 }
Security	zipFileSizeLimitInMegaBytes	optional	yes	yes			Defines the maximum zipFile size limit(in megabytes) during bulk audio import	50	no default

	disableCSRF	optional	yes	yes			Disable CSRF attack protection http://cwe.mitre.org/data/definitions/352.html By default CSRF attack protection is enabled, it can be disabled by setting the flag as true	false	false
	disableSecureCookie	optional	yes				Disable the secure cookies header	false	false
Session	idleTimeout	optional	yes	yes			Idle timeout, in seconds, before a user session is terminated while editing applications, modules, or data tables.	840	840
	lockTimeout	optional	yes	yes			Timeout, in seconds, before a resource lock is released, for an editing session of applications, modules, or data tables.	120	120
	lockKeepalive	optional	yes	yes			Interval, in seconds, before the client sends a ping to the server, to refresh the lock for an editing session of applications, modules, or data tables.	15	15
Workflow	maxBuilds	optional	yes	yes			Specify the maximum number of Build allowed per application.	20	20
	enablePTE	optional			yes	flowsettings	boolean value on whether PTE objects are enabled at runtime	true	false

Features:

This features should be configured under 'features' object in flowsettings.json or tenantsettings.json. For example:

"features": {
"nexus": true,

..

}

Note : These features are configured only in flowsettings.json and tenantsettings.json, and not in DesignerEnv.

Category	Feature Setting Name	Mandatory	flowsettings.json	tenantsettings.json	Description	Default value
Audio	enableBulkAudioImport	optional	yes	yes	Enable/disable bulk audio import feature	false
	grammarValidation	optional	yes	yes	If this is enabled, Designer will validate invalid grammar files during grammar upload. If it is enabled, only valid grammar files(.grxml or Nuance compiled binary grammar) are allowed to upload	false
Nexus	nexus	optional	yes	yes	Enable/disable nexus feature. If `true` , Designer uses	false
Survey	survey	optional	yes	yes	Enable/disable survey feature	true
Milestone	enableImplicitModuleMilestones	optional	yes	yes	Enable to report each Shared Module call as an internal milestone. If disabled, Shared Module calls will not generate a milestone	false

4.3 Logging

Designer and DAS supports console output(stdout) logging, it recommended to configure console output logging to minimize the host IOPs and PVCs consumption by using log volumes. The console output logs can be extracted using log collectors like fluentbit/fluentd and Elasticsearch. Please make sure the below configs are set in the respective values.yaml overrides for the console logging.

1. Designer
    a. designerEnv.envs.DES_FILE_LOGGING_ENABLED = false
2. DAS
    a. dasEnv.envs.DAS_FILE_LOGGING_ENABLED = false
    b. dasEnv.envs.DAS_STDOUT_LOGGING_ENABLE = true

4.3.2 Log Level

Post deployment, Designer and DAS log levels can be modified as follows:

4.3.2.1 Designer

1. Configure the "logging" settings in flowsettings override (flowsettings.yaml) - Refer "4.2.4 Configuration reference table" for these option description.
2. Run the "6.8 Flowsettings.json update" steps for the changes to take effect.

4.3.2.2 DAS changes

1. Configure the "dasEnv.envs.DAS_LOG_LEVEL" in the helm das values.yaml- Refer "4.1.3 Designer Application Server (DAS) deployment settings" for the settings description.
2. Run the DAS upgrade to non production color as mentioned in the steps "6.5.2.2.3.3 Upgrade", same DAS version running in production can be used for upgrade.
3. Run Cutover as mentioned in the steps "6.5.2.2.3.4 Cutover".

5 Platform / Configuration Server and GWS settings

This section explains the required Configuration Server objects/settings for Designer.

5.1 Create roles for Designer

Designer uses roles and access groups to determine permissions associated with the logged-in user. To enable this, you must make these changes in Agent Setup.

Designer supports a number of bundled roles suitable for various levels of users.

- Designer Developer : most users will fall in this category. These users can create Designer applications, upload audio and create business controls. They have full access to Designer Analytics.
- Designer Business User : these users cannot create objects but they can manage them e.g. upload audio, change data tables and view analytics.
- Designer Analytics : provides access only to Analytics.
- Designer Admin : this type of user can setup partitions and manage partitions associated with users and Designer objects.
- Designer Operations :Users with this role have full access to all aspects of the Designer workspace. This includes the **Operations** menu (normally hidden), where they can perform advanced operational and maintenance tasks.

To create these roles, import .conf files included in the Designer Deployment Manifest package. They are located in packages/roles/ folder.

There are few more items to follow with user accounts that need to login to Designer:

- The user must have read permissions on its own Person object
- Users must be associated with one or more roles via access groups
- The onPremise user must have at least read access on the user, access group(s) and roles(s).
- The access groups should have read/write permissions to the Agent Setup folders, Scripts and Transactions.

5.2 Create DesignerEnv transactions list

Designer requires a transaction list for configuration purposes as described in other sections of this document. To set this up,

- Create a transaction list called "DesignerEnv".
- Import the file configuration/DesignerEnv.conf in the Designer Deployment Manifest package.
- Edit any values according to the description above in "Designer Settings".
- Save the list.
- Ensure Designer users have at least read access to the DesignerEnv transaction list.

5.3 Platform settings

These platform settings are required if the Designer application is used for voice calls.

Component	Config Key	Value	Description
SIP Switch -> Voip Services -> msml service	"userdata-map-format"	"sip-headers-encoded"	Option needs to set to pass JSON data as user data in SIPS.
SIP Switch -> Voip Services -> msml service	userdata-map-filter	*	To allow userdata passing to MSML service
SIPServer --> TServer	divert-on-ringing	false	RONA is handled by the platform.
	agent-no-answer-timeout	12	
	agent-no-answer-action	notready	
	agent-no-answeroverflow	""	no value, empty.
	after-routing-timeout	24	

	sip-treatments-continuous	true	
	msml-record-support	true	To allow routed calls recording via the Media Server
Switch object annex --> gts	ring-divert	1	
ORS --> orchestration	new-session-on-reroute	false	Required for SIPS Default Routing (Default Routing handling (Voice))
MCP	[vxmli] transfer.allowed	TRUE	Required for Transfer block (allows VXML Transfer in MCP)
MCP	[cpa] outbound.method	NATIVE	Required for Transfer block (allow CPA detection for Transfer)
UCS	[cview] enabled	TRUE	Enables Customer Context Services

5.4 GWS configuration settings

5.4.1 Create Contact Center

Create a contact center in GWS if it is not already created. Refer to the GWS documentation for more information on this.

5.4.2 Create GWS client

Create new GWS client credentials if they are not already created . Refer to the GWS documentation for more information on this.

6 Deployment

This section explains the deployment process for Designer and DAS.

Preparation

Before you deploy Designer And DAS using Helm Charts, complete the following steps:

1. Make sure the Helm client is installed.
2. Set up an Ingress Controller.
3. Set up an NFS server.
4. Create Persistent Volumes - a sample YAML is provided in the Designer manifest package.
5. Download the Designer and DAS docker images and push them to the local docker registry.
6. Download the Designer manifest package and extract it to the current working directory.
7. Configure designer and das values (designer-values.yaml and das-values.yaml), please make sure the mandatory settings are configured. If blue-green deployment process is used, Ingress settings are explained in blue-green deployment section.

6.1 Set up ingress

Following are requirements to setup ingress for Designer UI

- Cookies name - designer.session
- Header requirements - client IP & redirect, passthrough
- Session stickiness -enabled
- Whitelisting - optional
- TLS for ingress - optional (should be able to enable or disable TLS on the connection).

6.2 Set up Application Gateway (WAF) for Designer

Designer Ingress should be exposed to the internet using Application Gateway enabled with WAF. So, when WAF is enabled, the following are the exceptions in the WAF rules for Designer:

Designer sends a JSON payload with data, for exmaple, {profile . {} }. At times, this is detected as a OSFileAccessAttempt, which is a false positive detection. Dsable this rule if you encounter a similar issue in your WAF setup.

6.3 Storage

6.3.1 Designer storage

Designer requires storage to store application workspaces. Designer storage is a shared file storage that will be used by both the Designer and DAS services. This storage is critical, so make backups and snapshots at a regular interval, for instance, every day.

A Zone-Redundant Storage system is required to replicate data from the RWX volumes and must be shared across multiple pods:

Capacity	1 TiB
Tier	Premium
Baseline IO/s	1424
Burst IO/s	4000
Egress Rate	121.4 MiBytes / s
Ingress Rate	81.0 MiBytes / s

6.3.2 DAS storage

If the Designer workspace is stored in cloud storage like Azure Files, then the data must be synced to DAS pods using Designer-Sync service (as mentioned in 4.1.5) and in this case, DAS should use Statefulset deployment type.

So, in the DAS StatefulSet pods, each pod must be attached to a premium SSD disk to store the workspace.

Size	> 500GiB
Max IOPS (Max IOPS w/ bursting)	2,300 (3,500)
Max throughput (Max throughput w/ bursting)	150 MB/second (170 MB/second)

6.4 Setup Secrets

Secrets will be needed for the Designer service to connect to GWS and Redis (if you are using it).

GWS Secrets:

GWS provides client id and secrets to all the clients which can be connected. You can create secrets for the client-designer as mentioned in step 5.5.2.

Redis password:

If you are using Designer connected to Redis then you need to provide the Redis password to Designer to authenticate the connections.

6.4.1 Provision secrets for Designer

Refer the section designer.designerSecrets in values.yaml and configure secrets as follows:

```
designerSecrets:
  enabled: true
  secrets:
    DES_GWS_CLIENT_ID: "xxxx"
    DES_GWS_CLIENT_SECRET: "xxxx"
    DES_REDIS_PASSWORD: "xxxxx"
```

6.5 Deployment strategies

Designer supports the following deployment strategies:

- Rolling update (default)
- Blue Green Strategy (recommended)

DAS (Designer Application Server) supports the following deployment strategies:

- Rolling update (default)
- Blue Green (recommended)
- Canary (has to be used along with Blue-Green) (recommended in production)

6.6 Designer - Rolling upgrade

The rolling deployment is the standard default deployment to Kubernetes. It works slowly, one by one, replacing pods of the previous version of your application with pods of the new version without any cluster downtime. It is the default mechanism of upgrading for both Designer and DAS.

6.6.1 Initial deployment

To perform the initial deployment for rolling upgrade in Designer, use the below given helm command. The values.yaml can be created as per your needs.

```
helm upgrade --install designer -f designer-values.yaml designer-100.0.112+xxxx.tgz  --set designer.image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

designer.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.6.2 Upgrade

To perform an upgrade, the image version has to be upgraded in designer-values.yaml or can be set using the --set flag and the below command is used. Once the values file is updated, use the below helm command to perform the upgrade.

```
helm upgrade --install designer -f designer-values.yaml designer-100.0.112+xxxx.tgz  --set designer.image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

designer.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.6.3 Rollback

To perform a rollback, the image version in the designer- values.yaml can be downgraded or it can also be done using the --set flag and the below command is used. Once the values file is updated, the rollback can be performed by following the below command.

```
helm upgrade --install designer -f designer-values.yaml designer-100.0.112+xxxx.tgz  --set designer.image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

designer.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.7 DAS - Rolling upgrade

6.7.1 Initial deployment

To perform the initial deployment for rolling upgrade in DAS, use the below command . The values.yaml can be created as per your needs.

```
helm upgrade --install designer-das -f designer-das-values.yaml designer-das-100.0.112+xxxx.tgz  --set das.image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.7.2 Upgrade

To perform an upgrade, the image version has to be upgraded in designer-values.yaml or can be set using the --set flag and the below command is used. Once the values files is updated, use the below command to perform the upgrade.

```
helm upgrade --install designer-das -f designer-das-values.yaml designer-das-100.0.112+xxxx.tgz  --set das.
image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.7.3 Rollback

To perform a rollback, the image version in the designer- values.yaml can be downgraded or it can also be done using the --set flag and the below command is used. Once the values file is updated, use the below command to perform the rollback.

