This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Tenant Service Private Edition Guide

5/5/2026

Table of Contents

Overview

- About the Tenant Service 6
- Architecture 8
- High availability and disaster recovery 14

Configure and deploy

- Before you begin 15
- Configure the Tenant Service 20
- Provision the Tenant Service 34
- Deploy Tenant Service 35

Upgrade, roll back, or uninstall

- Upgrade, roll back, or uninstall Tenant Service 47

Observability

- Observability in Tenant Service 53
- Tenant Service metrics and alerts 57

Contents

- 1 Overview
- 2 Configure and deploy
- 3 Upgrade, roll back, or uninstall
- 4 Observability

Find links to all the topics in this guide.

Related documentation:

-
-
-
-

RSS:

- For private edition

The Tenant Service is a service available with the Genesys Multicloud CX private edition offering. The Tenant Service is included with the Voice Microservices, however there is a separate Private Edition Guide for the Voice Services. For information about the Voice Services, including provisioning, configuration, and deployment information, see the *Voice Microservices Private Edition Guide*.

Overview

Learn more about the Tenant Service, its architecture, and how to support high availability and disaster recovery.

- About the Tenant Service
- Architecture
- High availability and disaster recovery

Configure and deploy

Find out how to configure and deploy the Tenant Service.

- Before you begin
- Configure the Tenant Service
- Provision the Tenant Service
- Deploy Tenant Service

Upgrade, roll back, or uninstall

Find out how to upgrade, roll back, or uninstall Tenant services and to migrate the database.

- Upgrade, roll back, or uninstall Tenant Service

Observability

Learn how to monitor the Tenant Service with metrics and logging.

- Observability in Tenant Service
- Tenant Service metrics and alerts

About the Tenant Service

Contents

- 1 Supported Kubernetes platforms
- 2 Service description

Learn about the Tenant Service and how it works in Genesys Multicloud CX private edition.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Supported Kubernetes platforms

Voice Tenant Service is supported on the following Kubernetes platforms:

- Azure Kubernetes Service (AKS)
- Google Kubernetes Engine (GKE)
- OpenShift Container Platform (OpenShift)

See the Voice Microservices Release Notes for information about when support was introduced.

Service description

The Voice Tenant Service is included with the Voice Microservices and is a core service of the Genesys Multicloud CX platform. The Tenant Service is an application layer between front-end Genesys Multicloud CX solutions and shared back-end core services in a region.

The Voice Tenant Service instances are dedicated to a tenant of the Genesys Multicloud CX 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 the authentication workflow for a tenant's agents.

Architecture

Contents

- 1 Introduction
- 2 Architecture diagram — Connections
- 3 Connections table

Learn about Tenant Service architecture

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Introduction

The following diagram shows an example of the high-level architecture specific to the Tenant Service.

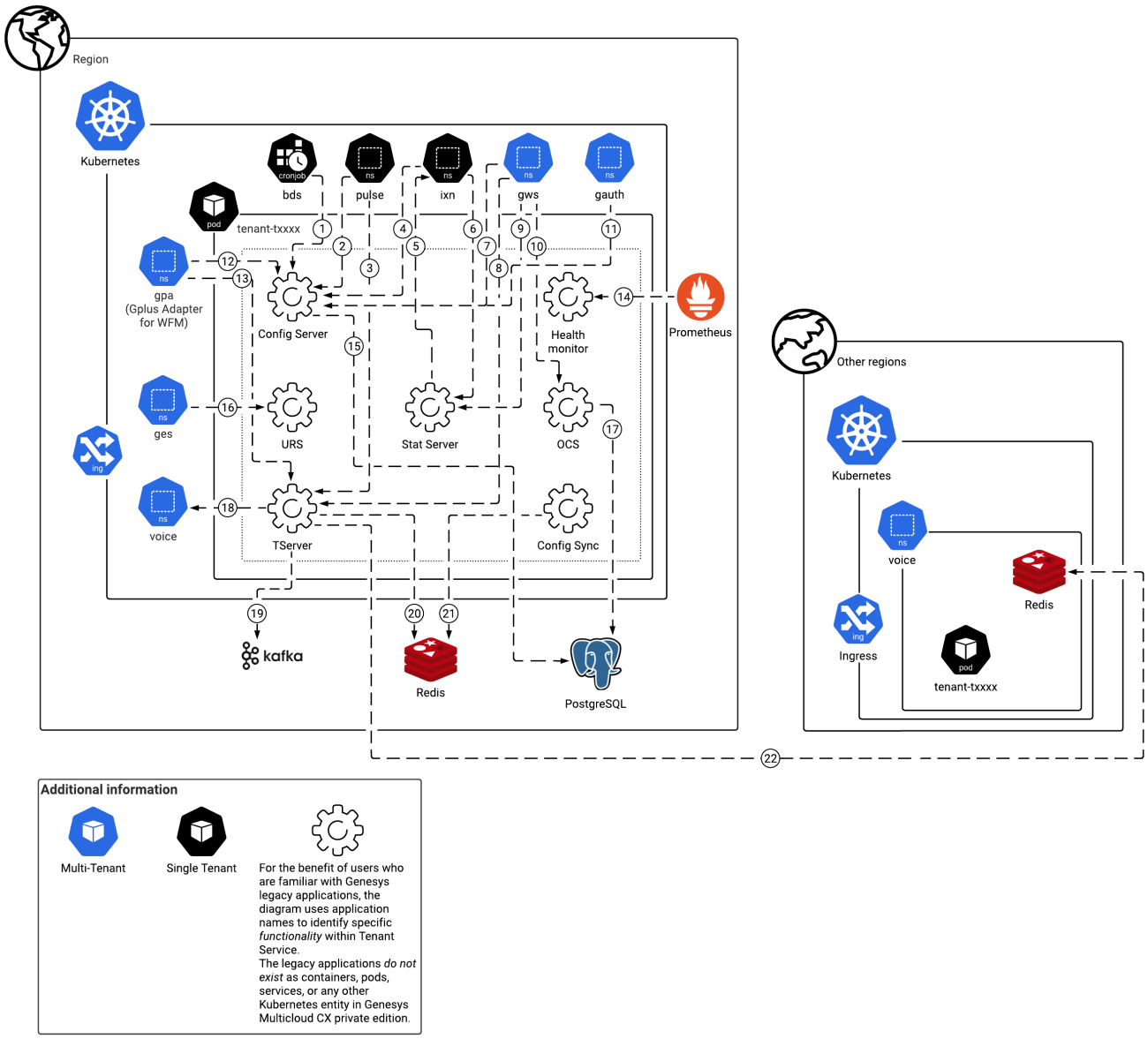
For the high-level architecture that includes all of the Voice Microservices, see Voice Microservices architecture.

