Genesys Callback Private Edition Guide

2/9/2026

Table of Contents

Contents

Find links to all the topics in this guide.

Related documentation:

-
-
-

RSS:

- For private edition

Genesys Engagement Service (GES) provides callback and click-to-call-in services, which are available with the Genesys Multicloud CX private edition offering.

Overview

Learn more about GES and Genesys Callback, its architecture, and how to support high availability and disaster recovery.

- About Genesys Engagement Service/Callback
- Architecture
- High availability and disaster recovery

Configure and deploy

Find out how to configure and deploy GES.

- Before you begin
- Configure Genesys Engagement Service
- Deploy Genesys Engagement Service
- Provision Genesys Engagement Service
- Provision an API key for GES

Upgrade, roll back, or uninstall

Find out how to upgrade, roll back, or uninstall Genesys Engagement Service.

- Upgrade, roll back, or uninstall Genesys Engagement Service

Operations

Learn how to monitor GES and the callback services with metrics and logging.

- Observability in Genesys Callback
- Callback metrics and alerts

About Genesys Engagement Service/Callback

Contents

Learn about Genesys Engagement Service/Callback and how it works in Genesys Multicloud CX private edition.

Related documentation:

-
-
-
-

RSS:

- For private edition

Welcome to the *Genesys Callback Private Edition Guide*. This document explains the provisioning, deployment, configuration, and start procedures for Genesys Callback. The microservice that provides the callback functionality is called Genesys Engagement Service. Because this guide covers the deployment of the service, Genesys Engagement Service and GES terminology is used in much of the descriptive text and in any sample commands.

Supported Kubernetes platforms

Genesys Engagement Service (GES) is supported on the following Kubernetes platforms:

- Google Kubernetes Engine (GKE)
- Azure Kubernetes Service (AKS)

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

General information about Genesys Callback

Genesys offers classic callback services, allowing consumers to select either a callback as soon as an agent is available who has skills that match the caller's needs or, alternatively, to schedule the callback for a specific day and time that is convenient. In addition, Genesys offers robust and feature-rich callback services so you can use Push Notifications and CAPTCHA widgets with your callback offering. You can monitor and manage your callback services in a UI, which includes a view of your callback queues. Key components of the consumer's app or web journey can be preserved for agent or reporting use.

On top of the traditional callback services and scenarios, Genesys also offers the Click-To-Call feature, which lets consumers call your contact center by simply tapping a button in your mobile app.

If you are new to GES/Callback and plan to use Callback on the hosted Genesys Multicloud CX

platform, see Provisioning Callback in Designer.

After you have completed Callback provisioning and testing to ensure that calls are routed correctly, and Callback users have been assigned to the correct roles and access groups, you can begin to use the Callback UI. The **Callback** tab displays the list of callback records. Users with sufficient permissions use the **Callback** tab to manage the callback records, including creating, editing, or cancelling callbacks.

Callback Administrators and Developers have access to a **Developer** tab in the Callback UI. Use the **Developer** tab to manage callback activity and features at a more technical level. For example, you can check for errors in Callback API queries or validate API keys. To learn more about the tools on the **Developer** tab, see Troubleshooting and validating functionality.

For information about the Callback user interface (UI), see Genesys Callback Administrator's Guide.

For Release Notes, see Callback Release Notes.

Genesys Callback provides the following callback scenarios when deployed in a Kubernetes cluster:

- Immediate callback
- Scheduled callback
- User-originated Click-To-Call-In

Callback supports integration with Push Notifications and CAPTCHA widgets.

Genesys Callback is enabled by Genesys Engagement Services REST APIs. The Callback APIs are:

- Callbacks: Create, retrieve, cancel callbacks.
- Estimated Wait Time: Retrieve Estimated Wait Time.
- Availability: Retrieve time slots for a callback, matching Office Hours.
- Call In: Request the phone number to call in.
- Queue Status: Retrieve information about a queue's readiness to accept callbacks.
- Statistics: Provides a proxy to the GWS Statistics API. To use the Callback Statistics API, you must first register your GWS credentials in the Callback UI (**Developer** > **Credential Management** > **GWS Credentials** tab).

Callback reporting

You can generate both real-time and historical reports for callbacks. For real-time reporting, you require Genesys Pulse. For historical reporting, you require Genesys Customer Experience Insights (GCXI). For more information about callback reporting, see Callback reports.

Architecture

Contents

Learn about Genesys Engagement Service architecture

Related documentation:

-
-
-
-

RSS:

- For private edition

Introduction

For more information about GES in relation to the Voice Microservices, including the Tenant Service, also see the Voice Microservices Private Edition Guide and the Tenant Service Private Edition Guide.