```
helm upgrade --install designer-das -f designer-das-values.yaml designer-das-100.0.112+xxxx.tgz  --set das.
image.tag=9.0.1xx.xx.xx
```

The values.yaml overrides passed as an argument to the above helm upgrade

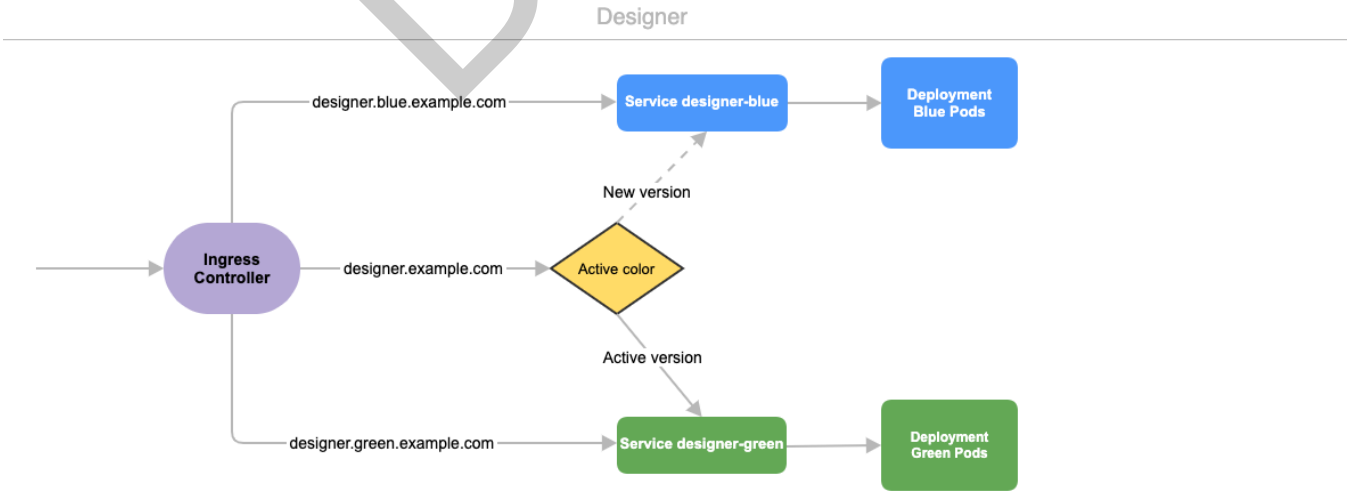
das.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

6.8 Designer - Blue-Green deployment

6.8.1 Deployment strategy

Blue-green deployment is a release management technique that reduces risk and minimizes downtime. It uses two production environments, known as Blue and Green or active/inactive, to provide reliable testing, continuous no-outage upgrades, and instant rollbacks. When a new release needs to be rolled out, an identical deployment of the application will be created using the Helm package and after the testing is completed, the traffic is moved to a newly created deployment which becomes the ACTIVE environment, and the old environment becomes INACTIVE. This way we can do fast rollback by just changing route if a new issue is found with live traffic. The old inactive deployment can be removed once the new active deployment becomes stable.

The service cutover is done by updating the ingress rules, Below diagram shows the high-level approach to how the traffic can be routed to Blue and Green deployments with Ingress rules.



6.8.2 Preparation

Before you deploy Designer using blue-green deployment, complete the preparation steps

1. Create 3 hostnames like below. The blue service hostname should contain the string "blue" like designer.blue.example.com or designer-blue.example.com, The green service hostname should contain the string "green" like designer.green.example.com or designer-green.example.com. The green/blue services can be accessed separately with these blue/green hostnames

- designer.example.com - This is for production host URL, this is used for external access
- designer.blue.example.com - This is for blue service testing
- designer.green.example.com - This is for green service testing

2. Configure the hostnames in the designer-values.yaml under ingress, annotations and paths can be modified based on the requirement

```
ingress:
  enabled: true
  annotations: {}
  paths: ["/"]
  hosts:
    - designer.example.com
    - designer.blue.example.com
    - designer.green.example.com
```

6.8.3 Deployment

6.8.3.1 Initial deployment

The resources - ingress and persistent volume claims (PVC) should be created initially before deploying the Designer service since these resources are shared between blue/green services and it is required to be created at the very beginning of the deployment, it will not be required for subsequent upgrades. The required values are passed using the SET command in the below steps, it can be done by changing the values.yaml as well

1. Create Persistent Volume Claims required for the Designer Service(Assuming the volume service name will be **designer-volume).**

```
helm upgrade --install designer-volume -f designer-values.yaml designer-9.0.xx.tgz --set designer.
deployment.strategy=blue-green-volume
```

The values.yaml overrides passed as an argument to the above helm upgrade
designer.deployment.strategy=blue-green-volume - This means, this helm install will create a persistent volume claim in the blue /green strategy.

2. Create ingress rules for the Designer service (Assuming the ingress service name will be **designer-ingress)**

```
helm upgrade --install designer-ingress -f designer-values.yaml designer-100.0.112+xxxx.tgz --set
designer.deployment.strategy=blue-green-ingress --set designer.deployment.color=green
```

The values.yaml overrides passed as an argument to the above helm upgrade

designer.deployment.strategy=blue-green-ingress - This means, this helm install will create ingress rules for Designer service

designer.deployment.color=green - This means, the current production(active) color is green

3. Deploy Designer service color selected in step 2. In this case, green is selected, assuming the service name will be designer-green

```
helm upgrade --install designer-green -f designer-values.yaml designer-100.0.112+xxxx.tgz --set designer.
deployment.strategy=blue-green --set designer.image.tag=9.0.1xx.xx.xx --set designer.deployment.
color=green
```

The values.yaml overrides passed as an argument to the above helm upgrade

designer.deployment.strategy=blue-green - This means, it will install Designer service in blue/green strategy

designer.image.tag=9.0.1xx.xx.xx - This is the Designer version to be installed , for example 9.0.116.07.10

designer.deployment.color=green - This means, it will install green color service

6.8.3.2 Upgrade

1. Identify the current production color by checking the Designer ingress rules (kubectl describe ingress designer-ingress). Green is the production color in the below example as the production host name points to the green service

kubectl describe ingress designer-ingress

```
Host                        Path  Backends
----                        ----  --------
designer.example.com        /   designer-green:http (10.244.0.23:8888)
designer.green.example.com  /   designer-green:http (10.244.0.23:8888)
designer.blue.example.com   /   designer-blue:http (10.244.0.45:8888)
```

2. Deploy Designer service into non-production color, for the above example, blue is the non-production color(assuming the service name will be designer-blue)

```
helm upgrade --install designer-blue -f designer-values.yaml designer-100.0.112+xxxx.tgz --set designer.
deployment.strategy=blue-green --set designer.image.tag=9.0.1xx.xx.xx --set designer.deployment.
color=blue
```

> The values.yaml overrides passed as an argument to the above helm upgrade
>
> designer.deployment.strategy=blue-green - This means, it will install Designer service in Blue green strategy
>
> designer.image.tag=9.0.1xx.xx.xx - This is the new Designer version to be installed , for example 9.0.116.08.12
>
> designer.deployment.color=blue - This means, it will install blue color release

3. The non-production color can be accessed with the non-production host name (for example - designer.blue.example.com), any testing can be done using this url

6.8.3.3 Cutover

Once the testing is completed on the non-production color, the traffic can be moved to the new version by updating the ingress rules

1. Update the Designer Ingress with the new deployment color by running the below command, this case, blue is the new deployment color(non-production color)

```
helm upgrade --install designer-ingress -f designer-values.yaml designer-100.0.112+xxxx.tgz --set
designer.deployment.strategy=blue-green-ingress --set designer.deployment.color=blue
```

> The values.yaml overrides passed as an argument to the above helm upgrade
>
> designer.deployment.strategy=blue-green-ingress - This means, this helm install will create ingress rules for Designer service
>
> designer.deployment.color=blue - This means, the current production(active) color is blue

2. Verify the ingress rule by running the command **kubectl describe ingress designer-ingress.** The production host name should point to the new color service

6.8.3.4 Rollback

In case the upgrade needs to be rolled back, the ingress rules can be modified to point to the old deployment pods (green, in this scenario) by performing a cutover again.

1. Perform cutover using the command

```
helm upgrade --install designer-ingress -f designer-values.yaml designer-100.0.112+xxxx.tgz --set
designer.deployment.strategy=blue-green-ingress --set designer.deployment.color=green
```

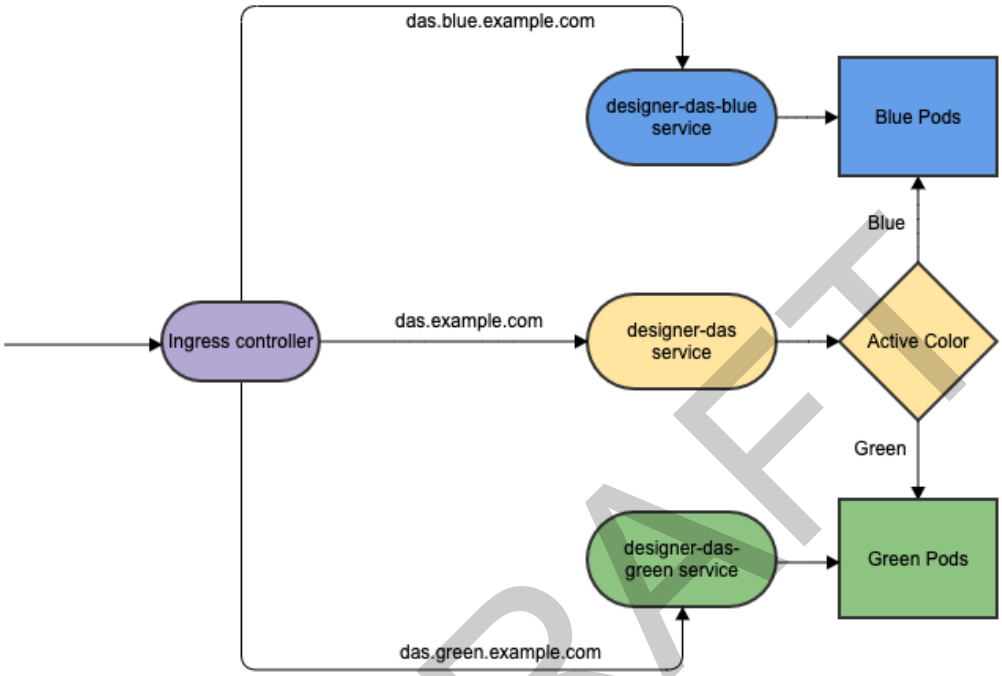
> The values.yaml overrides passed as an argument to the above helm upgrade
>
> designer.deployment.strategy=blue-green-ingress - This means, this helm install will create ingress rules for Designer service
>
> designer.deployment.color=green - This means, the current production(active) color is green

2. Verify the ingress rule by running the command **kubectl describe ingress designer-ingress.** The production host name should point to the green service.

6.9 DAS - Blue-Green deployment

6.9.1 Deployment strategy

Just like Designer, the Blue Green strategy can be adopted for DAS as well. The Blue-Green architecture used for DAS is given below. Here, the cutover mechanism is controlled by Service, the Kubernetes manifest responsible for exposing the pods. The ingress when enabled, will point to the appropriate service based on the URL.



6.9.2 Deployment

6.9.2.1 Initial deployment

The ingress should be created initially before deploying the DAS service since it is shared between blue/green services and it is required to be created at the very beginning of the deployment, it will not be required for subsequent upgrades. The required values are passed using SET command in the below steps, it can be done by changing the values.yaml as well

1. Deploy initial DAS pods and other resources by choosing an active color, in this case, green is chosen. Use the below command to create designer-das-green release

```
helm upgrade --install designer-das-green -f designer-das-values.yaml designer-das-100.0.106+xxxx.tgz --
set das.deployment.strategy=blue-green --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=green
```

> The values.yaml overrides passed as an argument to the above helm upgrade
>
> das.deployment.strategy=blue-green - This means, it will install the DAS service in the blue/green strategy.
>
> das.image.tag=9.0.1xx.xx.xx - This is the DAS version to be installed , for example, 9.0.111.04.4.
>
> das.deployment.color=green - This means, the current production(active) color is green.

2. Once the initial deployment is done, the pods have to be exposed to a service called 'designer-das'. Follow the below steps to create the designer-das service.

designer-das

```
helm upgrade --install designer-das designer-das-100.0.106+xxx.tgz -f designer-das-values.yaml --set das.
deployment.strategy=blue-green-service --set das.deployment.color=green
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.deployment.strategy=blue-green-service - This means, it will install the DAS service in the blue/green strategy.

designer-das service to expose the active color pods

das.deployment.color=green - This means, it designer-das service will point to green pods

Step 3 is required only when ingress is to be created to expose DAS outside the cluster. Please skip step 3 in case an ingress is not needed.

3. Create ingress rules for the DAS service (Assuming the ingress service name will be **das-ingress)**

helm

```
helm upgrade --install das-ingress designer-das-100.0.106+xxxx.tgz -f designer-das-values.yaml --set das.
deployment.strategy=blue-green-ingress
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.deployment.strategy=blue-green-ingress - This means, this helm install will create ingress rules for DAS service

6.9.2.2 Canary

Note

Canary is optional and is only used along with blue green. It is recommended in production. In case Canary is not needed, please skip this section.

6.9.2.2.1 Canary deployment

Canary pods are generally used to test new versions of images with live traffic. Steps to deploy canary include:

1. Identify the current production color by checking the designer-das service selector labels (kubectl describe service designer-das). Green is the production color in the below example as the selector label is color=green

kubectl describe service designer-das

```
Selector:               color=green
```

2. To deploy canary pods, the das.deployment.strategy value has to be set as Canary in the designer-das-values.yaml or can be set using the --set flag. The command used to deploy canary is given below. Another important to note is that, to make sure Canary pods receive live traffic, they have to be exposed to the designer-das service by setting das.deployment.color=<active_color>, which can be obtained from step 1.