For information about the overall architecture of Genesys Multicloud CX private edition, see the high-level Architecture page.

See also High availability and disaster recovery for information about high availability/disaster recovery architecture.

Architecture diagram — Connections

The numbers on the connection lines refer to the connection numbers in the table that follows the diagram. The direction of the arrows indicates where the connection is initiated (the source) and where an initiated connection connects to (the destination), from the point of view of Tenant Service as a service in the network.



Connections table

The connection numbers refer to the numbers on the connection lines in the diagram. The **Source**, **Destination**, and **Connection Classification** columns in the table relate to the direction of the arrows in the Connections diagram above: The source is where the connection is initiated, and the destination is where an initiated connection connects to, from the point of view of Tenant Service as a service in the network. *Egress* means the Tenant Service service is the source, and *Ingress* means the Tenant Service service is the destination. *Intra-cluster* means the connection is between services in the cluster.

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
1	Billing Data Service	Tenant Service	TCP	8888	Intra-cluster	Configuration and provisioning
2	Genesys Pulse	Tenant Service	TCP	8888	Intra-cluster	Configuration and provisioning
3	Genesys Pulse	Tenant Service	TCP	8000	Intra-cluster	Voice Microservices events
4	Interaction Server	Tenant Service	TCP	8888	Intra-cluster	Configuration and provisioning
5	Tenant Service	Interaction Server	TCP	7120	Intra-cluster	Multimedia transactions status
6	Interaction Server	Tenant Service	TCP	2060	Intra-cluster	Agent status for multimedia
7	Genesys Web Services and Applications	Tenant Service	TCP	8888		GWS (Configuration Service) access to provisioning
8	Genesys Web Services and Applications	Tenant Service	TCP	8000	Intra-cluster	GWS call control events
9	Genesys Web Services and Applications	Tenant Service	TCP	2060	Intra-cluster	GWS statistics
10	Genesys Web Services and Applications	Tenant Service	TCP	5050	Intra-cluster	Outbound campaign control through GWS
11	Genesys Authentication	Tenant Service	TCP	8888	Intra-cluster	Genesys Authentication access to provisioning
12	Gplus Adapters for WFM	Tenant Service	TCP	8888	Intra-cluster	Configuration and provisioning
13	Gplus Adapters for WFM	Tenant Service	TCP	8000	Intra-cluster	Voice Microservices events
14	Prometheus	Tenant	HTTP	15000	Ingress	Tenant

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
		Service				Service provides metrics for monitoring and alerting with Prometheus.
15	Tenant Service	PostgreSQL	TCP	5432	Egress	Persistent SQL storage for provisioning data
16	Genesys Engagement Service	Tenant Service	HTTP	5580	Intra-cluster	Routing requests and events
17	Tenant Service	PostgreSQL	TCP	5432	Egress	Persistent storage for outbound campaigns and calling lists
18	Tenant Service	Voice Microservices				For information, see connections 16, 27, and 32 in the Voice Microservices.
19	Tenant Service	Kafka	TCP	9092/9093	Egress	Outbound reporting
20	Tenant Service	Redis	TCP	6379	Egress	Voice Microservices call control events
21	Tenant Service	Redis	TCP	6379	Egress	Tenant configuration and provisioning synchronization for in-memory caching
22	Tenant Service	Redis	TCP	6379	Intra-cluster	Cross-region Voice Microservices call control events in

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
						remote Redis

High availability and disaster recovery

Find out how this service provides disaster recovery in the event the service goes down.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Service	High Availability	Disaster Recovery	Where can you host this service?
Tenant Service	N = N (N+1)	Active-spare	Primary or secondary unit

See High Availability information for all services: High availability and disaster recovery

Before you begin

Contents

- 1 Limitations and assumptions
- 2 Download the Helm charts
  - 2.1 Containers
  - 2.2 Helm charts
- 3 Third-party prerequisites
- 4 Storage requirements
- 5 Network requirements
- 6 Browser requirements
- 7 Genesys dependencies
  - 7.1 Specific dependencies
- 8 GDPR support

Find out what to do before deploying the Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Limitations and assumptions

Not applicable

Download the Helm charts

For information about how to download the Helm charts, see Downloading your Genesys Multicloud CX containers.

See Helm charts and containers for Voice Microservices for the Helm chart version you must download for your release.

Containers

The Tenant Service has the following containers:

- Core tenant service container
- Database initialization and upgrade container
- Role and privileges initialization and upgrade container
- Solution specific: pulse provisioning container

Helm charts

- Tenant deployment
- Tenant infrastructure

Third-party prerequisites

For information about setting up your Genesys Multicloud CX private edition platform, see Software Requirements.

The following table lists the third-party prerequisites for the Tenant Service.

Third-party services

Name	Version	Purpose	Notes
Kafka	2.x	Message bus.	
Consul	1.13.x	Service discovery, service mesh, and key/value store.	Starting with version 100.0.100.0041, Tenant Service supports Consul 1.10.
Redis	6.x	Used for caching. Only distributions of Redis that support Redis cluster mode are supported, however, some services may not support cluster mode.	

PostgreSQL	11.x	Relational database.	**NOTE:** Starting with version 100.0.100.0041, Tenant Service supports PostgreSQL 11.x.
			
			Before deploying Tenant Service, you must provision a PostgreSQL database for Tenant Service using one of the following methods:
			• Create a PostgreSQL database specifically for use by Tenant Service.
			• Use the shared PostgreSQL database, which is recommended in the OpenShift platform.
			
			Deploy a PostgreSQL database using the following commands:
			`helm repo add bitnami https://charts.bitnami.com/bitnami`
			`helm install -n voice postgres`

Name	Version	Purpose	Notes
			`bitnami/postgresql` During deployment, you require the database information, including credentials. For the list of database parameters that you override in the Tenant Helm chart **values.yaml** file, see Backend parameters in Override Helm chart values.
A container image registry and Helm chart repository		Used for downloading Genesys containers and Helm charts into the customer's repository to support a CI/CD pipeline. You can use any Docker OCI compliant registry.	