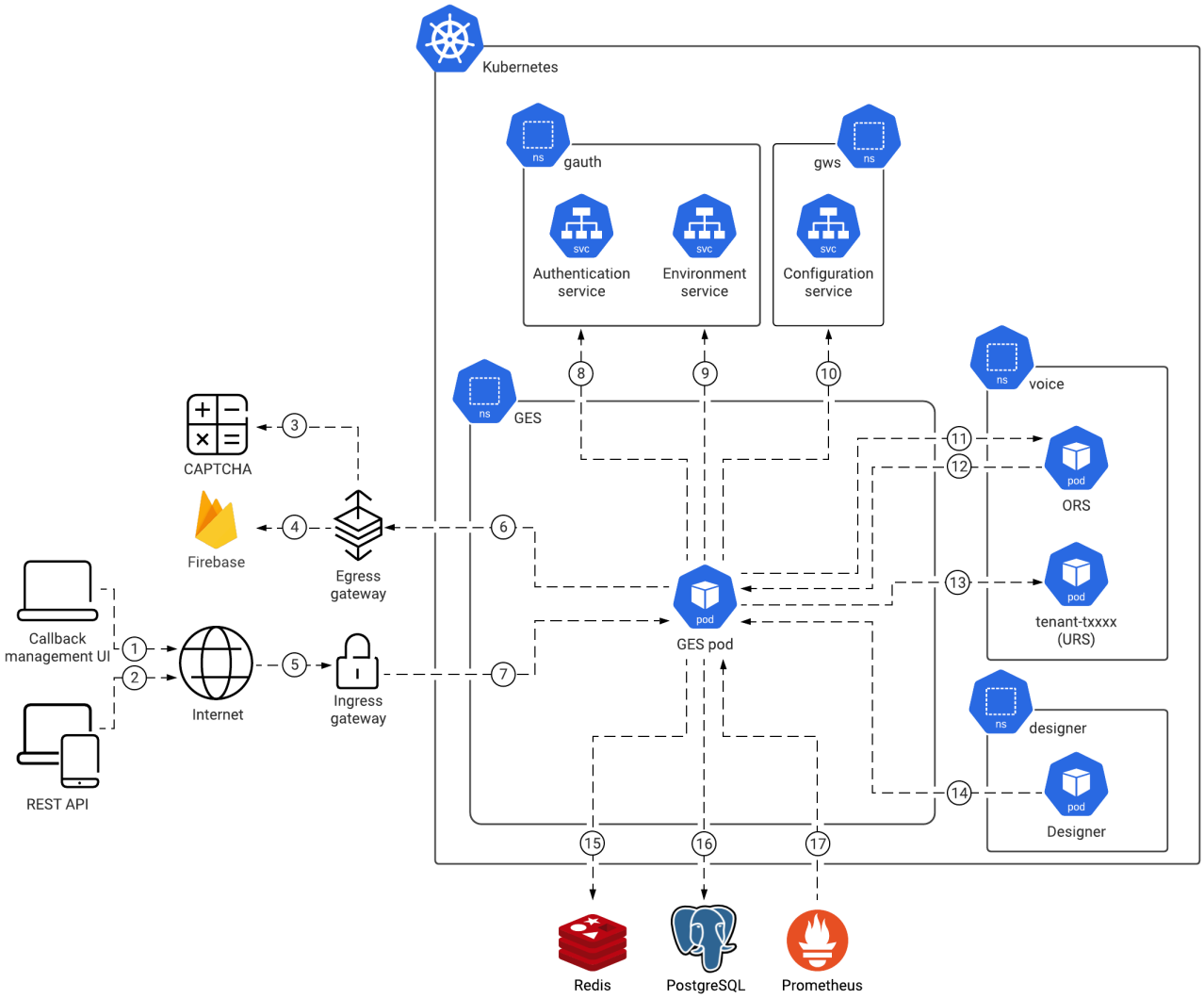
The following diagram shows the Genesys Engagement Service (GES) architecture. There must be at least two GES nodes spread across availability zones, forming a single service for load balancing and high availability.

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 Genesys Engagement 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 Genesys Engagement Service as a service in the network. *Egress* means the Genesys Engagement Service service is the source, and *Ingress* means the Genesys Engagement 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	Callback	Internet	HTTPS	443	Ingress	GES serves the files for rendering the UI front end and answers UI-specific API requests, such as gathering data to populate the Callback view, for authenticated users only.
2	External REST API	Internet	HTTPS	443	Ingress	The external REST API allows users to: • Create, retrieve, and cancel callbacks; • Query the office hours and capacity of callback services; • Retrieve estimated wait time of all virtual queues; • Create a call-in request; • Query a virtual queue's readiness for callbacks;

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
						• Retrieve statistics from Genesys Web Services and Applications. For more information about which APIs are available, see the Genesys Multicloud API Reference for the Engagement API.
3	Genesys Engagement Service	Outbound API requests to third-party services; in this case, Google reCAPTCHA.	HTTPS	443	Egress	reCAPTCHA verifies that the caller of the Callback Create API (on connection #2) is a real person. This is an optional, additional safety feature for fraud prevention.
4	Genesys Engagement Service	Outbound API requests to third-party services; in this case, Google Firebase Cloud Messaging (FCM).	HTTPS	443	Egress	FCM sends web or mobile push notifications to customers. GES uses FCM for Click-to-Call-In and callback mobile notifications.

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
5	UI or REST API	Ingress gateway	HTTPS	443	Ingress	Callback management UI and REST API data. Also see connections #1 and #2, above.
6	Genesys Engagement Service	Egress gateway	HTTPS	443	Egress	Outgoing Push Notifications and Captcha requests. Also see connections #3 and #4, above.
7	Ingress gateway	Genesys Engagement Service	HTTP	3050	Ingress	Incoming data from the UI or REST API.
8	Genesys Engagement Service	Genesys Authentication	HTTP	8095	Intra-cluster	GES queries the Genesys Authentication Service to validate a UI user's identity.
9	Genesys Engagement Service	Genesys Authentication	HTTP	8091	Intra-cluster	GES queries the Environment Service to obtain the tenant's configuration.
10	Genesys Engagement Service	Genesys Web Services and Applications	HTTP	8092	Intra-cluster	GES queries the GWS Configuration Service to obtain privileges and permissions for the authenticated user.
11	Genesys Engagement Service	Voice Microservices	HTTP	9098	Intra-cluster	GES starts a session in ORS when it is time to

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
						put the callback in the queue for an agent. To initiate the ORS session, GES stores an entry in the Voice Microservice's Redis (using port 6379), rather than communicating directly with ORS. Once the ORS session is started, GES regularly queries the ORS session (using port 9098) for diagnostics information about the callback. In addition, GES might send events to control the ORS session; for example, when the callback is cancelled through the API or UI.
12	Voice Microservices	Genesys Engagement Service	HTTP	3050	Intra-cluster	The callback ORS session updates the state and storage of the callback record in GES.
13	Genesys Engagement Service	Tenant Service	HTTP	5580	Intra-cluster	GES queries URS to obtain the

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
						estimated wait time of virtual queues.
14	Designer	Genesys Engagement Service	HTTP	3050	Intra-cluster	When the CALLBACK_SETTINGS data table is published in Designer, Designer sends the changed callback service configurations to GES.
15	Genesys Engagement Service	Redis	Redis	6379	Egress	GES uses Redis for in-memory data store for quick retrieval.
16	Genesys Engagement Service	PostgreSQL	Postgres	5432	Egress	GES uses Postgres as a persistent data store.
17	Prometheus	Genesys Engagement Service	HTTP	3050	Ingress	GES provides metrics for monitoring and alerting with Prometheus.