```
helm upgrade --install designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.
deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=green
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.deployment.strategy=canary - This means, this helm install will create canary pods

das.deployment.color=green - This means, the current production(active) color is blue

3. Once canary pods are up and running, we have to make sure the designer-das service points to it by using **kubectl describe svc designer-das.**

```
Endpoints: 10.206.0.101:8081,10.206.0.162:8081,10.206.0.90:8081
```

The IP address present in the Endpoints should match the IP of the Canary. Canary pod's IP address can be gotten by using **kubectl describe pod <canary_pod_name>**

```
IP: 10.206.0.90
IPs:
  IP: 10.206.0.90
```

6.9.2.2.2 Cleaning - Canary

Once done with the Canary testing, the Canary pods need to be cleaned up. The steps are:

1. To Clean up Canary, the das.deployment.replicaCount should be made zero and the release is upgraded. It can be changed in designer-das-values.yaml or --set can be used.

```
helm upgrade --install designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.
deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=blue --set das.
deployment.replicaCount=0
```

6.9.2.3 Upgrade

1. Identify the current production color by checking the designer-das service selector labels (kubectl describe service designer-das). Green is the production color in the below example as the selector label is color=green

kubectl describe service designer-das
`Selector:  color=green`

2. Deploy DAS service into non-production color, for the above example, blue is the non-production color(assuming the service name will be das-blue)

```
helm upgrade --install designer-das-blue -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.
deployment.strategy=blue-green --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=blue
```

The values.yaml overrides passed as an argument to the above helm upgrade

das.deployment.strategy=blue-green - This means, it will install DAS service in blue-green strategy.

das.image.tag=9.0.1xx.xx.xx - This is the new DAS version to be installed , for example 9.0.111.05.5

das.deployment.color=blue - This means, it will install blue color service

3. The non-production color can be accessed with the non-production service name

6.9.2.4 Cutover

Once the testing is completed on the non-production color, the traffic can be moved to the new version by updating the designer-das- service

1. Update the designer-das service with the new deployment color by running the below command, this case, blue is the new deployment color(non-production color)

```
helm upgrade --install designer-das-service -f designer-das-values.yaml designer-das-100.0.106+xxxx.tgz
--set das.deployment.strategy=blue-green-service --set das.deployment.color=blue
```

> The values.yaml overrides passed as an argument to the above helm upgrade
>
> das.deployment.strategy=blue-green-service - This means, this helm install will create ingress rules for DAS service
>
> das.deployment.color=blue - This means, the current production(active) color is blue

2. Verify the service by running the command **kubectl describe service designer-das.** The type label should have the active color label i.e., color=blue

6.9.2.5 Rollback

In case the upgrade needs to be rolled back, cutover has to done again to make the service point to the old deployment (green) again. Use the below command to perform the cutover.

```
helm upgrade --install designer-das-service -f designer-das-values.yaml designer-das-100.0.106+xxxx.tgz --set
das.deployment.strategy=blue-green-service --set das.deployment.color=green
```

Verify the service by running the command **kubectl describe service designer-das.** The type label should have the active color label i.e., color=green

> The values.yaml overrides passed as an argument to the above helm upgrade
>
> das.deployment.strategy=blue-green-service - This means, this helm install will create ingress rules for DAS service
>
> das.deployment.color=green - This means, the current production(active) color is green

6.10 Validation and checks

Here are some common validations and checks that can be performed to know if the deployment was successful. They are:

1. Checking if the application pods are in running state by using using the command **kubectl get pods.**
2. Try to connect to the Designer or DAS URL as per the ingress rules from your browser, you should be able to access the Designer and DAS webpages.

6.11 Post-deployment/upgrade

6.11.1 Workspace upgrade for Designer

Workspace resources should be upgraded after cutover. This will upgrade the system resources in the Designer workspace

1. Login to one of the Designer pods with the command : *kubectl exec -it <pod_name > bash*
2. Run the migration command - this will create new directories/new files introduced in the new version

```
node ./bin/cli.js workspace-upgrade -m -t <contact_center_id>
```

3. Run workspace resource upgrade command - this will upgrade system resources like system service PHP files, internal audio files and callback resources

```
node ./bin/cli.js workspace-upgrade -t <contact_center_id>
```

contact_center_id - is the contact center id created in GWS for this tenant, also the workspace resources are located under contact center id folder (/workspaces/<ccid>/workspace)

Note - The above steps "Upgrade non production color", "cutover" and "workspace upgrade" will be used for further upgrades

6.11.2 Designer flowsettings update

Post-deployment, flowsettings.json can be modified via helm install, below are the steps

1. Extract the designer helm chart and find the flowsettings.yaml under the designer chart/config folder.
2. Modify the necessary settings (refer to 4.2.4 Configuration reference table)

3. Run the below helm upgrade command to the non-production color service. It can be done as a part of Designer upgrade by passing the flowsettings.yaml in the extra argument *--values* , in this case, new Designer version can be used for upgrade. If it is only a flowsettings.json update, the same Designer version will be used

```
helm upgrade --install designer-blue -f designer-values.yaml designer-9.0.xx.tgz --set deployment.
strategy=blue-green --set desImage.tag=9.0.1xx.xx.xx --set-string deployment.color=blue --values
flowsettings.yaml
```

4. Once the testing is completed in non production service, run cutover step as mentioned in the "6.5.2.1.3.3 cutover", now the production service will contain the setting changes. The non active color Designer will have to be updated with above settings change once the cutover is done.

6.12 Uninstall

To uninstall a release, use the below command

```
helm uninstall <release-name>
```

7 Enabling optional features

7.1 Enable Designer analytics and Audit trails

Post Designer deployment, the Designer analytics and audit trails features can be enabled by following the below steps. Deploy Elasticsearch if it's not done yet before starting the steps

7.1.1 Designer changes

1. Configure the following settings in flowsettings override (flowsettings.yaml) - Refer "4.2.4 Configuration reference table" for these option description
    a. "enableAnalytics": true
    b. "enableESAuditLogs": true
    c. esServer
    d. esPort
    e. esUrl
2. Configure the below setting in DesignerEnv transactions list
    a. ReportingURL in "reporting" section
3. Run the "6.8 Flowsettings.json update" steps mentioned above for the changes to take effect

7.1.2 DAS changes

1. Configure the following settings in the helm das values.yaml- Refer "4.1.3 Designer Application Server (DAS) deployment settings" for the settings description
    a. dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_ENABLED = true
    b. dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_HOST
    c. dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_PORT
2. Run the DAS upgrade(6.5.2.2.3.3 Upgrade), same DAS version running in production can be used for upgrade
3. Run cutover(6.5.2.2.3.4 Cutover)

8 Cleanup

8.1 Elasticsearch maintenance recommendations

To help you better manage your indexes and snapshots and to prevent too many indexes from creating an overflow of shards, it is recommended that you set up a scheduled execution of Elasticsearch Curator with the following two actions.

- Delete indexes older than the durations specified below, according to the index name and mask:
    - sdr-* (3 months)
    - audit-* (12 months)
- Make a snapshot of each index:
    - sdr-* (yesterday and older)
    - audit-*
    - kibana-int-*

9 Limitations

- Designer currently supports multi-tenancy provided by the tenant's Configuration Server. That is, each tenant should have a dedicated Configuration Server and Designer can be shared across the tenants.

Deploy Workspace Web Edition

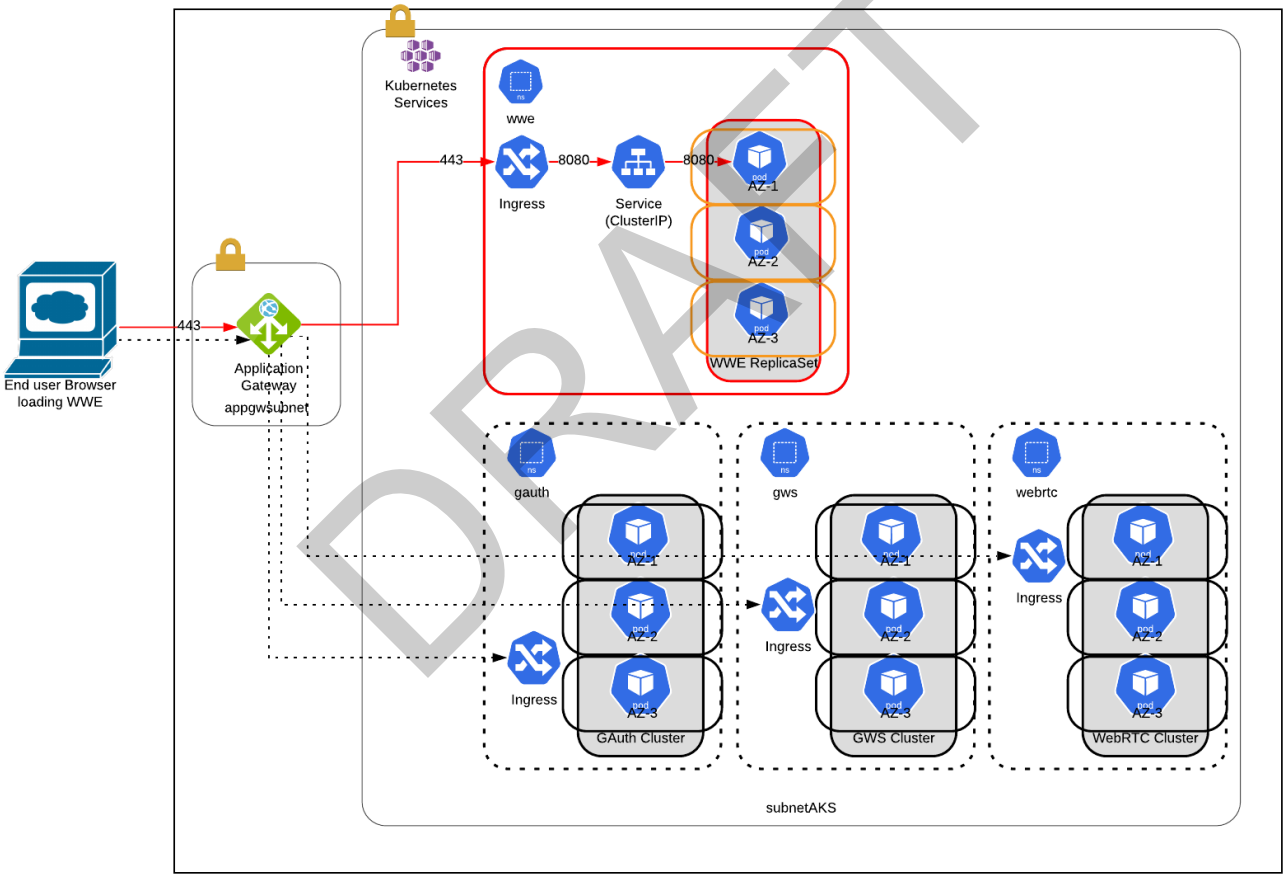
> **Disclaimer**
>
> Workspace Web Edition in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

The Workspace Web Edition Agent Desktop enables contact center agents and supervisors to communicate with customers and team members through phone calls, Outbound Campaigns, and Genesys Digital channels, such as chat, email, social media, SMS, WhatsApp, and workitems. Supervisors can use Workspace to monitor and coach their teams.

This topic describes how to **deploy** Workspace Web Edition in your environment.

The Workspace Web Edition (WWE) Service comprises:

- 1 Docker Container
- 1 Helm Chart



Prerequisites

Mandatory Dependencies

The following services must be deployed and running before deploying the WWE service:

- Genesys Authentication Service:
  - A redirect must be configured in Auth/Environment to allow an agent to login from the WWE URL. The redirect should be configured in the Auth onboarding script, according to the DNS assigned to the WWE service.
- GWS services:

    - The CORS rules for WWE URLs must be configured in GWS. This should be configured in the GWS onboarding script, according to the DNS assigned to the WWE service.
    - The GWS API URL should be specified at the WWE deployment time as part of the Helm values.
- TLM service:
    - The CORS rules for the domain where WWE is declared must be configured in Telemetry Service (TLM). For example: "genesysengage.com"

Optional Dependencies

Depending on the deployed architecture, the following services must be deployed and running before deploying the WWE service:

- WebRTC Service: To allow WebRTC in the browser
- Telemetry Service: To allow browser observability (metrics and logs)

Miscellaneous desktop-side optional dependencies

The following software must or might be deployed on agent workstations to allow agents to leverage the WWE service:

- **Mandatory**: A browser referenced in the supported browser list.
- **Optional**: Genesys Softphone: a SIP or WebRTC softphone to handle the voice channel of agents.

Secret configuration for pulling image

Use the following commands to create your Secret to access the jFrog registry:

```
oc create secret docker-registry <credential-name> --docker-server=<docker-repo> --docker-username=<username> --
docker-password=<API key from jfrog> --docker-email=<emailid>

oc secrets link default <credential-name> --for=pull
```

Security context configuration

Use the following command to enable the security context for the default service account:

```
oc adm policy add-scc-to-user genesys-restricted -z wwe
```

Prepare your environment

Check the cluster

Run the following command to get the version of the cluster:

```
oc get clusterversion
```

Create new project

Use the following command to create a new Workspace Web Edition (WWE) project:

```
oc new-project wwe
```

Download the WWE helm charts

Download the GWS helm charts from JFrog using your credentials.

Create the values.yaml file

From the following sample file, create the **values.yaml** file with appropriate overrides for a sample deployment.