Storage requirements

For information about storage requirements for Voice Microservices, including the Tenant Service, see Storage requirements in the *Voice Microservices Private Edition Guide*.

Network requirements

For general network requirements, review the information on the suite-level Network settings page.

Browser requirements

Not applicable

Genesys dependencies

For detailed information about the correct order of services deployment, see Order of services deployment.

The following prerequisites are required before deploying the Tenant Service:

- **Voice Platform** and all its external dependencies must be deployed before proceeding with the Tenant Service deployment.
- **PostgreSQL 10** database management system must be deployed and database shall be allocated either as a primary or replica. For more information about the sample deployment of a standalone DBMS, see Third-party prerequisites.

In addition, if you expect to use Agent Setup or Workspace Web Edition after the tenant is deployed, Genesys recommends that you deploy **GWS Authentication Service** before proceeding with the Tenant Service deployment.

Specific dependencies

The Tenant Service is dependent on the following platform endpoints:

- GWS environment API
- Interaction service core
- Interaction service vq

The Tenant Service is dependent on the following service component endpoints:

- Voice Front End Service
- Voice Redis (RQ) Service
- Voice Config Service

GDPR support

Not applicable.

Configure the Tenant Service

Contents

- 1 Override Helm chart values
- 2 Configure Kubernetes
- 3 Configure security
  - 3.1 Security context configuration
  - 3.2 Configure service-specific secrets

Learn how to configure the Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Override Helm chart values

For additional information about overriding Helm chart values, see Overriding Helm Chart values in the *Genesys Multicloud CX Private Edition Guide*.

This section describes the purpose and use case for each configurable parameter in a Tenant Service deployment.

The content in the following tables is not intended to be actual values or the names of override options for the Helm charts; you can extract those later from the **values.yaml** file for each Helm chart.

Versioning

Group	Name	Purpose	Comments
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.	

Identification

Group	Name	Purpose	Comments
Tenant	name	Nickname of the tenant.	Human-readable name. The default value is the Helm release name.

Group	Name	Purpose	Comments
Tenant	uuid	Unique identifier of all instances of the Tenant Service.	All nodes deployed to handle the end-customer environment use that UUID that is also registered. The last four positions of the UUID are used as the short Tenant ID, when applicable.

Backend parameters

Group	Name	Purpose	Comments
postgres	database host	A reference to the backend DBMS into which to persist the service.	Either a direct value or a file path that points to a mapped volume with file content to be used as the DBMS name.
postgres	database user	A reference to the backend database into which to persist the service.	Either a direct value or a file path that points to a mapped volume with file content to be used as the database username.
postgres	database name	A reference to the backend database into which to persist the service.	Either a direct value or a file path that points to a mapped volume with file content to be used as the database name.
postgres	database password parameters	Either direct value or reference to a secret name and key that hold a value; depends on relevant Kubernetes secret flag and Kubernetes 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 will be mounted as a volume and password exposed via file. Secret key can be specified if password will be mapped to environment variable.
postgres	Kubernetes secret usage flags	Indication of Kubernetes secret being used to keep password and whenever secret shall be mounted to env variable or expected to be mapped as a	Boolean values that force use of secret (instead of direct database password).

Group	Name	Purpose	Comments
		volume.	
postgres	ssl usage	specify secure connection preferences	Allow or require TLS.
consul	Kubernetes secret usage flags	indication of Kubernetes 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 Kubernetes secret flag and Kubernetes 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	Kubernetes secret usage flags	indication of Kubernetes 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 Kubernetes secret flag and Kubernetes 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 Kubernetes	these parameters control how string is being extracted by service. direct value is supported for testing

Group	Name	Purpose	Comments
		secret flag and Kubernetes secret env map flag	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	Kubernetes secret usage flags	indication of Kubernetes 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 Kubernetes secret flag and Kubernetes 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

Kubernetes 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

Group	name	purpose	comments
			tenant's POD having Kubernetes 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 Authorization	security context	Adjust security context to run with random user	If random user is used to launch tenant containers, an adjustment to security context is needed for all tenant containers. Tenant is running as user 500 in group/ fsgroup 500 by default and it may use group/ fsgroup 0 for random user. For more information, see Security context configuration.
Auth and Autorization	Pod identity	specify optional annotation to associate with POD in order to access Kubernetes resources	PODs may be assigned an ADODB identity 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 Kubernetes 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.

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, the following volume is added:
			`- name: fluent-logs`
			`emptyDir: {}`
			and volume mount:
			`- name: fluent-logs`
			`mountPath: "/opt/genesys/logs/JSON"`

file logging	logging persistent volume	Specify usage of persistent volumes for logging by tenant components that produce log files.	Tenant PODs won't use persistent volume claims by default and no storage classes are created. Persistent

Group	name	purpose	comments
			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 Kubernetes empty directory volume	part of default mounts provided for tenant service

Monitoring and observability

Group	name	purpose	comments
prometheus	Prometheus PodMonitor	Specify if PodMonitor to scrape Prometheus endpoints of tenant pods should be created.	Tenant-monitor doesn't create PodMonitor 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.	

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

Group	name	purpose	comments
			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	Kubernetes secret usage flags	indication of Kubernetes 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 Kubernetes secret flag and Kubernetes 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

Group	name	purpose	comments
			allocated for service mesh.
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	Kubernetes secret usage flags	indicate usage of Kubernetes 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 Kubernetes secret flag and Kubernetes 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

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

Group	name	purpose	comment
			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

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

Group	name	purpose	comment
			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 be bound
kafka			Only main container needs to be bound

Configure Kubernetes

For information, see the following resources:

- Override Helm chart values
- Configure security
- Configure service-specific secrets
- Deploy Tenant Service

Configure security

Before you deploy the Tenant Service, be sure to read Security Settings in the *Setting up Genesys Multicloud CX Private Edition* guide.