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

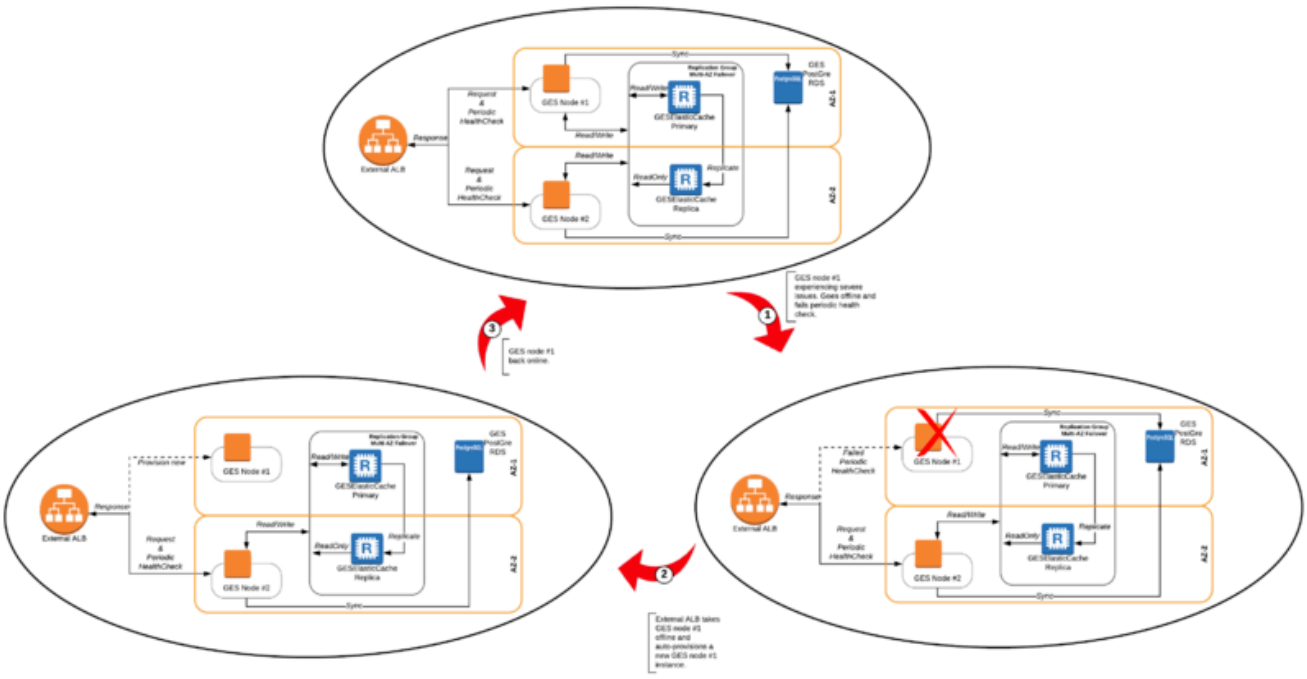
Service	High Availability	Disaster Recovery	Where can you host this service?
Genesys Engagement Service	N = N (N+1)	Not supported	Primary unit only

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

Genesys Engagement Service (GES) uses automated failure recovery mechanisms that allow GES to try to recover from critical failure scenarios such as a node failure or a partial or full Redis failure. The following sections illustrate how these failure recovery processes work.

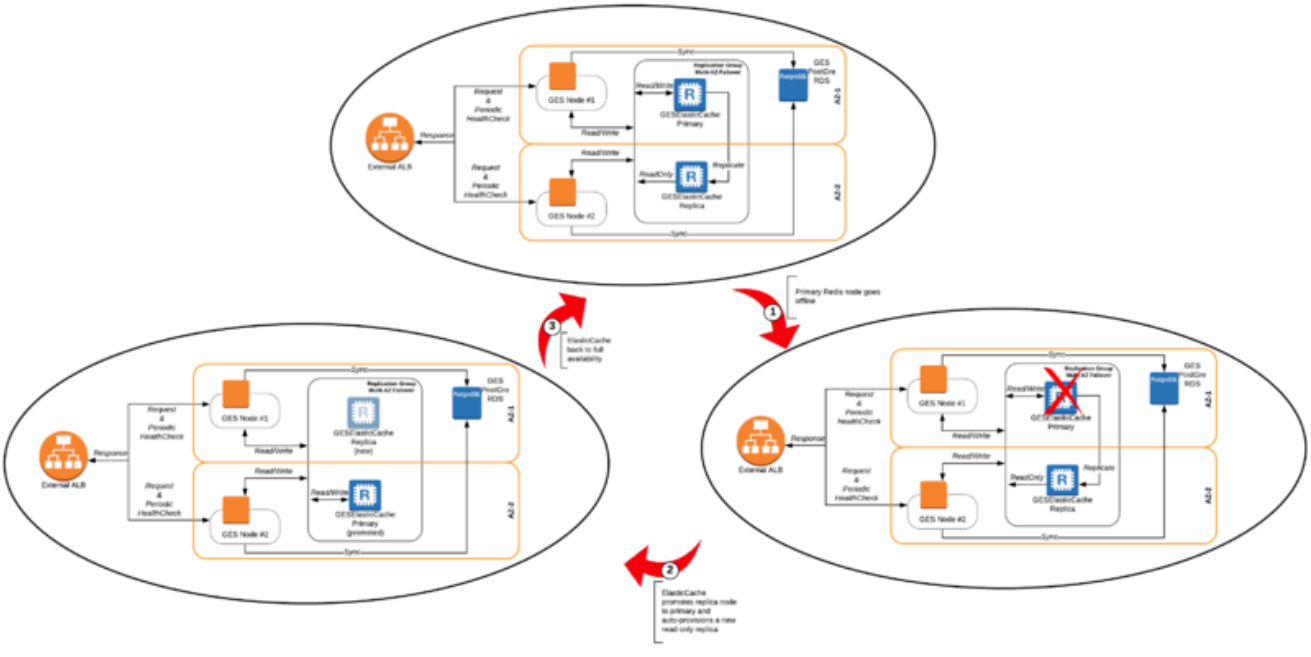
GES node failure

The following diagram depicts the automated recovery procedures when one of the GES nodes goes offline.