Note: **ingress** should be enabled and set with an appropriate hostname and the value for **gwsUrl** must be set with the external GWS URL:

```
  context:
    envs:
      optimizedConfig: false
      gwsUrl: 'https://<gws-external-url>'
```

For example:

```
namespace: wwe
nameOverride: ""
fullnameOverride: ""

securityContext:
  runAsNonRoot: true
  runAsUser: 500
  runAsGroup: 500
  fsGroup: 500

podLabels: {}
podAnnotations: {}

wwe:
  image:
    registry: <docker-repo>
    repository: wwe
    tag: "9.0.000.82.10461"
    pullPolicy: IfNotPresent
    imagePullSecrets: []
  service:
    enabled: true
    type: ClusterIP
    port: 80
  ingress:
    enabled: true
    hosts:
      # Example
       - host: wwe.apps.vce-c0.eps.genesys.com
         paths:
           - path: '/'
             port: 443
    annotations: {}
    #   Example
    #   cert-manager.io/cluster-issuer: letsencrypt-prod-nginx
    #   nginx.ingress.kubernetes.io/ssl-redirect: "false"
    #   kubernetes.io/ingress.class: nginx01-internal
    #   nginx.ingress.kubernetes.io/proxy-body-size: "0"
    tls:
      # Example
       - secretName: ""
         hosts:
           - wwe.apps.vce-c0.eps.genesys.com
  serviceName: wwe
  deployment:
    type: Deployment
    replicaCount: 3
    minReplicas: 1
    maxReplicas: 10
    strategy: {}
  annotations: {}
  livenessProbe:
    httpGet:
      path: /index.html
      port: http
    initialDelaySeconds: 10
    periodSeconds: 5
    failureThreshold: 3
    timeoutSeconds: 5
  readinessProbe:
```

```
    httpGet:
      path: /index.html
      port: http
    initialDelaySeconds: 10
    periodSeconds: 5
    failureThreshold: 3
    timeoutSeconds: 5
  context:
    envs:
      optimizedConfig: false
      gwsUrl: 'https://<g
ws-external-url>'
  resources:
    requests:
      cpu: 500m
      memory: 2Gi
    limits:
      cpu: "1"
      memory: 6Gi
  priorityClassName:
  affinity: {}
  nodeSelector:
    genesysengage.com/nodepool:
  tolerations: []
  labels: {}
  autoscaling:
    enabled: true
    targetCPUUtilizationPercentage: 40
    targetMemoryUtilizationPercentage: 80
```

WWE service installation

Login to OpenShift cluster

Use the following command to log in to OpenShift cluster from the host where you will run deployment:

```
oc login --token <token> --server <url of api server>
```

Select your WWE Project

Use the following command to select the default WWE project that was created as a prerequisite:

```
oc project wwe
```

Render the templates

To ensure that resources are created correctly, you can render the templates for debugging purposes without installing them. Use the following command to render the templates:

```
helm template --debug -f values.yaml wwe-helm wwe-nginx/
```

Kubernetes descriptors are displayed. The values are generated from Helm templates, based on settings from values.yaml and values-test.yaml. Ensure that no errors are displayed. This configuration will be applied to your Kubernetes cluster.

Deploy WWE

Use the following command to deploy WWE:

```
helm install --debug --namespace wwe --create-namespace -f values.yaml wwe-helm wwe-nginx
```

This process takes several minutes. Wait until all objects are created and allocated, and the Kubernetes descriptors applied to the environment appear.

Check the deployment

Use the following command to check the installed Helm release:

```
helm list –all-namespaces
```

Use the following command to check the status of the WWE project:

```
oc status
```

Use the following command to check the WWE OpenShift objects created by Helm:

```
oc get all -n wwe
```

Expose the WWE service

Perform the following steps to make WWE accessible from outside the cluster, using the standard HTTP port:

1. Use the following command to expose the WWE service:

```
oc create route edge --service=wwe-helm-wwe-nginx --hostname=<hostname>
```

2. Use the following command to verify that the new route is created in the WWE project:

```
oc get route -n wwe
```

The result should show details similar to the following:

```
NAME   HOST/PORT                          PATH   SERVICES             PORT   TERMINATION   WILDCARD
wwe    wwe.apps.vce-c0.eps.genesys.com           wwe-helm-wwe-nginx   http   edge/Allow    None
```

where <host> is the host name generated by OpenShift.

3. Verify that you can access WWE at the following URL:

http://wwe.<host>

Configure Workspace Web Edition in Agent Setup

Use Agent Setup to configure Workspace Web Edition.

Uninstall WWE

To remove WWE, execute the following command:

```
helm uninstall wwe-helm -n wwe
```

Deploy Voice Services

Disclaimer

Voice Services in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

1. Introduction

Voice Microservices is an application cluster composed of several microservices that run together:

- Voice SIP Cluster Service
- Voice SIP Proxy Service
- Voice Orchestration Service
- Voice Agent State Service
- Voice Call State Service
- Voice Dial Plan Service
- Voice Registrar Service
- Voice Config Service
- Voice Front End Service
- Voice Redis (RQ) Service

Voice Microservices provide the following functionality:

- Handle incoming voice (SIP) interactions
- Route voice and digital (IXN) interactions
- Support outbound interactions
- Provide events stream for reporting
- Support agents across regions

This document captures steps to deploy Voice Services including dependent infrastructure services.

2. Deploy Prerequisites

This section describes the deployment of infrastructure services that are required for Voice Services:

- Consul
- Redis
- Kafka
- Elastic Search
- Prometheus

NOTE: In the OpenShift platform, the **oc** command can be used in place of the **kubectl** command, because **oc** was built on top of **kubectl**.

2.1 Override Values for Infra and Voice service Deployment

For both infra services as well as voice services we have captured the values that need to be configured in the override files. These values will override the default values provided by the Helm chart. These override values can be found in the following path <override_values_path>. Copy from this path and change the working directory to **azure_voice_install**.

2.2 Deployment of Consul

Enable the following Consul features that are required for Voice Services:

- connectinject - to deploy sidecar containers in voice pods.

- controller - to provide service intention functionality.
- openshift - to set OpenShift specific permissions.
- syncCatalog - to sync K8 services into Consul: set **toK8S: false** and **addK8SNamespaceSuffix: false** for syncing services only from K8 to Consul.
- AccessControlList - to enable ACL, set **manageSystemACLs: true**.
- storageclass - to set the storage class to a predefined storage class.
- TLS - to enable TLS, set **enabled: true** and follow the steps/commands described below to set up TLS.

Here is the file content for the Consul configuration:

```
# config.yaml
global:
  name: consul
  tls:
    enabled: true
    caCert:
      secretName: consul-ca-cert
      # The key of the Kubernetes secret.
      secretKey: tls.crt
    caKey:
      # The name of the Kubernetes secret.
      secretName: consul-ca-key
      # The key of the Kubernetes secret.
      secretKey: tls.key
  acls:
    manageSystemACLs: true
  openshift:
    enabled: true
connectInject:
  enabled: true
controller:
  enabled: true
syncCatalog:
  enabled: true
  toConsul: true
  toK8S: false
  addK8SNamespaceSuffix: false
```

Command	Purpose
consul tls ca create	To generate Consul certificates
kubectl create secret generic consul-ca-cert -n consul --from-file='tls.crt=./consul-agent-ca.pem'	To obtain the TLS certification and store it as a K8s secret
kubectl create secret generic consul-ca-key -n consul --from-file='tls.key=./consul-agent-ca-key.pem'	To obtain the TLS key and store it as a K8s secret

2.1.1 Commands to deploy Consul

```
kubectl create ns consul
helm repo add hashicorp https://helm.releases.hashicorp.com
helm repo update
helm install consul hashicorp/consul -f ./consul/consul-config.yaml -n consul
kubectl get pods -n consul
```

2.1.2 Add a Rule for Consul DNS forwarding

OpenShift sends DNS requests to the DNS server in the **openshift-dns** namespace. To forward Consul FQDN resolution to a Consul DNS server, add a forwarding rule to **configmap** of the default DNS operator. Save the Consul's DNS IP address using the following command:

```
kubectl get svc consul-dns -n <consul namespace> -o jsonpath={.spec.clusterIP} (Internal IP of consul-dns
service)
> oc edit dns.operator/default
Add the below specs:
spec:
servers:
- name: consul-dns
zones:
- consul
forwardPlugin:
upstreams:
- <Internal IP of consul-dns service>
```

2.2 Deployment of Redis Kafka and Elasticsearch

2.2.1 Add a Helm Repository

```
kubectl create namespace infra
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add elastic https://helm.elastic.co
helm repo add confluentinc https://confluentinc.github.io/cp-helm-charts/
helm repo update
```

2.2.2 Deploy the Helm Charts

```
#REDIS#
helm install infra-redis bitnami/redis-cluster --set usePassword=true --set cluster.nodes=6 --set persistence.
size=1G -n infra

#Elastic Search#
helm install elasticsearch elastic/elasticsearch -n infra --set replicas=1

#KAFKA#
helm install infra-kafka confluentinc/cp-helm-charts -f ./kafka/values.yaml -n infra
kubectl set env statefulset -n infra  infra-kafka-cp-kafka KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
```


3. Creation of the Voice Namespace

Before deploying Voice Services and their dependencies, create a namespace using the following command:

```
kubectl create ns voice
```

NOTE: In all Voice Services and their dependencies configuration files, the namespace is **voice**. If a customer wants a specific/custom namespace, create the namespace (using the above command) and change the namespace in respective files below as required.

4. Installation of Prometheus and Voice Dashboard and Alerts

Deploy dashboards and alert Helm charts for voice services as described in this section.

4.1 Prerequisite for deploying voice monitoring

Deploy and run Prometheus.

4.2 Deploy Dashboard and Alert dashboards

Deploy dashboards and alert rules using these Helm charts:

- **voice-dashboards** - This installs the dashboards that are created to monitor various Voice Services.

- **voice-alertrules** - This installs the alert rules that specify what type of alarm must be triggered based on the metrics.

```
helm repo add helm-staging https://<jfrog artifactory/helm location> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"
helm repo update

helm install voice-alertrules -n voice https://<jfrog artifactory/helm location>/voice-monitoring/voice-
alertrules-1.0.5.tgz --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm install voice-dashboards -n voice https://<jfrog artifactory/helm location>/voice-monitoring/voice-
dashboards-1.0.8.tgz --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

5. Deployment of Voice Services

5.1 Creation of the Consul bootstrap token

When Access Control List (ACL) is enabled in Consul, the Voice Services must have read-write privileges for some of the core components of the consul.

1. Get the Bootstrap Token. Use the following command to obtain the bootstrap token:

```
kubectl get secret consul-bootstrap-acl-token -n <consul namespace> -o go-template='{{.data.token |
base64decode}}
```

2. Log in to the Consul UI using the bootstrap token.

3. Create a new Token. In the same page, click Create policy.

4. Create a new policy (voice-policy) with the following privileges:

```
service_prefix "" {
  policy = "read"
  intentions = "read"
}
service_prefix "" {
  policy = "write"
  intentions = "write"
}
node_prefix "" {
  policy = "read"
}
node_prefix "" {
  policy = "write"
}
agent_prefix "" {
  policy = "read"
}
agent_prefix "" {
  policy = "write"
}
session_prefix "" {
  policy = "write"
}
session_prefix "" {
  policy = "read"
}
namespace_prefix "" {
 key_prefix "" {
  policy = "write"
 }
 session_prefix "" {
  policy = "write"
 }
}
key_prefix "" {
  policy = "read"
 }
key_prefix "" {
  policy = "write"
 }
```


5. Assign the newly created voice-policy to this token and create the token. The sample value of Token (a7529f8a-1146-e398-8bd7-367894c4b37b).

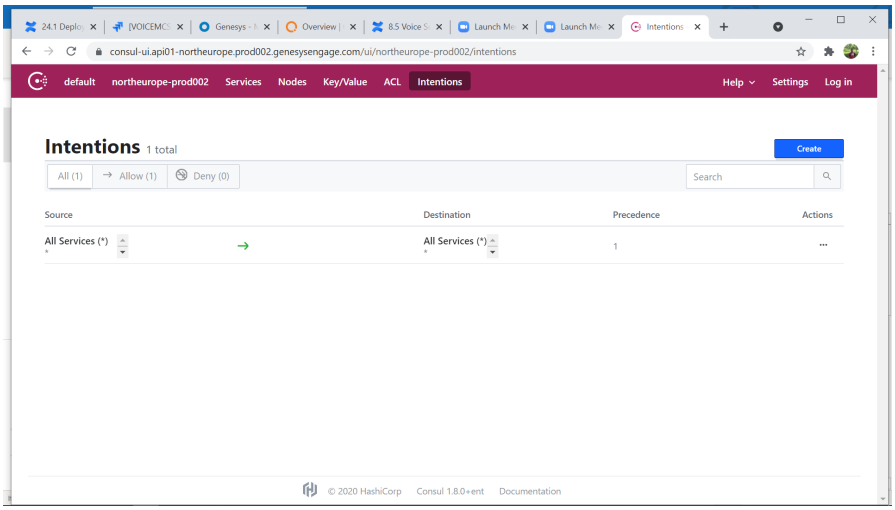
6. Click the token and copy the token, and create a K8 secret with this token, as follows:

```
kubectl create secret generic consul-voice-token -n voice --from-literal='consul-consul-voice-token=a7529f8a-
1146-e398-8bd7-367894c4b37b'
```

5.2 Creation of Intentions in Consul UI

Voice Services use a Consul service mesh to connect among services. Consul has provision to either allow or deny the connection between services. This is done using Intentions.

Log in to the Intentions tab using the bootstrap token. Then create a new intention to allow all source services to all destination services as shown below.



5.3 Security context configuration

In the OpenShift cluster, the access control is more restrictive and the UID and GID numbering is different. Because of this, the default Genesys User which is used in Genesys docker images cannot be installed directly. As described on Common prerequisites, you must create a Security Context Contraint (SCC) and add the required privileges for a Genesys user (genesys-restricted) and group (genesys-restricted-group).

Bind the SCC to service accounts used by Voice Services to grant permissions for any operations from the pod/container.