Security context configuration

By default, the user and group IDs are set in the **values.yaml** file as `500:500:500`, meaning the **genesys** user.

```
containerSecurityContext:
    # 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
    readOnlyRootFilesystem: false
    runAsNonRoot: true
    # base/centos7 uses uid=500
    runAsUser: 500
    runAsGroup: 500

  securityContext:
    # ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-
security-context-for-a-pod
    # fsGroup is only valid at pod level
    fsGroup: 500
```

Arbitrary UIDs in OpenShift

If you want to use arbitrary UIDs in your OpenShift deployment, you must override the **securityContext** settings in the **values.yaml** file, so that you do not define any specific IDs.

```
containerSecurityContext:
    # 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
    readOnlyRootFilesystem: false
    runAsNonRoot: true
    # base/centos7 uses uid=500
    runAsUser: null
    runAsGroup: 0

securityContext:
    # ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-
security-context-for-a-pod
    # fsGroup is only valid at pod level
    fsGroup: null
```

Configure service-specific secrets

Postgres database backend

Database backend can be allocated as shared or dedicated. The 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="
kubectl create secret generic dbname -n voice --from-literal="dbname="
kubectl create secret generic dbuser -n voice --from-literal="dbuser="
kubectl create secret generic dbpassword -n voice --from-literal="dbpassword="
```

Service account password

The default account that allows access to the Tenant Service config interface after initial deployment can be supplied a password through a secret. If not provided, **password** will be used as a default value (empty passwords are prohibited by the Tenant Service).

```
kubectl create secret generic svcuseraccount -n voice --from-literal="svcpassword="
```

Genesys Authentication backend secrets

Genesys Web Services (GWS)/Genesys Authentication integration requires a client ID and token to allow the Tenant Service to register at GWS.

```
kubectl create secret generic gauthclientid -n voice --from-literal="clientid="
```

```
kubectl create secret generic gauthclientsecret -n voice --from-literal="clientsecret="
```

Provision the Tenant Service

- Administrator

Learn how to provision the Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

There are no specific steps to provision Tenant Service itself. When you deploy Tenant Service or perform an upgrade, Tenant Service provisioning is done automatically.

After you complete the deployment of Tenant Service, use Agent Setup to provision the agents, DNs, and other objects that describe your environment and to enable features in your contact center. For more information about Agent Setup and how to use it to provision and enable features and functionality in your contact center, see Get started with Agent Setup.

When properly deployed, Tenant Service is immediately operational and ready to contain the provisioning information for every feature of the platform.

Deploy Tenant Service

Contents

- 1 Assumptions
- 2 Deployment scenarios
  - 2.1 Single region/location/cluster
  - 2.2 Multiple regions/locations/clusters: Basic deployment
- 3 Deploy the service
  - 3.1 Prerequisites
  - 3.2 Location-specific deployment steps
  - 3.3 Service-specific deployment steps: Single service at one location
- 4 Samples and references
- 5 Validate the deployment

Learn how to deploy Tenant Service into a private edition environment.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Assumptions

- The instructions on this page assume you are deploying the service in a service-specific namespace, named in accordance with the requirements on Creating namespaces. If you are using a single namespace for all private edition services, replace the namespace element in the commands on this page with the name of your single namespace or project.
- Similarly, the configuration and environment setup instructions assume you need to create namespace-specific (in other words, service-specific) secrets. If you are using a single namespace for all private edition services, you might not need to create separate secrets for each service, depending on your credentials management requirements. However, if you do create service-specific secrets in a single namespace, be sure to avoid naming conflicts.

For an overview of solution-level deployment, see the deployment tour.

Deployment scenarios

More than one deployment scenario is supported for Tenant Service, including single region, redundant, and multi-region deployment as well as multi-Tenant deployment.

Single region/location/cluster

You deploy Tenant resources in a single Kubernetes 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.

The Tenant deployment process creates resources using a *release name* parameter, specified when executing the Helm deployment step. When installed in a single namespace, you must make sure that the release name value is distinct across all Tenants and other deployments.

For example, you might specify the Helm release name in the format `t`. Optionally, if you want the Tenant service name to match other Voice services, you can prefix the Tenant name with `voice-` in the Helm release name. So, in this example, you would specify the release name as `voice-t` during Helm deployment. The value for is the last four characters of the Tenant UUID that you configure in the **values.yaml** file. For more information about the identification parameters for the Tenant service, see Identification.

If you plan to use Prometheus monitoring or Fluent Bit logging framework for Tenant, you must execute the **tenant-monitor** module, as described in tenant-monitor. The module enables the following features:

- Prometheus PodMonitor definition for all tenant pods.
- Common Fluent Bit framework configuration for all tenant pods.

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.

To increase the number of nodes, adjust the **node count** parameter. For more information, see Scalability and redundancy parameters.

Upgrade

For information about upgrading Tenant Service, see Upgrade, rollback, or uninstall the Tenant Service.

Multiple Tenants at one location: Basic deployment

You can deploy additional Tenants at the same location using the following guidelines:

- Each Tenant Service must have a unique tenant uuid, shortid, and nickname.
- Each Tenant Service is deployed or upgraded and adjusted independently.

Multiple regions/locations/clusters: Basic deployment

In multi-regional/multi-location deployments, one region/location is considered "master" (from the 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 through the config API) or have all its nodes running only as replicas (`read` access to configuration).

Multi-regional deployments must be performed using the following steps (with prerequisites already satisfied at each region/location):

- If required, deploy the **tenant-monitor** module at a location planned as a Master Tenant node.
- Complete the basic deployment of a Tenant Service in the Master region, including specification of DR parameters for the Master, as per Scalability and redundancy parameters.
- Complete the deployment of the database backend with a replica of the Master database at the location(s) where the replica Tenant nodes are expected to run, including provisioning of access keys/

secrets to access the local replica.

- If required, deploy the **tenant-monitor** module at location(s) where replicas are expected to run.
- Complete the basic Tenant deployment for additional region(s) and specify DR parameters for the Master region (see Scalability and redundancy parameters).

The same customization scenarios described for Tenant nodes can be applied for each location independently.

Upgrade

For information about upgrading Tenant Service, see Upgrade, rollback, or uninstall the Tenant Service.

Deploy the service

This section provides reference commands with key parameters that are required to complete each deployment step.

On this page, the **tenant-values.yaml** file refers to the **values.yaml** file in the Tenant Helm chart. Likewise, the **tenant-monitor-values.yaml** file refers to the **values.yaml** file in the Tenant Monitor Helm chart.

Prerequisites

- Read Before you begin for the full list of prerequisites required to deploy the Tenant Service.
- Mandatory parameter values for basic installation are:
  - tenant uuid (v4)
  - tenant nickname (becomes a Helm release name)
  - all backend parameters (along with all secrets that may be required based on these parameters)
- Before proceeding with the Tenant Service deployment, ensure you have completed procedures in the Configure security section of this guide.
- Ensure you have configured all required overrides in the Helm chart **values.yaml** files, including specifying the correct SIP domain. For information, see Override Helm chart values.

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:///tenant-monitor-$TENANT_MANIFEST_VERSION.tgz  --username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

To enable Prometheus monitoring, you can use the following overrides with tenant-monitor. Use the following changes in the **tenant-monitor-values.yaml** file to implement the changes:

```
prometheus:
  podMonitor:
    create: "true"
```

To enable Fluent Bit to send additional logs to stdout in json format (for selected Tenant functions, such as configuration audit) and/or raw format (such as from internal applications such as StatServer and URS), modify the following changes to upgrade tenant-monitor. Use the following changes in the **tenant-monitor-values.yaml** file to implement the changes.