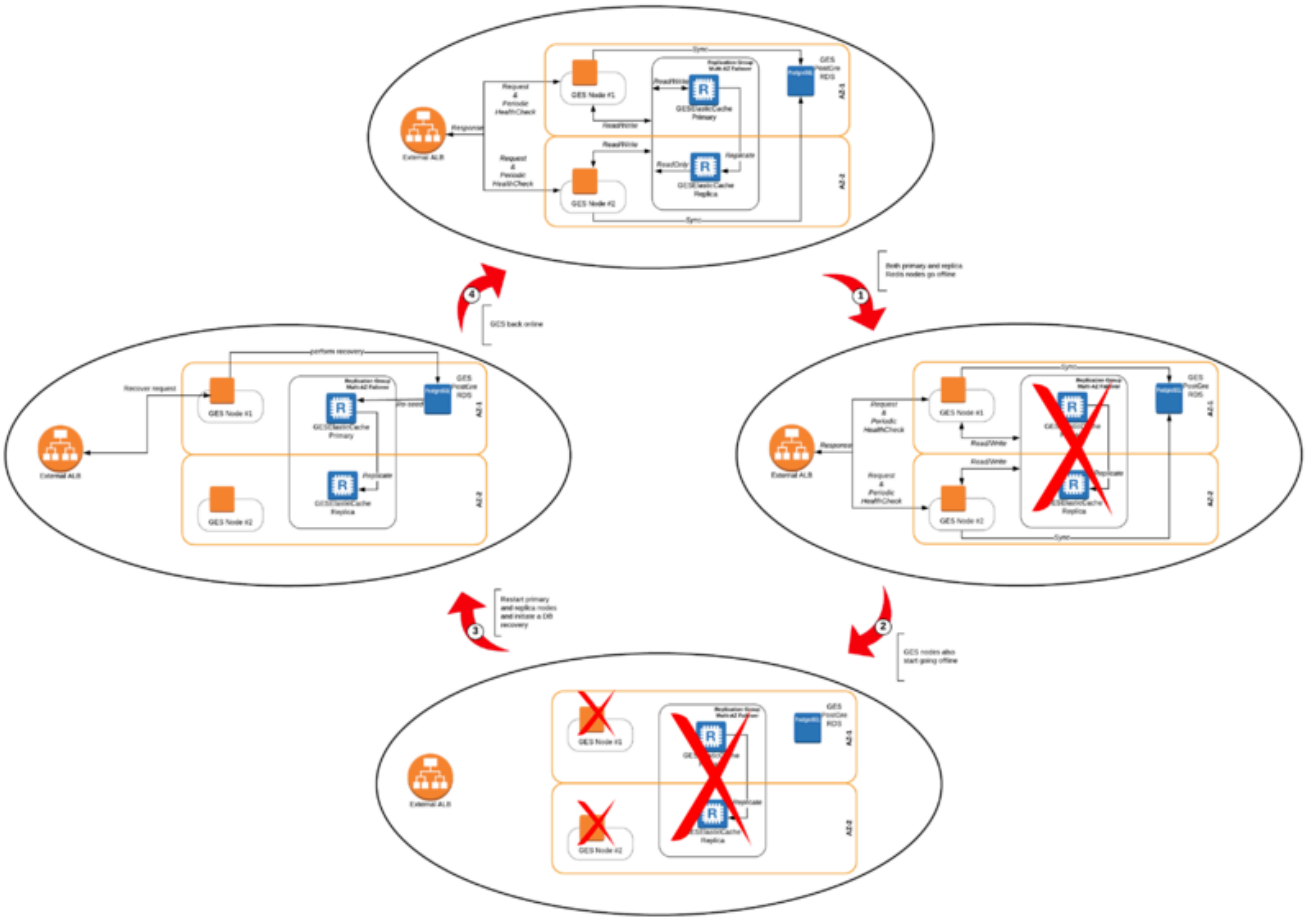
Partial Redis failure

The following diagram depicts the automated recovery procedures when the primary Redis node goes offline.



Full Redis failure

The following diagram depicts the automated recovery procedures when both the primary and replica Redis nodes go offline.



Before you begin

Contents

Find out what to do before deploying Genesys Callback.

Related documentation:

-
-
-
-

RSS:

- For private edition

Limitations and assumptions

Not applicable

Download the Helm charts

Genesys Engagement Service (GES) is the only service that runs in the GES Docker container. The Helm charts included with the GES release provision GES and any Kubernetes infrastructure necessary for GES to run, such as load balancing, autoscaling, ingress control, and monitoring integration.

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

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

Third-party prerequisites

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

Third-party services

Name	Version	Purpose	Notes
Redis	6.x	Used for caching. Only distributions of Redis that support Redis	You require the following information for GES deployment:

Name	Version	Purpose	Notes
		cluster mode are supported, however, some services may not support cluster mode.	• Redis hostname • Redis password (only required if you use a password) • Redis port You require access to the Voice Microservices deployment Redis as well (specifically the Voice Orchestration Service Redis stream). You require the following information about the Voice Microservices Redis for your GES deployment: • Redis hostname • Redis password (only required if you use a password) • Redis port
PostgreSQL	11.x	Relational database.	Before you deploy Genesys Engagement Service (GES), a table and user with database owner (DBO) permissions must be deployed and ready for use by GES. You require the following information for GES deployment: • Database name • Database port • Database hostname • Database user
Consul	1.13.x	Service discovery, service mesh, and key/value store.	

Storage requirements

The primary contributor to the size of a callback record is the amount of user data that is attached to a callback. Since this is an open-ended field, and the composition will differ from customer to customer, it is difficult to state the precise storage requirements of GES for a given deployment. To

assist you, the following table lists the results of testing done in an internal Genesys development environment and shows the impact that user data has when it comes to the storage requirements for both Redis and Postgres.

Test	Redis size	Postgres size (MB)
10,000 Scheduled Callbacks with no user data	26.51 MB	41.1 MB
10,000 Scheduled Callbacks with 10 KB of user data	64.44 MB	252.91 MB
10,000 Scheduled Callbacks with 100k of user data	110.58 MB	595.79 MB

Note: *This is 100k of randomized string in a single field in the user data.*

Hardware requirements

Genesys strongly recommends the following hardware requirements to run GES with a single tenant. The requirements are based on running GES in a multi-tenanted environment and scaled down accordingly. Use these guidelines, coupled with the callback storage information listed above, to gauge the precise requirements needed to ensure that GES runs smoothly in your deployment.

GES

(Based on t3.medium)

- vCPUs: 1
- Memory: 2 GiB
- Network burst: 5 Gbps

Redis

(Based on cache.r5.large) Redis is essential to GES service availability. Deploy two Redis caches in a cluster; the second cache acts as a replica of the first. For more information, see Architecture.

Callback data is stored in Redis memory.

- vCPUs: 1
- Memory: 8 GiB
- Network burst: 10 Gbps

PostgreSQL

(Based on db.t3.medium)

- vCPUs: 2
- Memory: 4 GiB
- Network burst: 5 Gbps

- Storage: 100 GiB

Sizing calculator

The information in this section is provided to help you determine what hardware you need to run GES and third-party components. The information and formulas are based on an analysis of database disk storage and Redis memory usage requirements for callback data. The numbers provided here include only storage and memory usage for callbacks. Additional storage and memory is required for configuration data and basic operations.