```
oc adm policy add-scc-to-user genesys-restricted -z voice-agent -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-callthread -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-config -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-registrar -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-sip -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-sipfe -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-sipproxy -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-rq -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-ors -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-dialplan -n voice
oc adm policy add-scc-to-user genesys-restricted -z voice-cprs-test-cprs -n voice
oc adm policy add-scc-to-user genesys-restricted -z tenant-<TENANT-UUID> -n voice
```

5.4 Register the Redis service in Consul

After the creation of the Redis cluster, the Redis FQDN must be registered in Consul. Use the **voice-redis-service** Helm chart for this purpose, as follows:

```
##get IPaddress of the master redis pod
export REDIS_MASTER_IP=$(kubectl get pods infra-redis-redis-cluster-0 -n infra -o jsonpath='{.status.podIP}')

helm repo add helm-staging https://<jfrog artifactory/helm location> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"
helm repo update

helm install -n voice -f ./voice_helm_values/voice_redis_values.yaml voice-redis-services  https://<jfrog
artifactory/helm location>/voice-redis-services/voice-redis-services-1.0.1.tgz --username "$JFROG_USER" --
password "$JFROG_PASSWORD" --set redis.agentnode.hostname=$REDIS_MASTER_IP,redis.callthreadnode.
hostname=$REDIS_MASTER_IP,redis.confignode.hostname=$REDIS_MASTER_IP,redis.orsnode.hostname=$REDIS_MASTER_IP,
redis.orsnodeStream.hostname=$REDIS_MASTER_IP,redis.registrarnode.hostname=$REDIS_MASTER_IP,redis.rqnode.
hostname=$REDIS_MASTER_IP,redis.sipnode.hostname=$REDIS_MASTER_IP,redis.tenantnode.hostname=$REDIS_MASTER_IP
```

5.5 Secrets for Voice Services

Create the following K8 secrets for other infrastructure services:

1. Kafka
2. docker-registry

3. Redis

5.5.1 Kafka Secrets

When Kafka is deployed without authentication, create Kafka secrets as shown below:

```
kubectl create secret generic -n voice kafka-secrets-token --from-literal=kafka-secrets={\"bootstrap\":<kafka-
bootstrap-url>}
for ex, kubectl create secret generic -n voice kafka-secrets-token --from-literal=kafka-secrets={\"bootstrap\":
\"infra-kafka-cp-kafka.infra.svc.cluster.local:9092\"}
```

When Kafka is deployed with authentication, create Kafka secrets as shown below:

```
kubectl create secret generic -n voice kafka-secrets-token --from-literal=kafka-secrets={\"bootstrap\":<kafka-
bootstrap-url>, \"username\": <USERNAME>, \"password\": <PASSWORD>}
for ex, kubectl create secret generic -n voice kafka-secrets-token --from-literal=kafka-secrets={\"bootstrap\":
\"infra-kafka-cp-kafka.infra.svc.cluster.local:9092\",\"username\":\"kafka-user\",\"password\":\"kafka-
password\"}
```

5.5.2 Redis Secrets

Use the following commands to create Redis secrets:

```
export REDIS_PASSWORD=$(kubectl get secret infra-redis-redis-cluster -n infra -o jsonpath="{.data.redis-
password}" | base64 --decode)
kubectl create secret generic -n voice redis-agent-token --from-literal=redis-agent-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-callthread-token --from-literal=redis-call-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-config-token --from-literal=redis-config-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-tenant-token --from-literal=redis-tenant-stream={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-registrar-token --from-literal=redis-registrar-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-sip-token --from-literal=redis-sip-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-ors-stream-token --from-literal=redis-ors-stream={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-ors-token --from-literal=redis-ors-state={\"password\":
\"$REDIS_PASSWORD\"}
kubectl create secret generic -n voice redis-rq-token --from-literal=redis-rq-state={\"password\":
\"$REDIS_PASSWORD\"}
```


5.5.3 Jfrog Secrets

Use the following commands to create Jfrog secrets:

```
kubectl create secret docker-registry <credential-name>  --docker-server=<docker-repo> --docker-
username="$JFROG_USER" --docker-password="$JFROG_PASSWORD" -n voice
```

5.6 Deployment of Voice Services

Before deploying Voice Services, we must create Persistent Volume Claims (PVC) that are required for Voice Services.

5.6.1 Persistent Volumes

The Voice SIP Cluster Service uses the persistent volume--a piece of storage--to store traditional SIP Server logs in OpenShift. For this, you do the following:

1. Persistent Volume for OCS
2. Persistent Volume for NFS

Persistent Volume in OCS Storage Type

A Storage Class might have been already created in the OCP (OpenShift Container Platform). This Storage Class is used for creating PVC and must be set in the override values of **sip_node_override_values.yaml**.

For the OpenShift cluster with OpenShift Container Storage (OCS) storage, configure the Storage Class to be used for creating the persistent volume. In the case of OCS storage, PV will be created automatically when PVC is claimed. For such clusters, the **volumeName** parameter in **sip_node_override_values.yaml** must be empty

```
# pvc will be created for logs
volumes:
  pvcLog:
    create: true
    claim: sip-log-pvc
    storageClass: voice
    volumeName:

  pvcJsonLog:
    create: true
    claim: sip-json-log-pvc
    storageClass: voice
    volumeName:
```

Persistent Volume in NFS Storage Type

For the OpenShift cluster with Network File System (NFS) storage, the persistent volume must be created manually:

```
kubectl apply -f ./voice_helm_values/sip_node_log_pv.yaml
```

The created persistent volume must be configured in **sip_node_override_values.yaml** as shown below:

```
# pvc will be created for logs
volumes:
  pvcLog:
    create: true
    claim: sip-log-pvc
    storageClass: voice
    volumeName: <pv name> (ex sip-log-pv)

  pvcJsonLog:
    create: true
    claim: sip-json-log-pvc
    storageClass: voice
    volumeName: <pv name> (ex sip-log-pv)
```


5.6.2 Configure the DNS Server for voice-sip

The Voice SIP Cluster Service requires that the DNS server is configured in **sip_node_override_values.yaml**.

In the OCP environment, the K8 DNS server name can be obtained using the following command:

```
oc get dns.operator/default -o jsonpath={.status.clusterIP}
```

In a non-OCP environment, follow the steps provided in Kubernetes documentation (https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/) and install the **dnsutils** pod. Using the **dnsutils** pods, get the **dnsserver** that is used in the environment.

Default value in the SIP Helm chart is "10.0.0.10", if the DNS server address is different, update it in **sip_node_override_values.yaml** as shown below:

```
# update dns server ipaddress
  context:
    envs:
      dnsServer: "10.202.0.10"
```

5.6.3 Voice Service Helm chart deployment

Deploy the Voice Services using the provided Helm charts.

```
helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/agent_override_values.yaml
voice-agent <helm-repo>/voice-agent-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/callthread_override_values.
yaml voice-callthread <helm-repo>/voice-callthread-<helmchart-version>.tgz --set version=<container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/config_override_values.
yaml voice-config <helm-repo>/voice-config-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/dialplan_override_values.
yaml voice-dialplan <helm-repo>/voice-dialplan-<helmchart-version>.tgz --set version=<container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/ors_node_override_values.
yaml voice-ors <helm-repo>/voice-ors-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/registrar_override_values.
yaml voice-registrar <helm-repo>/voice-registrar-<helmchart-version>.tgz --set version=<container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/rq_node_override_values.
yaml voice-rq <helm-repo>/voice-rq-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/sip_node_override_values.
yaml voice-sip <helm-repo>/voice-sip-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipfe_override_values.yaml
voice-sipfe <helm-repo>/voice-sipfe-<helmchart-version>.tgz --set version=<container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipproxy_override_values.
yaml voice-sipproxy <helm-repo>/voice-sipproxy-<helmchart-version>.tgz --set version=<container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"
```


The following table contains a list of the minimum recommended Helm chart versions that should be used for the OCP installation:

S.No	Service Name	Helm chart version
1	voice-config	voice-config-9.0.11.tgz
2	voice-dialplan	voice-dialplan-9.0.08.tgz
3	voice-registrar	voice-registrar-9.0.14.tgz
4	voice-agent	voice-agent-9.0.10.tgz
5	voice-callthread	voice-callthread-9.0.12.tgz
6	voice-sip	voice-sip-9.0.22.tgz
7	voice-sipfe	voice-sipfe-9.0.06.tgz
8	voice-sipproxy	voice-sipproxy-9.0.09.tgz
9	voice-rq	voice-rq-9.0.08.tgz
10	voice-ors	voice-ors-9.0.08.tgz

5.7 Upgrade of Voice Services

Because Voice Services are real-time services, we use canary-based deployment for their upgrades. The canary deployment is a technique of deploying one or more canary instances with the new version and verification of the new version to ensure it works as expected and also works with the previous version. Deploying only one or two canary instances should be sufficient to discover a faulty version and to minimize the risk of adding a new version into production.

The upgrade procedure consists of these major steps:

1. Canary deployment
2. Upgrade
3. Delete canary

5.7.1 Canary Deployment

For any new Voice Service version, the canary instance of it is deployed, and after the new version of the canary is approved, this version is rolled out to all instances of a Voice Service using the procedure covered in upgrade section.

For the canary deployment, some parameters in the **canary_override_values.yaml** file must be overridden. This file is passed to the Helm chart during the deployment of the canary instance.

```
# serviceaccount is created during initial deployment
serviceAccount:
create: false

deployment:
postfix: canary

# configmap is already created during initial deployment
context:
create: false

# this is needed for SIP canary only
loggingSidecar:
context:
create: false

# this is also needed for SIP canary only
volumes:
pvcLog:
create: false
pvcJsonLog:
create: false

# podmonitor is not needed for canary, but metric server enabling is needed
prometheus:
podMonitor:
enabled: false
metricServer:
enabled: true

# canary does not need HPA
hpa:
enabled: false
```


Commands to deploy a canary instance:

```
helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/agent_override_values.yaml
-f ./voice_helm_values/canary_override_values.yaml voice-agent-canary <helm-repo>/voice-agent-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/callthread_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-callthread-canary <helm-repo>/voice-callthread-
<helmchart-version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/config_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-config-canary <helm-repo>/voice-config-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/dialplan_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-dialplan-canary <helm-repo>/voice-dialplan-
<helmchart-version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/ors_node_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-ors-canary <helm-repo>/voice-ors-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/registrar_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-registrar-canary <helm-repo>/voice-registrar-
<helmchart-version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/rq_node_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-rq-canary <helm-repo>/voice-rq-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/sip_node_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-sip-canary <helm-repo>/voice-sip-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipfe_override_values.yaml
-f ./voice_helm_values/canary_override_values.yaml voice-sipfe-canary <helm-repo>/voice-sipfe-<helmchart-
version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipproxy_override_values.
yaml -f ./voice_helm_values/canary_override_values.yaml voice-sipproxy-canary <helm-repo>/voice-sipproxy-
<helmchart-version>.tgz --set version=<new-container-version> --username "$JFROG_USER" --password
"$JFROG_PASSWORD"
```


5.7.2 Service Upgrade

When the canary deployment of a Voice Service is ready for an upgrade, use the following commands to upgrade the current version of a Voice Service to the desired version:

```
helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/agent_override_values.yaml
voice-agent <helm-repo>/voice-agent-<helmchart-version>.tgz --set version=<new-container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/callthread_override_values.
yaml voice-callthread <helm-repo>/voice-callthread-<helmchart-version>.tgz --set version=<new-container-
version> --username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/config_override_values.
yaml voice-config <helm-repo>/voice-config-<helmchart-version>.tgz --set version=<new-container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/dialplan_override_values.
yaml voice-dialplan <helm-repo>/voice-dialplan-<helmchart-version>.tgz --set version=<new-container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/ors_node_override_values.
yaml voice-ors <helm-repo>/voice-ors-<helmchart-version>.tgz --set version=<new-container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/registrar_override_values.
yaml voice-registrar <helm-repo>/voice-registrar-<helmchart-version>.tgz --set version=<new-container-version>
--username "$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/rq_node_override_values.
yaml voice-rq <helm-repo>/voice-rq-<helmchart-version>.tgz --set version=<new-container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values/sip_node_override_values.
yaml voice-sip <helm-repo>/voice-sip-<helmchart-version>.tgz --set version=<new-container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipfe_override_values.yaml
voice-sipfe <helm-repo>/voice-sipfe-<helmchart-version>.tgz --set version=<new-container-version> --username
"$JFROG_USER" --password "$JFROG_PASSWORD"

helm upgrade --install --force --wait --timeout 300s -n voice -f ./voice_helm_values/sipproxy_override_values.
yaml voice-sipproxy <helm-repo>/voice-sipproxy-<helmchart-version>.tgz --set version=<new-container-version> --
username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

5.7.3 Delete the canary instance

If the upgrade of a Voice Service is successful, delete the canary instance of the service by using the following commands:

```
helm delete voice-agent-canary -n voice
helm delete voice-callthread-canary -n voice
helm delete voice-config-canary -n voice
helm delete voice-dialplan-canary -n voice
helm delete voice-ors-canary -n voice
helm delete voice-registrar-canary -n voice
helm delete voice-sip-canary -n voice
helm delete voice-sipfe-canary -n voice
helm delete voice-sipproxy-canary -n voice
```

5.7.4 Upgrade of RQ Node Service

The upgrade procedure of the RQ node service differs from other Voice Services and consists of the following steps:

1. **Change Strategy to OnDelete:**
   Set the strategy to **OnDelete** in **rq_node_override_values.yam**. Note that when a fresh RQ node service is deployed, the strategy is set to **RollingUpdate** in **rq_node_override_values.yaml** by default.