```
fluent:
  enable: "true"
  rawlogs:
    stdout:
      enable: "true"
  jsonlogs:
    stdout:
      enable: "true"
```

To enable RWX Persistent Volume Claim (PVC) in tenant-monitor to store Tenant logs shared across all Tenant pods, make the following modifications to override values in the **tenant-monitor-values.yaml** file:

```
tenant:
  logging:
    volume:
      enabled: "true"
      createSC: "false"
      createpvClaim: "true"
      logClaim: "tenant-logs-pvc"
      logClaimSize: "5Gi"
      logStorageClass: ""
      Storageprovisioner: "TBD OC provisioner"
      parameters: {}
```

RWX PV is disabled by default; no overrides are required in the **tenant-monitor-values.yaml** file to disable it.

Service-specific deployment steps: Single service at one location

A PostgreSQL database must be available for the Tenant Service before you begin the service deployment. For more information about the database requirements, see Third-party prerequisites. In addition, after the PostgreSQL database is deployed and before you deploy the Tenant Service, you must configure secrets that contain values for certain PostgreSQL database parameters. To configure

the secrets, see Service-specific secrets.

Use the following template if you are deploying with the tenant Helm chart. A single-service deployment can be implemented with the following sample parameters in the **tenant-values.yaml** file:

```
##UUID 4 format ( Set a new UUID for new tenant deployment)
tenantid:


serviceAccount:
    create: true

images:
  imagePullSecrets: mycred
  registry:
  pullPolicy: Always
  tenant:
    tag:

  pgdbInit:
    tag:

  rcsInit:
    tag:
    enable: "true"

  pulseInit:
    tag:
    enable: "true"
    pulseMode: "setup"



tenant:
  general:
    upstreamServices: voice-sipfe:9101,voice-config:9100,ixn-server-{{ $.Values.tenantid
}}:7120,ixn-vqnode-{{ $.Values.tenantid }}:7122"
  pgdb:
    dbhost: "/opt/genesys/dbserver/dbserver"
    dbuser: "/opt/genesys/dbuser/dbuser"
    dbname: "/opt/genesys/dbname/dbname"
  securityContext:
    fsGroup: 0

  logging
 ....
  volumes:
    logPvc:
        enabled: "true"
        logClaimSize: "5Gi"
        accessModes: "ReadWriteOnce"
        logStorageClass: ""   #Replace the storage class with a relevant storage class for
the cluster type

    mounts:
      log:
        - name: log
          mountPath: /opt/genesys/logs/volume
        - name: log
          mountPath: /logs
  secrets:
```

```
pgdb:
  pwd:
    secretName: "/opt/genesys/dbpassword/dbpassword"
    secretKey: "dbpassword"
  volumes: |
    - name: dbpassword
      secret:
        secretName: dbpassword
    - name: dbserver
      secret:
        secretName: dbserver
    - name: dbname
      secret:
        secretName: dbname
    - name: dbuser
      secret:
        secretName: dbuser
  mounts:
    - name: dbpassword
      readOnly: true
      mountPath: "/opt/genesys/dbpassword"
    - name: dbserver
      readOnly: true
      mountPath: "/opt/genesys/dbserver"
    - name: dbname
      readOnly: true
      mountPath: "/opt/genesys/dbname"
    - name: dbuser
      readOnly: true
      mountPath: "/opt/genesys/dbuser"

consul:
  acl:
    secretName: "/opt/genesys/consul-shared-secret/consul-consul-voice-token"
  volumes:
    - name: consul-shared-secret
      secret:
        secretName: consul-voice-token
  mounts:
    - name: consul-shared-secret
      readOnly: true
      mountPath: "/opt/genesys/consul-shared-secret"
redis:
  configPwd:
    secretName: "/opt/genesys/redis-config-secret/redis-config-state"
    volumes:
      - name: redis-config-secret
        secret:
          secretName: redis-config-token
    mounts:
      - name: redis-config-secret
        readOnly: true
        mountPath: "/opt/genesys/redis-config-secret"

  streamPwd:
    secretName: "/opt/genesys/redis-tenant-secret/redis-tenant-stream"
    volumes:
      - name: redis-tenant-secret
        secret:
          secretName: redis-tenant-token
    mounts:
      - name: redis-tenant-secret
        readOnly: true
```

```
            mountPath: "/opt/genesys/redis-tenant-secret"

    kafka:
      pwd:
         secretName: "/opt/genesys/kafka-secrets/kafka-secrets"
      volumes:
        - name: kafka-secrets
          secret:
            secretName: kafka-secrets-token
      mounts:
        - name: kafka-secrets
          mountPath: "/opt/genesys/kafka-secrets"
    gws:
      user:
        secretName: "/opt/genesys/gauth-client-id/clientid"
      pwd:
        secretName: "/opt/genesys/gauth-client-token/clientsecret"
      volumes:
        - name: gauth-client-id
          secret:
            secretName: gauthclientid
        - name: gauth-client-token
          secret:
            secretName: gauthclientsecret
      mounts:
        - name: gauth-client-id
          readOnly: true
          mountPath: "/opt/genesys/gauth-client-id"
        - name: gauth-client-token
          readOnly: true
          mountPath: "/opt/genesys/gauth-client-token"



  redis:
    isCluster: true
```

In addition, use the following deployment command:

```
helm upgrade --install --force --wait --timeout 600s -n voice -f ./tenant-node-values.yaml t \
https:///tenant-.tgz \
--username "$JFROG_USER" --password "$JFROG_PASSWORD"
```

The preceding deployment will create a Tenant with the password of the service account set up explicitly and without enabling GWS integration. See Samples and references for values that allow you to reset the Tenant password upon deployment using a pre-generated value from the secret and to enable automated GWS integration.

Samples and references

Enabling a service admin password (the secret should be created as described in the Service account password section):

```
...
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"
    ....
```

To enable stdout log output for all Tenant components, make the following modifications to override values in the **tenant-values.yaml** file.