Requirements per callback

Each callback record (excluding user data) requires approximately 6.5 to 7.0 kB of database disk storage, plus additional disk storage for the user data. Each kB of user data consumes approximately 3.0 kB of disk storage.

Each callback record (excluding user data) requires approximately 4.5 to 5.5 kB of Redis memory, plus an additional 1.25 kB for each kB of user data.

Use the following formulas to estimate disk storage and Redis memory requirements:

- Estimate database disk storage requirements for callback data:
  number of callbacks per day> × (7 kB + (3 kB × kB of user data per callback>)) × 14 days
- Estimate Redis memory requirements for callback data:
  number of callbacks per day> × (5.5 kB + (1.25 kB × kB of user data per callback>)) × 14 days

For example, if a tenant has an average of 100,000 callbacks per day with 1kB user data in each callback:

- The database storage requirement is approximately 14 GB.
- The Redis memory requirement is approximately 9.5 GB.

NOTE: Each callback record is stored for 14 days. If you average about 10k scheduled callbacks every day, and the scheduled callbacks are all booked as far out as possible (that is, 14 days in the future), the number of callbacks to use in storage and memory calculations is 28 days × 10k callbacks per day = 280k callbacks.

Redis operations

The Redis operations primarily update the connectivity status to other services such as Tenant Service (specifically ORS and URS) and Genesys Web Services and Applications (GWS).

When GES is idle (zero callbacks in the past, no active callback sessions, no scheduled callbacks), GES generates about 50 Redis operations per second per GES node per tenant.

Each Immediate callback generates approximately 110 Redis operations from its creation to the end of the ORS session.

For Scheduled callbacks, assuming each callback generates 110 Redis operations when the ORS session is active (based on Immediate callback numbers), there is 1 additional Redis operation for each minute that a callback is scheduled.

For example, if a callback is scheduled for 1 hour from the time it was created, the number of Redis operations is approximately 60 + 110 = 170.

For a callback scheduled for 1 day from the time it was created, it generates approximately 60 × 24 + 110 = 1550 Redis operations, using the following formula for the number of Redis operations per callback:
number of callbacks> × (110 + number of minutes until scheduled time>)

Because the longevity of a callback ORS session depends on the estimated wait time (EWT), the total number of Redis operations performed by GES per minute varies, based on both the number of callbacks in the system and the EWT of the callbacks.

Use the following formula to estimate the number of Redis operations performed per minute:
Total number of Redis operations per minute = (50 base GES Redis operations per second × 60 seconds) + number of upcoming scheduled callbacks in the system> + ((total number of active callbacks> / EWT>) × 110)

Where:

- Total number of active callbacks = number of active immediate callbacks> + number of active scheduled callbacks>, and
- Number of active scheduled callbacks = (number of scheduled callbacks per time slot> / time slot duration>) × EWT>

For example, let's say we have the following scenario:

- Scheduled callbacks:
  - Time slot duration = 15 minutes
  - Maximum capacity per time slot = 100
  - Business hours = 24x7
  - Assume that all time slots are fully booked for the next 14 days
- Number of active immediate callbacks = 1,000
- Estimated wait time = 90 minutes

Using the preceding formulas, estimate the Redis operations per minute:

- Total number of scheduled callbacks = (100 × (60 / 15)) × 24 × 14 = 134,400
- Number of active scheduled callbacks = (100 / 15) × 90 = 600
- Number of upcoming scheduled callbacks = total number of scheduled callbacks> - number of active scheduled callbacks> = (134,400 - 600) = 133,800
- Total number of active callbacks = 1,000 + 600 = 1,600
- Total number of Redis operations per minute = (50 × 60) + 133,800 + ((1,600 / 90) × 110) = 138,756

Redis keys

Each callback creates three additional Redis keys. Given the preceding calculations for Redis memory requirements for each callback, the formula for the average key size is:

(5.5 kB + (1.25 kB × kB_of_user_data_per_callback>)) / 3

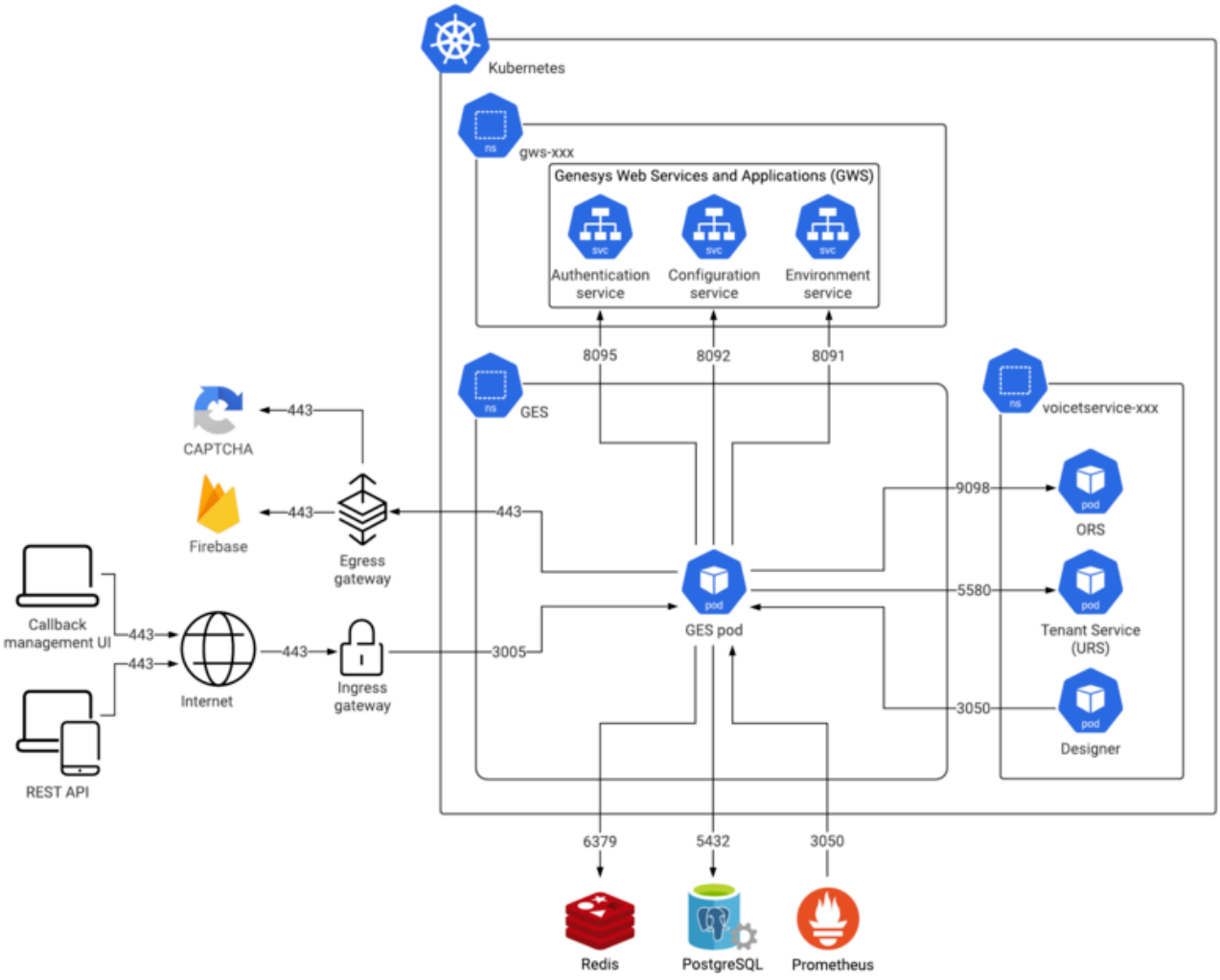
Network requirements

Incoming connections to the GES deployment are handled either through the UI or through the external API. For information about how to use the external API, see the Genesys Multicloud CX Developer Center.

Connection topology

The diagram below shows the incoming and outgoing connections amongst GES and other Genesys and third-party software such as Redis, PostgreSQL, and Prometheus. In the diagram, Prometheus is shown as being part of the broader Kubernetes deployment, although this is not a requirement. What's important is that Prometheus is able to reach the internal load balancer for GES.

The other important thing to note is that, depending on the use case, GES might communicate with Firebase and CAPTCHA over the open internet. This is not part of the default callback offering, but if you use Push Notifications with your callback service, then GES must be able to connect to Firebase over TLS. The use of Push Notifications or CAPTCHA is optional and not necessary for the basic callback scenarios.



Web application firewall rules

Information in the following sections is based on NGINX configuration used by GES in an Azure cloud environment.

Cookies and session requirements

When interacting with the UI, GES and GWS ensure that the user's browser has the appropriate session cookies. By default, UI sessions time out after 20 minutes of inactivity.

The external Engagement API does not require session management or the use of cookies, but it is important that the GES API key be provided in the request headers in the **X-API-Key** field.

For ingress to GES, allow requests to only the following paths to be forwarded to GES:

- /ges/
- /engagement/v3/callbacks/create

- /engagement/v3/callbacks/cancel
- /engagement/v3/callbacks/retrieve
- /engagement/v3/callbacks/availability/
- /engagement/v3/callbacks/queue-status/
- /engagement/v3/callbacks/open-for/
- /engagement/v3/estimated-wait-time
- /engagement/v3/call-in/requests/create
- /engagement/v3/statistics/operations/get-statistic-ex

In addition to allowing connections to only these paths, ensure that the `ccid` or `ContactCenterID` headers on any incoming requests are empty. This enhances security of the GES deployment; it prevents the use of external APIs by an actor who has only the CCID of the contact center.

TLS/SSL certificate configuration

There are no special TLS certificate requirements for the GES/Genesys Callback web-based UI.

Subnet requirements

There are no special requirements for sizing or creating an IP subnet for GES above and beyond the demands of the broader Kubernetes cluster.

Browser requirements

The Genesys Callback user interface is supported in the following browsers.

Browsers

Name	Version	Notes
Firefox	Current release or one version previous	Genesys also supports the current ESR release. Genesys supports the transitional ESR release only during the time period in which the new ESR release is tested and certified. For more information, see Firefox ESR release cycle. Firefox updates itself automatically. Versions of Firefox are only an issue if your IT department restricts automatic updates.
Chrome	Current release or one version previous	Chrome updates itself automatically. Versions of Chrome are only an issue if your IT department restricts automatic updates.
Microsoft Edge (Legacy)	Current release	
Microsoft Edge Chromium	Current release	

Genesys dependencies

GES has dependencies on several other Genesys services. You must deploy the services on which GES depends and verify that each is working as expected *before* you provision and configure GES. If you follow this advice, then – if any issues arise during the provisioning of GES – you can be reasonably assured that the fault lies in how GES is provisioned, rather than in a downstream program.

GES/Callback requires your environment to contain supported releases of the following Genesys services, which must be deployed before you deploy Callback:

- Genesys Web Services and Applications (GWS)
- Genesys Authentication
- Voice Microservices (includes Tenant Service)
- Designer
- Agent Setup

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

GDPR support

Callback records are stored for 14 days. The 14-day TTL setting starts at the Desired Callback Time. The `Callback TTL (seconds)` setting in the CALLBACK_SETTINGS data table has no effect on callback record storage duration; 14 days is a fixed value for all callback records.

Configure Genesys Engagement Service

Contents

Learn how to configure Genesys Callback.

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*.

Genesys Engagement Service (GES) has a Helm value layout similar to other Engage services, however the use of environment variables as the primary way to configure GES means that setting up the Helm values is particularly important. Below is a section-by-section description of the Helm chart parameters, what they influence, and any other information that you might find helpful to know when provisioning GES within Private Edition.