   Example:

```
deployment:
  deploymentType: statefulset
  strategy: OnDelete
```

2. **Upgrade voice-rq Helm to a new version:**

   Upgrade the **voice-rq** Helm to the newer version using the following command:

```
helm upgrade --install --force --wait --timeout 200s -n voice -f ./voice_helm_values
/rq_node_override_values.yaml voice-rq https://<jfrog artifactory/helm location>/voice-rq/voice-rq-
9.0.07.tgz --set version=9.0.6 --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

3. **Delete voice-rq-0 pod:**

   Delete the **voice-rq-0** pod, and then the **voice-rq-0** pod will be upgraded to a new version. Note that only when a pod is deleted, the upgraded Helm version will be considered to new pods. And this canary pod can be verified to ensure it works with other RQ nodes.
4. **Downgrade Helm to a previous version**:

   If other RQ node pods are deleted, they would also get upgraded to a newer version. To avoid such random upgrade of RQ nodes, downgrade Helm version to a previous version. And **voice-rq-0** will have a new version available for testing.

5. **Deploy Helm to a new version**:

   If a canary pod (**voice-rq-0**) works correctly with other pods and in the environment, upgrade the **voice-rq** Helm to the newer version (same as step 2). When the upgrade is successful, delete all RQ pods, so the newer RQ node pods will have the upgraded new version.

6. Deployment of Voice Tenant Service

Refer to Deploy Voice Tenant Service for details for the Voice Tenant Service deployment.

7. Testing of Voice Services

After the Voice Services and at least one instance of a Tenant Service are deployed for testing purposes, the smoke test can be run to validate the Voice Services.

1. Bind the SCC to service account **voice-cprs-test-cprs** used by a voice test service to grant permissions for any operations from the pod /container. This operation will be executed as a part of step 4.2.
2. Install the cprs Helm chart as follows:

```
helm upgrade --install --force --wait -n voice -f ./cprs/cprs_helmvalues.yaml voice-cprs-test https://<jfrog
artifactory/helm location>/voice-test/voice-test-9.0.05.tgz --set version=latest --username "$JFROG_USER" --
password "$JFROG_PASSWORD"
```

3. Start the cprs which will start the smoke test.

```
helm test voice-cprs-test -n voice
```

4. This smoke test does the following:

- SIP Registration
- Tlib Registration
- Agent Login
- Inbound call being routed to an agent

Deploy Voice Tenant Service

> **Disclaimer**
>
> Voice Tenant Service in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

1. Introduction

A Voice Tenant Service is a core service of the Genesys Engage cloud platform that serves as an application layer between front-end Genesys Engage cloud solutions and shared backend core services in a region.

The Voice Tenant Service instances are dedicated to a tenant of Genesys Engage cloud platform and provide these main functions: provisioning of tenant resources, such as agents and DNs; routing of interactions within a tenant; execution of outbound campaigns for a tenant; providing call control functionality; participation in authentication workflow for tenant's agents.

This document describes use-cases and deployment steps for a Voice Tenant Service.

2. Prerequisites

- **Voice Platform 9.0** and all its external dependencies must be deployed before proceeding with the Tenant Service deployment. See Steps 1-5 in Deploy Voice Services for details.
- **PostgreSQL 10** database management system must be deployed and database shall be allocated either as a master or replica. See 2.1 PostgreSQL DB for details of the sample deployment of the standalone DBMS.
- **GWS Authentication Service 9.0 -** recommended before proceeding with tenant deployment if use of Agent Setup, WWE is expected after tenant is deployed.

See Dependencies for details about interconnections between tenant and its dependencies.

2.1 PostgreSQL DB

For a Tenant Service, the PostgreSQL database is required and credentials of the database can be obtained in two ways: either create its own PostgreSQL DB for a Tenant Service or use the shared PostgreSQL DB (which is recommended in the OpenShift platform).

Deploy a PostgreSQL DB using the below commands:

```
helm repo add bitnami https://charts.bitnami.com/bitnami

helm install -n voice t100postgres -f ./tenant_helm_values/postgres_values.yaml bitnami/postgresql
```

3. Deployment Scenarios

Several deployment scenarios are supported, including a single region, redundant and multi-region deployments, and multi-tenant deployment.

3.1 Single region/location/cluster

Tenant resources are deployed in a single Azure Kubernetes Service (AKS) cluster within the same or separate namespace (project) with the voice platform. If shared resources are being deployed across all tenants, they must also be added to the same target namespace.

Before proceeding with the Tenant(s) deployment at a location, the following features are required:

- POD Monitor for all tenant pods
- FluentBit logging framework

The deployment must include the **tenant-monitor** module execution at a location, as described in tenant-monitor.

3.1.1 Single tenant

Basic deployment

Single-node deployment requires a single override file and one "tenant" module to deploy, with reference implementation described at Single service at one location.

Mandatory parameters for basic installation are:

- tenant uuid (v4)
- tenant nickname (become a helm release name)
- all backend parameters (along with all secrets that may be required based on these parameters)

To increase number of nodes, the **node count** parameter can be adjusted. See Scalability and redundancy parameters.

Upgrade

The upgrade can be performed by re-running deployment with the same mandatory parameters and adjusted version of tenant image(s) and Helm charts. The upgrade will be performed automatically, one node at a time (if **node count** is > 1). The update of the tenant configuration may happen automatically upon upgrade of the master node. It is recommended to perform a backup of the backend database for a Tenant Service before upgrade.

3.1.2 Multiple tenants at one location

Basic deployment

Additional tenants can be deployed at the same location with following guidelines:

- Each Tenant Service has to have unique tenant uuid, shortid, and nickname
- Each Tenant Service is deployed/upgraded and adjusted independently

3.2 Multiple regions/locations/clusters

3.2.1 Basic deployment

In multi-regional/multi-location deployments, one region/location is considered "master" (from a tenant perspective) and includes the database backend with write capabilities. Other regions/locations have replicas of the database backend in read-only mode. A Tenant Service at each location may be deployed to have one of its nodes running as master (write access to provisioning data via config API) or have all its nodes running only as replicas (read access to configuration).

Multi-regional deployments have to be done according to these steps (with pre-requisites already satisfied at each region/location):

- If required, deploy the **tenant-monitor** module at a location planned as a master tenant node.
- Complete basic deployment of a Tenant Service in the master region, including specification of DR parameters for master, as per Scalability and redundancy parameters.
- Complete deployment of the database backend with replica of master database at location(s) where replica tenant nodes are expected to run, including provisioning of access keys/secrets to access local replica.
- If required, deploy the **tenant-monitor** module at location(s) , where replicas are expected to run.
- Complete basic tenant deployment for additional region(s) and specify DR parameters for a master region (see Scalability and redundancy parameters).

The same customization scenarios discussed for tenant nodes can be applied for each location independently.

3.2.2 Upgrade

Can be performed by re-running deployment with the same mandatory parameters and adjusted version of tenant image(s) and helm charts for every location. Update of tenant configuration may happen automatically upon upgrade of master node in master region. It is recommended to perform back up of the backend database for a Tenant Service before upgrading the master region.

4. Dependencies

This section describes dependencies between service components, service dependencies on the rest of the platform and external (third party) dependencies, with endpoint information.

4.1 Service components endpoint dependencies

- Voice Front End Service

- Voice Redis (RQ) Service
- Voice Config Service

4.2 Platform Endpoint dependencies

- GWS environment API
- Interaction service core
- Interaction service vq

4.3 Third-party endpoint dependencies

- Kafka
- Consul
- Redis
- PostgreSQL

5. Deployment Parameters

This section describes purpose and use cases for configurable parameters of a Tenant Service deployment. It is NOT intended to provide actual values or names of override option of helm charts (later have to be extracted from **values.yaml**) file of each helm chart.

5.1 Versioning

Group	name	purpose
version	tenant image versions	target image to install, must use same version for all init containers
version	roles and permissions version	target version of roles and permissions to apply
location	image location	target registry to pull images from

5.2 Identification

Group	name	purpose	comments
Tenant	uuid	unique identifier of all instances of tenant service	all nodes deployed to handle end customer environment use that uuid that is also registered. last 4 positions of this uuid are used as short tenant id, when applicable.
Tenant	name	nickname of tenant	human readable name , default to "t<short_id>"

5.3 Backend parameters

Group	name	purpose	comment
postgres	database host	reference to backend dbms to persist service into	either direct value or file path that can point out to a mapped volume with file content that can be used as dbms name
postgres	database user	reference to backend database to persist service into	either direct value or file path that can point out to a mapped volume with file content that can be used as database username
postgres	database name	reference to backend database to persist service into	either direct value or file path that can point out to a mapped volume with file content that can be used as database username
postgres	database password parameters	either direct value or reference to a secret name and key that hold a value, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how password is being extracted by service. direct value is supported to testing purposes; secret name can be specified if password from external secret shall be mounted as a volume and password exposed via file; secret key can be specified if password shall be mapped to environment variable
postgres	k8s secret usage flags	indication of k8s secret being used to keep password and whenever secret shall be mounted to env variable or expected to be mapped as a volume	boolean values that forces use of secret (instead of direct database password).
postgres	ssl usage	specify secure connection preferences	allow or require tls
consul	k8s secret usage flags	indication of k8s secret being used to keep token and whenever secret shall be mounted to env variable or expected to be mapped as a volume	boolean values that force use of secret (instead of direct consul token value).
consul	consul token parameters	either direct value or reference to a secret name and key that hold a value, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how token is being extracted by service. direct value is supported to testing purposes; secret name can be specified if token from external secret shall be mounted as a volume and token exposed via file; secret key can be specified if password shall be mapped to environment variable

redis	k8s secret usage flags	indication of k8s secrets being used to keep connection strings and whenever secrets shall be mounted to env variables or expected to be mapped as a volumes	boolean values that forces use of secret (instead of direct connection strings). applicable for all type of redis connections supported by tenant
redis	redis connection string for tenant stream	either direct value or reference to a secret name and key that hold a value, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how string is being extracted by service. direct value is supported for testing purposes; secret name can be specified if value from external secret shall be mounted as a volume and connection parameters exposed via file; secret key can be specified if string shall be mapped to environment variable
redis	redis connection string for config cache	either direct value or reference to a secret name and key that hold a value, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how string is being extracted by service. direct value is supported for testing purposes; secret name can be specified if value from external secret shall be mounted as a volume and connection parameters exposed via file; secret key can be specified if string shall be mapped to environment variable
redis	cluster flag	type of redis backend to connect	indicate whenever backend for redis is cluster or standalone server (same is used for all types of redis endpoints)
kafka	k8s secret usage flags	indication of k8s secrets being used to keep connection strings and whenever secrets shall be mounted to env variables or expected to be mapped as volumes	boolean values that force use of secret (instead of direct connection strings).
kafka	kafka connection string	either direct value or reference to a secret name and key that hold a value, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how string is being extracted by service. direct value is supported for testing purposes; secret name can be specified if value from external secret shall be mounted as a volume and connection parameters exposed via file; secret key can be specified if string shall be mapped to environment variable

5.4 K8s parameters

Group	name	purpose	comments
Auth and Autorization	Service Account	Specify non-default service account that shall be associated with all PODs of a tenant service deployment	PODs will be assigned default account if not specified. If required, a separate account will be created during tenant deployment and set to use by all PODs. This could be required if consul registration of tenant service relies on tenant's POD having k8s accounts named after tenant service being registered. Service account is being created with name matching service name of as tenant (as visible in consul)
Auth and Autorization	Pod identity	specify optional annotation to associate with POD in order to access K8s resources	PODs may be assigned adodbidentity if needed
Scheduling	pod node selector	specify optional node pool selector for tenant PODs	PODs can be assigned to a specific pool, if needed. There is no default K8s pool selections
Scheduling	pod toleration	specify optional toleration for tenant PODs	PODs can be set to tolerate specific taints, There is no default tolerations
Scheduling	affinity	enable affinity of tenant PODs to provide high availability	PODs of same tenant service may be forced to schedule in different locations (if supported by underlying infrastructure) using failure-domain.beta.kubernetes.io/zone annotation when this is enabled. there is no affinity by default
Scheduling	priority class	specify optional priority class for tenant PODs	PODS may be assigned specific priority class, not set by default

5.5 Logging parameters

Group	name	purpose	comments
logging frameworks	fluentbit	specify usage of fluentbit for tenant console logging	tenant PODs won't use fluentbit logging solution by default, tenant-specific fluentbit configuration can be enabled when deploying tenant-monitor resources, and use of fluentbit sidecar can be enabled on tenant PODs afterward.

logging frameworks	fluentbit config	specify config map with fluentbit sidecar configuration that should be mounted as a volume	if fluent is enabled in both tenant-monitor and tenant charts, a volume referencing config map shall be specified, as below (if custom config map is created, map name can be set differently)
			`- name: tenants-fluent-bit-config`
			`configMap:`
			`name: tenants-fluent-bit-config`

logging frameworks	fluentbit local storage	specify volume and volume mount where fluentbit logs will accumulate	if fluent bit is enabled, below volume : - name: fluent-logs emptyDir: {} and volume mount: - name: fluent-logs mountPath: "/opt/genesys/logs/JSON" shall be added
file logging	logging persistent volume	specify usage of persistent volumes for logging by tenant component that produce log files	tenant PODs won't use persistent volume claims by default and no storage classes are created. Persistent volume claims can be added to tenant PODs when this is enabled
file logging	logging persistent volume storage class	if use of persistent volume for logging has been enabled, storage class can be specified for these volumes as needed	if tenant PODS are set to use persistent volume claims, storage class can be specified explicitly for volume claims. storage class is not specified in PVC by default
file logging	logging storage class	if logging to persistent volume is enabled and storage class has been specified explicitly then creation of storage class can also be enabled as needed	tenant-specific storage class for persistent volumes won't get created by default, can be enabled when deploying tenant-monitor resources and later utilized by tenant PODs using same storage class name. storage class name shall be specified for this to work
file logging	logging empty directory	default logging mount path pointing to a K8s empty directory volume	part of default mounts provided for tenant service

5.6 Monitoring and observability

Group	name	purpose	comments
prometheus	Prometheus pod monitor	specify if pod monitor to scrape prom endpoints of tenant PODs should be created	tenant-monitor won't create pod monitor for prometheus by default, prometheus specific monitor can be created as part of infrastructure deployment when this is enabled
prometheus	Pod level annotations	alternative to enabling pod monitor	

5.7 Integration

Group	name	purpose	comments
GWS	GWS endpoint	specify dedicated endpoint to register tenant service upon initial deployment or upgrade	when provided, define contact address to reach master GWS environment service that should be used to register tenant resources to make them available for GWS-based applications. by default, endpoint isn't specified and localhost connection attempt will be made by tenant container, GWS endpoint shall be reachable via service mesh upstream
GWS	GWS endpoint tls mode	enable https (secure) connection mode to contact GWS endpoint	when endpoint is provided, use of this parameter forces secure connection when accessing GWS on that address. NOTE: not available in initial release, only http connections are supported
GWS	k8s secret usage flags	indication of k8s secrets being used to keep client id and token and whenever secrets shall be mounted to env variables or expected to be mapped as volumes	boolean values that force use of secret (instead of direct client id and token).
GWS	client id and token	either direct values or references to secret names and keys that hold values, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how client id and token is being extracted by service. direct value is supported for testing purposes; secret name can be specified if value from external secret shall be mounted as a volume and connection parameters exposed via file; secret key can be specified if string shall be mapped to environment variables
Service Mesh	upstream services	override upstream service references used by tenant instance to locate intra-service and platform dependencies via consul	tenant POD has default service references that tenant service has to access via consul service mesh, this parameter allow to specify alternative values to make consul service names to tenant POD local ports allocated for service mesh. For more details see Dependencies.
voice	sip domain	provide sip communication domain for voice /sbc integration	customize SIP URI used to deliver between tenant and core voice platform and / or SBC
Tenant	service user	provide name of service user account	this account is being used to manage all parts of tenant service provisioning and is created at bootstrap
Tenant	k8s secret usage flags	indicate usage of k8s secrets to keep tenant service account password	

Tenant	service user password	either direct value or references to a secret name and key that hold values, depends on relevant k8s secret flag and k8s secret env map flag	these parameters control how password for internal admin account is being accessed service. direct value is supported for testing purposes; secret name can be specified if password from external secret shall be mounted as a volume and password exposed as a file; secret key can be specified if password shall be mapped to environment variables

5.8 Scalability and redundancy parameters

Group	name	purpose	comment
Tenant	node count	manage number of nodes deployed per service instance	by default service instance is being deployed with single node. if local high availability is required, additional nodes can be added to as service by value of this parameter and re-running deployment
Tenant	master location	manage location where master (writable) tenant node can be found for multi-regional deployments	by default tenant service is deployed as master, and expects to have writable local database backend accessible at its location, this parameter is required when deploying additional regions of the same tenant at other locations and its content should match with the name of consul datacenter where master tenant nodes are deployed. it shall be set the same across all locations

5.9 Extended parameters

Group	name	purpose	comment
postgres	main volume mounts for backend secrets	specify volumes and volume mounts for main tenant container that reference secrets to bound	usage of volumes and volume mounts depends on backend parameters selected as part of backend provisioning. volumes and mounts need to be specified if backend secrets are provisioned to come from secrets mapped as file systems. Note: Tenant has only one set of volume and volumeMounts entries in parameters overrides, all volumes for all purposes must be concatenated into one entry
postgres	init volume mounts for backend secrets	specify volumes and volume mounts for init containers (which are started as part of tenant onboarding and require secrets to operate)	Similar to volumes/mounts of main tenant container. Note: Tenant has only one set of initVolumes and initVolumeMounts entries in parameters overrides, all volumes for all purposes must be concatenated into one entry.
consul	main volume mounts for consul token	specify volume mounts for main container to bound for consul access creds	usage of volumes and volume mounts depends on backend parameters selected as part of consul provisioning. Volumes and mounts need to be specified if consul token is being provisioned from secret mapped as file systems. Note: Tenant has only one set of volume and volumeMounts entries in parameters overrides, all volumes for all purposes must be concatenated into one entry
consul	init volume mounts for consul token	specify volume mounts for main container to bound for consul access creds	Similar to volumes/mounts of main tenant container. Tenant has only one set of initVolumes and initVolumeMounts entries in parameters overrides, all volumes for all purposes must be concatenated into one entry.
redis			only main container needs to bound
kafka			only main container needs to bound

6. Containers and Charts Deliverables

6.1 Container Deliverables

container name	delivered via
core tenant service container	internal versions for testing: **<docker-repo>/genesys/katana/tenant_node:9.0.X**
database initialization and upgrade container	internal versions for testing: **<docker-repo>/genesys/katana/tenant_pgdb_init:9.0.X**
role and privileges initialization and upgrade container	internal versions for testing: **<docker-repo>/genesys/bec/tools/cspostdeplrs:8.5.00X.XX**
solution specific: pulse provisioning container	internal versions for testing: **<docker-repo>/genesys/katana/tenant_pulse_init:9.0.X**

6.2 Helm Chart Deliverables

chart name	delivered via
tenant deployment	internal versions for testing: **<docker-repo>/tenant-9.0.X.tgz**
tenant infrastructure	internal versions for testing: **<docker-repo>/tenant-monitor-9.0.X.tgz**

7. Deployment Steps

This section provides reference commands with key parameters that are required to complete each deployment step.

7.1 Service-specific infrastructure definitions

7.1.2 Docker container account adjustments

A Tenant Service requires a non-run Linux user **genesys** with **uid:gid** of 500:500 to be present and available to run tenant containers. If this user isn't available on nodes of K8s cluster, SecurityContextConstraint must be created. SecurityContextConstrain may need to be applied to the K8s account that Tenant Service is set to run. If a non-default account is being used (see K8s parameters), then it may be required to associate context with that particular account:

```
oc adm policy add-scc-to-user genesys-restricted -z tenant-<TENANT-UUID> -n <namespace>
```

For a sample security context policy for the **genesys** use, see Samples and References below.

7.2 Service-specific secrets

7.2.1 Postgres database backend

Database backend can be allocated shared or dedicated. A Tenant Service requires a separate database. Once deployed, secrets with details of postgres backend parameters must be created as follows:

```
kubectl create secret generic dbserver -n voice --from-literal="dbserver=<postgres service endpoint>"
kubectl create secret generic dbname -n voice --from-literal="dbname=<tenant DB name at postgres>"
kubectl create secret generic dbuser -n voice --from-literal="dbuser=<postgres user for tenant database>"
kubectl create secret generic dbpassword -n voice --from-literal="dbpassword=<password>"
```

7.2.2 Service account password

Default account allowing to access the Tenant Service config interface after initial deployment can be supplied a password via a secret. If not provided, **password** will be used as a default value (empty passwords are prohibited by a Tenant Service).

```
kubectl create secret generic svcaccount -n voice --from-literal="svcpassword=<password>"
```

7.2.3 GAUTH backend secrets

GWS/GAuth integration requires a client ID and token to allow a Tenant Service to register at GWS.

```
kubectl create secret generic gauthclientid -n voice --from-literal="clientid=<client id>"

kubectl create secret generic gauthclientsecret -n voice --from-literal="clientsecret=<token>"
```

7.3 Location-specific deployment steps

tenant-monitor

Monitoring/logging shared configuration and infrastructure deployment:

```
helm upgrade --install --force --wait --timeout 600s -n voice  tenant-monitor https://<jfrog artifactory/helm
location>/tenant-monitor-$TENANT_MANIFEST_VERSION.tgz  --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

To enable fluent bit logging and Prometheus monitoring, the following overrides can be used with tenant-monitor:

```
prometheus:
  podMonitor:
    create: "true"

fluent:
  enable: "true"
```

Enable Persistent Volume Claim to store tenant logs, then the following overrides can be used with tenant-monitor:

logStorageClass must be provided by the infrastructure team.

```
tenant:
  logging:
    volume:
      enabled: "true"
      createSC: "false"
      createpvClaim: "true"
      logClaim: "tenant-logs-pvc"
      logClaimSize: "5Gi"
      logStorageClass: "azure-files"
      Storageprovisioner: "TBD OC provosioner"
      parameters: {}
```

7.4 Service-specific deployment steps

7.4.1 Single service at one location

A single-service deployment can be implemented with these sample parameters in **tenant-node-values.yaml:**

```
tenantid: 9350e2fc-a1dd-4c65-8d40-1f75a2e080dd

replicaCount: 1
pgdb:
  dbhost: "/opt/genesys/dbserver/dbserver"
  dbuser: "/opt/genesys/dbuser/dbuser"
  dbname: "/opt/genesys/dbname/dbname"
  pgpwdSecretName: "/opt/genesys/dbpassword/dbpassword"
  usesecret: "true"

pullSecrets: mycred


image:
  registry: <docker-repo>
  tag: <version of tenant service>

initcontianer:
  registry: <docker-repo>
  tag: <version of tenant service>

addonpulseinitcontainer:
```

```
  registry: <docker-repo>
  tag: <version of tenant service>
  enable: "true"
  pulsemode: "setup"

rcsinitcontianer:
  enable: "true"
  registry: <docker-repo>
  tag: <version of roles service batch image>


resources:
  limits:
    cpu: "2"
    memory: 4Gi
  requests:
    cpu: "1"
    memory: 1Gi

containerSecurityContext:
  readOnlyRootFilesystem: false
  runAsNonRoot: true
  runAsUser: 500
  runAsGroup: 500

service:
  type: ClusterIP


tenant:
  serviceuser: "default"
  serviceuserpwd: "<random-password-string>"
  consul:
    acl:
      usetoken: "true"
      token: "/opt/genesys/consul-shared-secret/consul-consul-voice-token"

  volumes: |
    - name: fluent-logs
      emptyDir: {}
    - name: tenants-fluent-bit-config
      configMap:
        name: tenants-fluent-bit-config
    - name: consul-shared-secret
      secret:
        secretName: consul-voice-token
    - name: dbserver
      secret:
        secretName: dbserver
    - name: dbname
      secret:
        secretName: dbname
    - name: dbuser
      secret:
        secretName: dbuser
    - name: dbpassword
      secret:
        secretName: dbpassword
    - name: redis-config-secret
      secret:
        defaultMode: 420
        secretName: redis-config-token
    - name: kafka-secrets
      secret:
        defaultMode: 420
        secretName: kafka-secrets-token
    - name: redis-tenant-secret
      secret:
        defaultMode: 420
        secretName: redis-tenant-token
```

```
  volumeMounts: |
    - name: fluent-logs
      mountPath: "/opt/genesys/logs/JSON"
    - name: consul-shared-secret
      readOnly: true
      mountPath: "/opt/genesys/consul-shared-secret"
    - name: dbserver
      readOnly: true
      mountPath: "/opt/genesys/dbserver"
    - name: dbname
      readOnly: true
      mountPath: "/opt/genesys/dbname"
    - name: dbuser
      readOnly: true
      mountPath: "/opt/genesys/dbuser"
    - name: dbpassword
      readOnly: true
      mountPath: "/opt/genesys/dbpassword"
    - name: redis-config-secret
      readOnly: true
      mountPath: "/opt/genesys/redis-config-secret"
    - name: redis-tenant-secret
      readOnly: true
      mountPath: "/opt/genesys/redis-tenant-secret"
    - name: kafka-secrets
      readOnly: true
      mountPath: "/opt/genesys/kafka-secrets"


  initVolumeMounts: |
    - name: consul-shared-secret
      readOnly: true
      mountPath: "/opt/genesys/consul-shared-secret"
    - name: dbserver
      readOnly: true
      mountPath: "/opt/genesys/dbserver"
    - name: dbname
      readOnly: true
      mountPath: "/opt/genesys/dbname"
    - name: dbuser
      readOnly: true
      mountPath: "/opt/genesys/dbuser"
    - name: dbpassword
      readOnly: true
      mountPath: "/opt/genesys/dbpassword"


gws:
  enabled: "false"

redis:
  enabled: "true"
  port: 6379
  isCluster: true
  usesecret: "true"
  usesecretenv: "false"
  redisCacheSecretName: "/opt/genesys/redis-config-secret/redis-config-state"
  redisTenantSecretName: "/opt/genesys/redis-tenant-secret/redis-tenant-stream"

kafka:
  enabled: "true"
  usesecret: "true"
  usesecretenv: "false"
  kafkaSecretName: "/opt/genesys/kafka-secrets/kafka-secrets"

serviceAccount:
  create: true
```