```
images
fbregistry: fluent/fluent-bit
...
  fluentBit:
    enable: "true"
    name: json-sidecar
    tag: 1.8.x

  fluentBitUrs:
    enable: "true"
    name: stdouturs-sidecar
    tag: 1.8.x

  fluentBitSs:
    enable: "true"
    name: stdoutss-sidecar
    tag: 1.8.x

  fluentBitOcs:
    enable: "true"
    name: stdoutocs-sidecar
    tag: 1.8.x

  fluentBitCs:
    enable: "true"
    name: stdoutcs-sidecar
    tag: 1.8.x

tenant:
 ...
  logging:
    volumes:
      log:
        - name: log
      jsonLog:
        - name: fluent-logs
          emptyDir: {}
      stdoutUrsLog:
        - name: fluenturs-logs
          emptyDir: {}
      stdoutOcsLog:
        - name: fluentocs-logs
          emptyDir: {}
      stdoutSsLog:
```

```
      - name: fluentss-logs
        emptyDir: {}
    stdoutCsLog:
      - name: fluentcs-logs
        emptyDir: {}
    fluentBconfigmap:
      - configMap:
          defaultMode: 420
          name: tenants-fluent-bit-config
        name: tenants-fluent-bit-config
    fluentBconfigmapCs:
      - configMap:
          defaultMode: 420
          name: tenants-fluent-bit-config-cs
        name: tenants-fluent-bit-config-cs
    fluentBconfigmapSs:
      - configMap:
          defaultMode: 420
          name: tenants-fluent-bit-config-ss
        name: tenants-fluent-bit-config-ss
    fluentBconfigmapOcs:
      - configMap:
          defaultMode: 420
          name: tenants-fluent-bit-config-ocs
        name: tenants-fluent-bit-config-ocs
    fluentBconfigmapUrs:
      - configMap:
          defaultMode: 420
          name: tenants-fluent-bit-config-urs
        name: tenants-fluent-bit-config-urs
  mounts:
    log:
      - name: log
        mountPath: /opt/genesys/logs/volume
      - name: log
        mountPath: /logs
    jsonLog:
      - name: fluent-logs
        mountPath: "/opt/genesys/logs/JSON"
    stdoutUrsLog:
      - name: fluenturs-logs
        mountPath: "/opt/genesys/logs/URS"
    stdoutSsLog:
      - name: fluentss-logs
        mountPath: "/opt/genesys/logs/SS"
    stdoutOcsLog:
      - name: fluentocs-logs
        mountPath: "/opt/genesys/logs/OCS"
    stdoutCsLog:
      - name: fluentcs-logs
        mountPath: "/opt/genesys/logs/confserv"

    fbJsonLog:
      - name: fluent-logs
        mountPath: "/mnt/logs"
    fbstdoutUrsLog:
      - name: fluenturs-logs
        mountPath: "/mnt/logs"
    fbstdoutSsLog:
      - name: fluentss-logs
        mountPath: "/mnt/logs"
    fbstdoutOcsLog:
      - name: fluentocs-logs
```

```
        mountPath: "/mnt/logs"
    fbstdoutCsLog:
      - name: fluentcs-logs
        mountPath: "/mnt/logs"

    fluentBconfigmap:
        - mountPath: /fluent-bit/etc/
          name: tenants-fluent-bit-config
    fluentBconfigmapCs:
        - mountPath: /fluent-bit/etc/
          name: tenants-fluent-bit-config-cs
    fluentBconfigmapSs:
        - mountPath: /fluent-bit/etc/
          name: tenants-fluent-bit-config-ss
    fluentBconfigmapOcs:
        - mountPath: /fluent-bit/etc/
          name: tenants-fluent-bit-config-ocs
    fluentBconfigmapUrs:
        - mountPath: /fluent-bit/etc/
          name: tenants-fluent-bit-config-urs
```

You can deploy Persistent Volume/Persistent Volume Claim (PV/PVC) in two ways:

1. Enable ReadWriteOnce (RWO) from the tenant Helm chart, which maintains unique PVCs for each pod/ replica from the same Tenant.
2. Enable ReadWriteMany (RWX) from the tenant-monitor Helm chart, which has multiple Tenant pods sharing the same PVC.

To enable RWO PV/PVC logging from individual Tenant pods in Statefulset, make the following modifications to override the values in the **tenant-values.yaml** file. RWO Persistent Volume is disabled by default.

```
logging
 ....
  volumes:
    logPvc:
        enabled: "true"
        logClaimSize: "5Gi"
        accessModes: "ReadWriteOnce"
        logStorageClass: ""   #Replace the storage class that's relevant to the Openshift
Cluster
```

Enabling GWS integration (the secret should be created as described in the Genesys Authentication backend secrets section):

```
tenant:
…
gws:
    # enable: Enable GWS registeration about tenant
    # tls: Enable/Disable Secure connection to GWS
    # authEndpoint: GWS auth end point
    # envEndpoint: GWS env end point for Registeration
    # db: Pass DB infromation for GWS to connect to PSQL DB for read and store data
    enable: "true"
    tls: false
    authEndpoint: "gauth-auth.gauth.svc.cluster.local"
    envEndpoint: ""
    db:
      enable: "false"
      read: "false"
```

```
      init: "false"
             secrets:
……….
gws:
      enabled: true
      user:
        secretName: "/opt/genesys/gauth-client-id/clientid"
      pwd:
        secretName: "/opt/genesys/gauth-client-token/clientsecret"
      volumes:
        - name: gauth-client-id
          secret:
            secretName: gauthclientid
        - name: gauth-client-token
          secret:
            secretName: gauthclientsecret
      mounts:
        - name: gauth-client-id
          readOnly: true
          mountPath: "/opt/genesys/gauth-client-id"
        - name: gauth-client-token
          readOnly: true
          mountPath: "/opt/genesys/gauth-client-token"
```

To mount the PVC to store Tenant logs, make the following modifications to override the values in the **tenant-values.yaml** file:

```
.....