All values to configure GES are within the `.ges` subsection of the `.Values.yaml` file. For example, to set affinity, you reference `.Values.ges.affinity`. For those values listed in a `misc` section, omit that from the path when referencing an environment variable.

labels

Parameter	Description	Default	Valid values
globalLabels	Labels that are going to be applied to all resources within the GES helm deployment. This is required.	{}	YAML object
podLabels	Labels that will be applied to all GES pods. This is required.	{}	YAML object
serviceLabels	Labels that will be applied to just the GES service. This is required.	{}	YAML object

annotations

Parameter	Description	Default	Valid values
globalAnnotations	Annotations that are going to be applied to all resources within the GES helm deployment. This is required.	{}	YAML object
podAnnotations	Annotations that will be applied to all GES pods. This is required.	{}	YAML object

configMap

Set values in the config map. The values are divided into two sections:

- Environment variables that control the behavior of GES.
- Values supplied to GES that allow it to integrate with dependencies like Redis, Postgres, GWS, ORS, and so on.

env

log

Parameter	Description	Default	Valid values
level	Controls the level of logging output for GES.	DEBUG	(DEBUG, TRACE, ERROR, FATAL, INFO, WARN)
maxLen	The max length (in bytes) before GES log messages get auto-truncated.	2048	Number

server

Parameter	Description	Default	Valid values
internal_port	The port GES listens to for incoming requests from within Engage. Typically 3050. This is required.		Number
external_port	The port GES listens to for incoming messages to the external APIs (typically from the outside world). Typically		Number

Parameter	Description	Default	Valid values
	2050. This is required.		
internal_url	A default URL that would allow for ORS to query this instance of GES. The default value assumes GES is within the GES namespace. This is required.	ges.ges	String

misc

Parameter	Description	Default	Valid values
regionAffinity	Determines the ORS/URS nodes that will be given priority to handle requests for a given GES nodes. If all servers within the priority region are down, a secondary region will be used. This is required.		String
devopsAccessGroups	The list of roles that are to be granted the DevOps permission in GES. This is required.	Platform Read Only,Platform Provisioning,Platform Internal Administrator	String

integrations

redis

Parameter	Description	Default	Valid values
host	The host of GES' redis instance. This can also be provisioned using secrets. This is required.		String
port	The port the Redis instance listens on. This can also be provisioned using secrets. This is required.	6379	Number

Parameter	Description	Default	Valid values
secure	True if GES is to connect to Redis over TLS. This can also be provisioned using secrets. This is required.	false	Boolean

db

Parameter	Description	Default	Valid values
host	The host of GES' Postgres instance. This can also be provisioned using secrets. This is required.		String
name	The name of the DB provisioned for GES. This can also be provisioned using secrets. This is required.		string
secure	True if GES is to connect to Postgres over TLS. This can also be provisioned using secrets. This is required.		Boolean

gws

Parameter	Description	Default	Valid values
auth	The internal-facing hostname/port for GWS auth service. This is required.		String
env	The hostname/port for GWS env service. This is required.		String
conf	The hostname/port for GWS config service. This is required.		String
public_auth	The public-facing hostname/port for GWS auth service.		String

Parameter	Description	Default	Valid values
	This is required.		

vmcs

Parameter	Description	Default	Valid values
redis_host	The hostname of the Voice Microservices Redis instance. This can also be supplied through the use of secrets, but if supplied through secrets, it will override the value supplied in the map. This is required.		String
redis_port	The port of the Voice Microservices Redis instance. This can also be supplied through the use of secrets, but if supplied through secrets, it will override the value supplied in the map. This is required.		String
redis_cluster_mode	True if Voice Microservices is running in cluster mode. Otherwise, False. This is required.		String
redis_stream_name	The Redis stream that GES should connect to to write information about new callbacks. This is required.	"NewCallbackStream"	String
ors_redis_location	A compound value made from defining the Redis host/port. Leave as the default value. This can also be supplied through the use of secrets, but if supplied through secrets, it will override the value supplied in the config map. This is required.	"{{ .Values.ges.configMap.integrations.vmcs.redis_host }}:{{ .Values.ges.configMap.integrations.vmcs.redis_port | default 10000 }}"	

deployment

Parameter	Description	Default	Valid values
enableServiceLinks	Controls whether service information is added to environment variables or not. When the value is `true`, a set of environment variables for each active service is added to the pod.	false	Boolean

image

Parameter	Description	Default	Valid values
imagePullSecrets	A secret needed to authenticate with the docker registry. Might not be required; implementation is based on how you have set up your cloud. For example, in Genesys CX on Azure, the Pod ID is used to authenticate with the Azure Container Registry.	[ ]	List
imagePullPolicy	How often the pod will try to pull a fresh image on start up. This is required.	Always	String
registry	The docker registry that the GES image is to be pulled from. This is required.		String
image	The name of the docker image for GES. This is required.	genesys/ges	String

hpa

Parameter	Description	Default	Valid values
enabled	True when horizontal pod auto-scaling is enabled. Might not be required; implementation is based on how you have set up your cloud. For example, in Genesys CX on Azure, the Pod ID is used to authenticate with the Azure Container Registry.	True	Boolean

Parameter	Description	Default	Valid values
maximumReplicas	Maximum number of GES pods that will be running in a deployment.	5	String
minimumReplicas	Minimum number of pods that will run in a deployment.	1	String
targetMetrics	The metrics that the Kubernetes controller will consult when evaluating if additional GES pods are needed. For more information, see the Kubernetes documentation.		String
behavior	Defines the behavior when extra pods for GES are spun up and down, including how many pods should be added or removed at a time and how long the deployment is given to stabilize before spinning up or winding down extra pods. For more information, see the Kubernetes documentation.		

misc

Parameter	Description	Default	Valid values
env	A list of environment variables to be defined within the pod. These can be defined using simple key value pairs or using secrets as described in the Kubernetes documentation.	[ ]	List
envFrom	A list of config maps from which the GES pod will get the environment. Be sure to include all config maps and other entries from which you wish to pull environment variables. For more information, see the Kubernetes documentation.		List

Parameter	Description	Default	Valid values
min_ready_seconds	The amount of seconds that a pod must be in the ready state before the Kube controller begins routing traffic to it.	300	String
dnsConfig	Values related to the configuration of DNS resolution. For more information, see the Kubernetes documentation.		Object
replicas	The number of replica GES deployments that are part of the cluster by default. This can be scaled up or down. For more information, see the Kubernetes documentation.	1	Number
nodeSelector	Contains any and all values that might aid the Kubernetes controller to determine what kind of cluster GES is to be scheduled on. For more information, see the Kubernetes documentation.		Object

resources

Parameter	Description	Default	Valid values
limits	Defines the limits of the resource requests that a GES pod can make from the cluster as well as the maximum amount to allot. For more information, see the Kubernetes documentation.	cpu: 1250m memory: 2048Mi	Object
requests	Defines the size of CPU/RAM requests a GES pod can make from the cluster as well as the maximum amount to allot.	cpu: 75m memory: 512Mi	Object

Parameter	Description	Default	Valid values
	For more information, see the Kubernetes documentation.		