In addition, use the following deployment command:

```
helm upgrade --install --force --wait --timeout 600s -n voice -f ./tenant-node-values.yaml tenant<shortid>
https://<jfrog artifactory/helm location>/tenant-<helm version>.tgz --username "$JFROG_USER" --password
"$JFROG_PASSWORD"
```

Above deployment will create a Tenant with the password of the service account set up explicitly and without enabling GWS integration. See Samples and references (below) for values that allow to reset the Tenant password upon deployment using a pre-generated value from the secret and to enable automated GWS integration.

7.5 Samples and references

Security context constraint for the **genesys** user:

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
 name: genesys
allowPrivilegedContainer: false
runAsUser:
 type: RunAsAny
seLinuxContext:
 type: RunAsAny
fsGroup:
 type: RunAsAny
supplementalGroups:
 type: RunAsAny
users:
- genesys
groups:
- genesys
```

Enabling a service admin password (the secret should be created in the Service account password step above):

```
...
tenant:
  serviceuser: "default"
  svcpwdSecretName: "/opt/genesys/service-user-account/svcpassword"
  ...
  volumes: |
    - name: service-user-account
      secret:
           secretName: svcuseraccount
    ...
  volumeMounts: |
    - name: service-user-account
      readOnly: true
      mountPath: "/opt/genesys/service-user-account"
    ....
  initVolumeMounts: |
    - name: service-user-account
      readOnly: true
      mountPath: "/opt/genesys/service-user-account"
    ....
```

Enabling GWS integration (the secret should be created in the GAUTH backend secrets step above):

```
...
gws:
  enabled: "true"
  usesecret: "true"
  gwsuserSecretName: "/opt/genesys/gauth-client-id/clientid"
  gwspwdSecretName: "/opt/genesys/gauth-client-token/clientsecret"

tenant:
  ...
  volumes: |
    - name: gauth-client-id
      secret:
           secretName: gauthclientid
    - name: gauth-client-token
      secret:
           secretName: gauthclientsecret
    ....

  volumeMounts: |
    - name: gauth-client-id
      readOnly: true
      mountPath: "/opt/genesys/gauth-client-id/clientid"
    - name: gauth-client-token
      readOnly: true
      mountPath: "/opt/genesys/gauth-client-token/clientsecret"
    ....
  initVolumeMounts: |
    - name: gauth-client-id
      readOnly: true
      mountPath: "/opt/genesys/gauth-client-id/clientid"
    - name: gauth-client-token
      readOnly: true
      mountPath: "/opt/genesys/gauth-client-token/clientsecret"
    ....
```

Mount Persistent Volume Claim to store Tenant logs, then add the following override values in **tenant-node-values.yaml:**

```
.....

tenant:
  ...
  volumes: |
    ...

    - name: log
      persistentVolumeClaim:
        claimName: tenant-logs-pvc

    ...

  volumeMounts: |
    .....

    - mountPath: /opt/genesys/logs/volume
      name: log
    - mountPath: /logs
      name: log

    ......

  initVolumeMounts: |
    ........

    - mountPath: /opt/genesys/logs/volume
      name: log
    - mountPath: /logs
      name: log

 .....
```

Deploy Voice Voicemail Service

Disclaimer

Voicemail Service is part of Voice Services in Genesys Engage cloud private edition is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

1. Introduction

Voicemail Service is part of Voice Services and provides the following functionality:

- Provides deposit of voicemail messages to agent and agent group mailboxes.
- Provides access to voice mailboxes by dialing to a voicemail access number.
- Uses the Config Service to retrieve agent configuration and states.
- Stores voicemail recordings and metadata in a storage system.
- Provisioning is done through Agent Setup.

2. Choosing Voicemail Storage

To store mailbox metadata and messages, consider the following supported options for storage in the Private Edition:

1. Persistent Volumes & Persistent Volume Claims
2. Azure Blob Storage

Consult sections that follow to learn how to use these storage options and their limitations.

2.1 Persistent Volume & Claim

- Persistent Volume (PV) is a piece of storage that can be mounted to a Voicemail Service deployment inside the Kubernetes cluster.
- PVs in OpenShift can be created with different plugins
    - Plugin Reference: https://cloud.netapp.com/blog/kubernetes-persistent-storage-why-where-and-how#h_h10
- Voicemail Service requires a separate storage class and PV to be created for a Voicemail storage.
- If the customer wants to extend the deployment to more than one Kubernetes cluster, Voicemail Service requires to mount the same PV for all the Kubernetes cluster for that customer.
- Create the Persistent Volume Claim (PVC) from the Voicemail PV.
- The access mode for the PVC must be **ReadWriteMany**, since the Voicemail Service will edit the existing data while updating the mailbox settings or the message state.
- Use the sizing doc (from the doc site) to calculate the required storage space.

Here is the sample K8s YAML file for creating PVCs for a Voicemail Service. The PVC creation is controlled by the Voicemail Service Helm chart by overriding the **values.yaml**.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: voice-voicemail-pvc
  namespace: voice
  labels:
    servicename: voice-voicemail
spec:
  storageClassName: voice-voicemail
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
```

2.1.1 Limitations

1. Replication strategies are not available for the data.
2. Retention limit: Admins can't configure the auto-expiration for a Voicemail message.
3. When a customer has more than one Kubernetes cluster deployed, the PV for all the K8s clusters must be created from a single storage drive, so that the data from one K8s cluster is shared among other K8s clusters.

2.2 Azure Blob

- Unlike PV, the Azure Blob Storage provides options to replicate and configure Time to live for the files and can be accessed from any K8s cluster by using the storage access keys.
- Create the Azure Storage with the blob storage.
- The access keys for the blob storage must be securely mounted to the Voicemail pod. You can do one of the following:
  - Store access keys in Azure Key Vault and mount it via a Container Storage Interface (CSI) driver.
  - Create access keys as a K8s secret and volume mount the K8s secret. (This option is considered less secured than the CSI driver approach.)
- The **values.yaml** file can be overridden for configuring either a K8s secret or CSI driver, which is explained under Section 4.1 Overriding values.yaml.

3. Prerequisites

Ensure to complete the following before proceeding with a Voicemail Service deployment:

- Complete Steps 1-5 in Deploy Voice Services before deploying a Voicemail Service.
- Choose a Voicemail storage to be created in Section 2.

The following services are required before proceeding with Voicemail deployment testing:

- Voice Tenant Service
- GVP Deployment
- GWS and Agent Setup (for accessing the UI and configuring the Voicemail solution)

4. Voicemail Deployment

1. Similar to Voice Services, a Voicemail Service also requires security context constraints for Genesys users. Since the security context already created for Voice Services, we reuse the same for the Voicemail Service. Use the following command to add the security context for a Voicemail Service:

```
oc adm policy add-scc-to-user genesys-restricted -z voice-voicemail -n voice
```

2. From the cloned repository (<repo location>/azure_voice_install/voice_helm_values), edit the **voicemail_override_values.yaml** file in the **voice_helm_values** directory as per required storage. See Section 4.1 Overriding values.yaml.
3. Install the Voicemail Helm chart with **overridden_values.yaml** (with the **-f** flag), below is the sample command.
   a. The pass "--set version=<voicemail node image version>"

```
helm upgrade --install --force --wait --timeout 300s -n voice -f ./overridden_values.yaml voice-
voicemail  https://<jfrog artifactory/helm location>/voice-voicemail/voice-voicemail-9.0.07.tgz --set
version=9.0.10 --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

4.1 Overriding values.yaml

Common changes:

```
#Configure the GWS Base URL
context:
...
  envs:
    ...
    gwsBaseUrl: <GWS URL> # sample URL https://<GWS_HOST>/auth/v3
    ...
```

Voicemail Service values to be overridden for the following parameters based on what kind of storage is chosen.

4.1.1 For PV and PVCs

```
#Blob Storage to be disabled and no value for mounts and volumes
  blobStorage:
    general:
      mode: 'k8s | csi'
      enabled: false
    mounts:
    volumes:

context:
...
  envs:
    ...
    storageInterface: "FileSystem"
    voicemailHome: "/storage/data"
    ...
```

4.1.2 For Azure Blob Storage

Azure Blob Storage with the CSI driver:

1. Store the Azure Blob storage account access keys into Key Vault with a key-value pair:
    a. key=<storage account name>
    b. value=<primary access key>
2. Create an CSI driver from the Key Vault into the K8s cluster with the name "keyvault-voice-voicemail-storage-csi-secrets".
3. Do the following changes in **values.yaml:**

```
#Blob Storage is enabled
  blobStorage:
    general:
      mode: 'k8s | csi' # Secrets needs to mounted via K8s or CSI driver
      enabled: true
    mounts:
      - name: voicemail-secrets
        readOnly: true
        mountPath: "/opt/genesys/katana/voicemail/secret"
    volumes:
      - name: voicemail-secrets
        csi:
          driver: secrets-store.csi.k8s.io
          readOnly: true
          volumeAttributes:
            secretProviderClass: keyvault-voice-voicemail-storage-csi-secrets


context:
...
  envs:
    ...
    storageInterface: "AzureBlob"
    voicemailHome: ""
    ...
```

Azure Blob Storage with a K8s secret:

1. Create a K8s secret cluster with the name "voicemail-storage-secrets" having a key-value pair:
    a. key=<storage account name>
    b. value=<primary access key>
2. Do the following changes in **values.yaml**:

```
#Blob Storage is enabled
  blobStorage:
    general:
      mode: 'k8s | csi' # Secrets needs to mounted via K8s or CSI driver
      enabled: true
    mounts:
      - name: voicemail-secrets
        readOnly: true
        mountPath: "/opt/genesys/katana/voicemail/secret"
    volumes:
      - name: voicemail-secrets

secret:
          secretName: voicemail-storage-secrets


context:
...
  envs:
    ...
    storageInterface: "AzureBlob"
    voicemailHome: ""
    ...
```

Now run the Helm install to deploy a Voicemail Service.

5 Upgrade of Voicemail Service

The upgrade procedure is the same as for other Voice Services and consists of these major steps:

1. Canary deployment
2. Upgrade
3. Delete canary

5.1 Canary Deployment

Other than the parameters mentioned in Section 4.1, override the following values in **canary_override_values.yaml**:

```
### Canary Override Values
serviceAccount:
  create: false   # Service account will be already created while initial deployment

service:
  canaryName: canary # Postfix that will be added for canary deployment

prometheus:
  podMonitor:
    enabled: false  # Podmonitor deployed during initial deployment will get metrics from all pod instance
including canary.

hpa:
  enabled: false  # HPA is not needed for canary
```

Deploy a canary instance:

```
helm upgrade --install --force --wait --timeout 500s -n voice -f ./voicemail_override_values.yaml -f .
/voice_helm_values/canary_override_values.yaml voice-voicemail-canary  https://<jfrog artifactory/helm
location>/voice-agent/voice-voicemail-9.0.07.tgz --set version=9.0.10 --username "$JFROG_USER" --password
"$JFROG_PASSWORD"
```

5.2 Service Upgrade

When the canary deployment of a Voicemail Service is ready for an upgrade, use the following command to upgrade the current version of a Voicemail Service to the desired version:

```
helm upgrade --install --force --wait --timeout 500s -n voice -f ./voicemail_override_values.yaml -f voice-voicemail-canary  https://<jfrog artifactory/helm location>/voice-agent/voice-voicemail-9.0.07.tgz --set version=9.0.10 --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

5.3 Delete Canary

If the upgrade of a Voicemail Service is successful, delete the canary instance of the service by using the following command:

```
helm delete voice-voicemail -n voice
```