tenant:
  ...
    logging:
    mounts:
      log:
        - name: log
          mountPath: /opt/genesys/logs/volume
        - name: log
          mountPath: /logs
```

Validate the deployment

Content coming soon

Upgrade, roll back, or uninstall Tenant Service

Contents

- 1 Supported upgrade strategies
  - 1.1 Single region/location/cluster
  - 1.2 Multiple regions/locations/clusters
- 2 Backend upgrade
- 3 Timing
  - 3.1 Scheduling considerations
- 4 Monitoring
- 5 Preparatory steps
- 6 Rolling Update
  - 6.1 Rolling Update: Upgrade
  - 6.2 Rolling Update: Verify the upgrade
  - 6.3 Rolling Update: Rollback
  - 6.4 Rolling Update: Verify the rollback
- 7 Uninstall

Learn how to upgrade, roll back, or uninstall Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

> **Important**
> The instructions on this page assume you have deployed the services in service-specific namespaces. If you are using a single namespace for all private edition services, replace the namespace element in the commands on this page with the name of your single namespace or project.

Supported upgrade strategies

Tenant Service supports the following upgrade strategies:

Service	Upgrade Strategy	Notes
Tenant Service	Rolling Update	

For a conceptual overview of the upgrade strategies, refer to Upgrade strategies in the Setting up Genesys Multicloud CX Private Edition guide.

Tenant Service uses a rolling upgrade process. When multiple Tenant nodes are deployed, you can perform an upgrade to your existing deployment without causing a complete service outage. Tenant instances are rolled over, one by one, affecting only a portion of your agents each time.

Upgrading the master Tenant instance (node "0", which is responsible for `write` access to the provisioning data) causes a temporary degradation of functionality with no `write` access until instance "0" is restored.

Single region/location/cluster

Genesys recommends performing a backup of the backend database for Tenant Service before upgrading.

To perform an upgrade, follow the deployment process – use the same `helm upgrade` command and the same mandatory parameters. Remember to update the Tenant image(s) and Helm charts version tags in the **values.yaml** file. The upgrade is performed automatically, one node at a time (if **node count** is > 1). When you upgrade the primary node, the Tenant configuration updates automatically.

For information about the deployment process, see Deployment scenarios and Deploy the service.

Multiple regions/locations/clusters

Genesys recommends performing a backup of the backend database for Tenant Service before you upgrade the master region.

To perform an upgrade, follow the deployment process – use the same `helm upgrade` command and the same mandatory parameters. Remember to update the Tenant image(s) and Helm charts version tags in the **values.yaml** file for every location. When you upgrade the master node in the master region, the Tenant configuration updates automatically.

For information about the deployment process, see Deployment scenarios and Deploy the service.

Backend upgrade

When you perform an upgrade to Tenant Service, Tenant Service provisioning is done automatically, as part of the `helm upgrade` step. There is no need to make any provisioning changes, unless noted in Release Notes for a particular version.

If you must perform an upgrade or maintenance to the database management system (DBMS) that Tenant Service uses to store its provisioning data, then the DBMS upgrade or maintenance can be performed in place, separate from Tenant Service. Perform the DBMS upgrade according to the vendor's instructions, making sure that:

- Tenant Service is disconnected from the backend during maintenance, and
- you specify the same parameters that were provisioned during the Tenant Service upgrade in order for Tenant Service to access the updated database.

Be aware that the Tenant Service is in a degraded state (no `write` access to provisioning) during the database upgrade.

If the database upgrade involves moving to a new DBMS instance with new parameters, then you trigger this as part of the Tenant Service upgrade. Both the new database and the new values for the database parameters (specified in the **values.yaml** file used for the `helm upgrade` step) must be ready and available before you trigger the Tenant Service upgrade.

If you're performing an upgrade or maintenance to the Redis backend that is shared with the core Voice platform, and you require a change to the Redis connection parameters, first scale Tenant Service down to a single instance (setting the `replicaCount` parameter in the **values.yaml** file), then perform the Redis backend maintenance or upgrade, and finally, trigger a restart of the Tenant Service. After that's complete, you can scale up Tenant Service again.

Timing

A regular upgrade schedule is necessary to fit within the Genesys policy of supporting N-2 releases, but a particular release might warrant an earlier upgrade (for example, because of a critical security fix).

If the service you are upgrading requires a later version of any third-party services, upgrade the third-party service(s) before you upgrade the private edition service. For the latest supported versions of third-party services, see the Software requirements page in the suite-level guide.

Scheduling considerations

Genesys recommends that you upgrade the services methodically and sequentially: Complete the upgrade for one service and verify that it upgraded successfully before proceeding to upgrade the next service. If necessary, roll back the upgrade and verify successful rollback.

Monitoring

Monitor the upgrade process using standard Kubernetes and Helm metrics, as well as service-specific metrics that can identify failure or successful completion of the upgrade (see Observability in Tenant Service).

Genesys recommends that you create custom alerts for key indicators of failure — for example, an alert that a pod is in pending state for longer than a timeout suitable for your environment. Consider including an alert for the absence of metrics, which is a situation that can occur if the Docker image is not available. Note that Genesys does not provide support for custom alerts that you create in your environment.

Preparatory steps

Ensure that your processes have been set up to enable easy rollback in case an upgrade leads to compatibility or other issues.

Each time you upgrade a service:

1. Review the release note to identify changes.
2. Ensure that the new package is available for you to deploy in your environment.
3. Ensure that your existing **-values.yaml** file is available and update it if required to implement changes.

Rolling Update

Rolling Update: Upgrade

Execute the following command to upgrade :

```
helm upgrade --install  -f -values.yaml  -n
```

Tip: If your review of Helm chart changes (see Preparatory Step 3) identifies that the only update you need to make to your existing **-values.yaml** file is to update the image version, you can pass the image tag as an argument by using the `--set` flag in the command:

```
helm upgrade --install  -f -values.yaml  --set .image.tag=
```

When you perform an upgrade to Tenant Service, Tenant Service provisioning is done automatically, as part of the `helm upgrade` step. There is no need to make any provisioning changes, unless noted in Release Notes for a particular version.

Rolling Update: Verify the upgrade

Follow usual Kubernetes best practices to verify that the new service version is deployed. See the information about initial deployment for additional functional validation that the service has upgraded successfully.

Rolling Update: Rollback

Execute the following command to roll back the upgrade to the previous version:

```
helm rollback
```

or, to roll back to an even earlier version:

```
helm rollback
```

Alternatively, you can re-install the previous package:

1. Revert the image version in the `.image.tag` parameter in the **-values.yaml** file. If applicable, also revert any configuration changes you implemented for the new release.
2. Execute the following command to roll back the upgrade:

   ```
   helm upgrade --install  -f -values.yaml
   ```

   Tip: You can also directly pass the image tag as an argument by using the `--set` flag in the command:

   ```
   helm upgrade --install  -f -values.yaml  --set .image.tag=
   ```

Rolling Update: Verify the rollback

Verify the rollback in the same way that you verified the upgrade (see Rolling Update: Verify the upgrade).

Uninstall

> ### Warning
> Uninstalling a service removes all Kubernetes resources associated with that service. Genesys recommends that you contact Genesys Customer Care before uninstalling any private edition services, particularly in a production environment, to ensure that you understand the implications and to prevent unintended consequences arising from, say, unrecognized dependencies or purged data.

Execute the following command to uninstall :

```
helm uninstall  -n
```

Observability in Tenant Service

Contents

- 1 Monitoring
  - 1.1 Enable monitoring
  - 1.2 Configure metrics
- 2 Alerting
  - 2.1 Configure alerts
- 3 Logging

Learn about the logs, metrics, and alerts you should monitor for Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Monitoring

Private edition services expose metrics that can be scraped by Prometheus, to support monitoring operations and alerting.

- As described on Monitoring overview and approach, you can use a tool like Grafana to create dashboards that query the Prometheus metrics to visualize operational status.
- As described on Customizing Alertmanager configuration, you can configure Alertmanager to send notifications to notification providers such as PagerDuty, to notify you when an alert is triggered because a metric has exceeded a defined threshold.

The services expose a number of Genesys-defined and third-party metrics. The metrics that are defined in third-party software used by private edition services are available for you to use as long as the third-party provider still supports them. For descriptions of available Tenant Service metrics, see:

- Tenant Service metrics

See also System metrics.

Tenant Service supports various metrics in Prometheus format, exposed via a dedicated endpoint.

Enable monitoring

Tenant Service deployment supports the creation of a Prometheus monitor resource in order to scrape the monitoring endpoint; this feature is disabled by default. For information about the parameters that enable a Prometheus monitoring resource, see Monitoring and observability.

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
Tenant Service	PodMonitor	15000	/metrics	30 seconds

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
			(http://:15000/metrics)	(Applicable for any metric(s) that Tenant Service exposes. The update interval is not a property of the metric; it is a property of the optional PodMonitor that you can create.)

Configure metrics

The metrics that are exposed by the Tenant Service are available by default. No further configuration is required in order to define or expose these metrics. You cannot define your own custom metrics.

The Metrics pages linked to above show some of the metrics the Tenant Service exposes. You can also query Prometheus directly or via a dashboard to see all the metrics available from the Tenant Service.

Alerting

Private edition services define a number of alerts based on Prometheus metrics thresholds.

> **Important**
> You can use general third-party functionality to create rules to trigger alerts based on metrics values you specify. Genesys does not provide support for custom alerts that you create in your environment.

For descriptions of available Tenant Service alerts, see:

- Tenant Service alerts

Configure alerts

Private edition services define a number of alerts by default (for Tenant Service, see the pages linked to above). No further configuration is required.

The alerts are defined as **PrometheusRule** objects in a **prometheus-rule.yaml** file in the Helm charts. As described above, Tenant Service does not support customizing the alerts or defining additional **PrometheusRule** objects to create alerts based on the service-provided metrics.

Logging

The Tenant Service Helm Chart **values.yaml** files include the following configurable log volume options:

- Persistent Volume Claim (PVC) with RWX storage: This creates a shared RWX volume in tenant-monitor; this volume is then claimed across all pods of all tenants. For more information, see tenant-monitor and Samples and references.
- Persistent Volume (PV)/PVC with RWO storage for logging from individual Tenant pods: Mount the PVC from the Tenant StatefulSet volume claim template, instead of using tenant-monitor. For more information about Statefulsets, see Statefulsets in the Kubernetes documentation.
  For more information about the Tenant Service RWO PV/PVC log volume configuration, see tenant-monitor and Samples and references.
- Ephemeral volume (emptyDir) with a Fluent Bit logging sidecar that tails log files and sends them to standard output (stdout). The optional log forwarding to stdout is disabled by default. To enable the log forwarding option, see tenant-monitor.
  For general information about forwarding logs from internal components to stdout, see Sidecar processed logging in the *Genesys Multicloud CX Private Edition Operations guide*.

Tenant Service metrics and alerts

Contents

- 1 Metrics
- 2 Alerts

Find the metrics Tenant Service exposes and the alerts defined for Tenant Service.

Related documentation:

-
-
-
-
-

RSS:

- For private edition

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval
Tenant Service	PodMonitor	15000	/metrics (http://:15000/metrics)	30 seconds (Applicable for any metric(s) that Tenant Service exposes. The update interval is not a property of the metric; it is a property of the optional PodMonitor that you can create.)

See details about:

- Tenant Service metrics
- Tenant Service alerts

Metrics

You can query Prometheus directly to see all the metrics that the Tenant Service exposes. The following metrics are likely to be particularly useful. Genesys does not commit to maintain other currently available Tenant Service metrics not documented on this page.

Metric and description	Metric details	Indicator of
tenant_service_health_level	**Unit:** N/A	Health

Metric and description	Metric details	Indicator of
Health level of the tenant node. Values are -1 (fail), 0 (starting), 1 (degraded), 2 (pass). When the value is 2, the tenant Tenant Service node is fully functional. When the value is 1, the tenant might have issues with some of its internal functions and external dependencies, but is still capable of providing some services. When a value of 1 is reported, additional investigation is needed, via tenant logs, to troubleshoot and recover. A value of 0 or -1 indicates an inoperable node, either pending start or it has failed.	**Type:** gauge **Label:** **Sample value:** 2	

Alerts

If you enable a Tenant PodMonitor to expose the Tenant health metric, then you can create a basic alert rule for the Tenant Service using a template like the following:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: "custom-tenant-alert-rules"
spec:
        - alert: HealthFailFor5min
          expr: (max by (tenant) (tenant_service_health_level{namespace="",pod=~""}))
```

Enter your values where there are placeholders in the preceding template; the placeholders are:

-
-

Values are based on how you deployed tenant(s); in other words, what you used for override values.

No alerts are defined for Tenant Service.