ingress

ingressint

Parameter	Description	Default	Valid values
enabled	True when you want a shared app gateway to govern incoming connections to GES. This is required.	false	Boolean
annotations	Any annotations that are to be applied to the ingress object that governs how GES communicates with a shared App Gateway.		
hosts	The host name to reach GES from a shared app gateway.		
paths	The set of paths that the app gateway will forward to GES. Omitting these paths could cause the external APIs to break.	- /ges/ - /engagement/v3/callbacks/create - /engagement/v3/callbacks/cancel - /engagement/v3/callbacks/retrieve - /engagement/v3/callbacks/availability/ - /engagement/v3/callbacks/queue-status/ - /engagement/v3/callbacks/open-for/ - /engagement/v3/estimated-wait-time - /engagement/v3/call-in/requests/create - /engagement/v3/statistics/operations/get-statistic-ex	List
tls	Details of the TLS certificates for incoming connections to GES.		

monitoring

prometheus

Parameter	Description	Default	Valid values
use_service_monitor	True when you use a Service Monitor as the mechanism for service/pod discovery by the Prometheus framework. For more information, see Simple Management of Prometheus Monitoring Pipeline with the Prometheus Operator.		boolean

grafana

Parameter	Description	Default	Valid values
enabled	True if Grafana dashboards will be deloyed from config maps defined in the Helm Chart.	False	Boolean

podDisruptionBudget

Parameter	Description	Default	Valid values
enabled	True when a PodDisruptionBudget is defined for GES.	True	Boolean
strategy	The strategy used to implement the pod disruption budget. For more information, see the Kubernetes documentation.		

secrets

The **secrets** section defines how different types of secrets are made available to the GES deployment through the use of volume mounts. Using this method is not strictly necessary, as it is possible to provide the same configuration information using Environment variables. At a high level, the kind of data that is made available through the use of secrets includes Redis/ORS Redis hosts and credentials, GWS client information as well as the DB credentials. For a more thorough explanation and examples of how GES can leverage the secrets capabilities, see Define secrets. For a detailed

look at the capabilities of Kubernetes in general, see the Kubernetes documentation on secrets and the Kubernetes documentation on volumes and volume mounts, which GES uses to actually deliver the secrets.

volumeMounts

Parameter	Description	Default	Valid values	Notes
enabled	True when volume mounts are used for the deployment. This is required.	false	Boolean	Required
list	A list of volume mounts to be provisioned for the GES deployment.			

volumes

Parameter	Description	Default	Valid values
enabled	True when volumes are used for the deployment. This is required.	false	Boolean
list	A list of volumes to be provided for the GES deployment.		

misc

Parameter	Description	Default	Valid values
config_folders	The list of folders that GES uses to read configuration information. These folders are the volumes mounted by the secrets infrastructure. For more information, see Configure security.	""	String

service

Parameter	Description	Default	Valid values
port_external	The port on which the service listens for external connections from outside the Genesys Multicloud CX solution.	80	Number

Parameter	Description	Default	Valid values
	This is required. For more information, see the Kubernetes documentation.		
port_internal	The port on which the service listens for connections from inside the Genesys Multicloud CX solution. This is required. For more information, see the Kubernetes documentation.	8080	Number
type	Defines whether or not (and how) a service is exposed to the outside world. This is required. For more information, see the Kubernetes documentation.	ClusterIP	See the Kubernetes documentation.

misc

Parameter	Description	Default	Valid values
serviceName	The service name you want to attach to a GES deployment. Only used in the Helm charts to help determine the name attached to the deployment.	ges	String
tolerations	Defines the tolerations to be applied to the GES pods. Pods with matching taints will be favored for scheduling. For more information, see the Kubernetes documentation.	{}	Object
affinity	Defines the pod affinity behavior used to determine which nodes the GES pods are scheduled on. For more information, see the Kubernetes documentation.	{}	

Configure Kubernetes

For information, see the following resources:

- the configMap section in the Helm values description
- the secrets section in the Helm values description
- Security
- Deploy the service

Configure security

In addition to supporting configuration through the supply of environment variables, GES supports configuration information being supplied through Kubernetes' secrets, which are subsequently mounted as volumes on the Kubernetes pod. This "secrets" mechanism is required when providing sensitive information to GES, such as usernames and passwords for connections to Redis, Postgres, GWS, and others. However, you can also use secrets to supply less-sensitive information to GES.

To successfully acquire the GES docker image, deploy a secret in the GES namespace that provides Kubernetes with necessary credentials to authenticate to the JFrog artifactory.

This section provides information about the security context settings and a general overview of how to enable secrets in the Helm charts through the use of volume mounts and includes a detailed look at how to integrate with other components using secrets.

Security context configuration

The security context settings define the privilege and access control settings for pods and containers. For more information, see the Kubernetes documentation.

Configuring a JFrog secret

For a full description of downloading Genesys Multicloud CX containers and accessing repositories, see Downloading your Genesys Multicloud CX containers in *Setting up Genesys Multicloud CX Private Edition*.

If you're creating a secret containing JFrog account details, the secret must be of type `kubernetes/dockerconfigjson`. You can find more information on this topic in the Kubernetes documentation. Regardless of platform, you create the secret using the following `kubectl` command:

```
kubectl create secret docker-registry  \
  --docker-server= \
  --docker-username= \
  --docker-password= \
  --docker-email=
```

Updating the Values.yaml file

In your **Values.yaml** file, update the value of `.Values.ges.deployment.image.imagePullSecrets`. For example:
`imagePullSecrets: - name:`

Providing secrets to GES

At present, while it is possible to supply confidential values to GES through the facility in Kubernetes where secrets can be read into Pod environment variables, this is not always the case. Genesys strongly recommends that you make confidential information such as database usernames and passwords, GWS client information, and Redis keys available to GES through volume mounts. Volume mounts are described in this section, but for more information, see the Kubernetes documentation.

Secrets through volumes

GES incorporates secrets through the use of Volumes that are mounted to the GES Kubernetes pod. There are a number of ways in which you can define the secrets in the Kubernetes deployment, including manually defining secrets that are deployed along with the GES template, using tools such as Terraform, and so on. Instructions about how to leverage the different tools is outside the scope of this documentation. What's important is that you turn the secret into a volume.

To enable volumes, set the value `.Values.ges.secrets.volumes.enabled` to **true**, and then – for each secret that you want to make available – fill out a list entry. For example:

```
volumes:
  enabled: true
  list:
  - name:
    secret:
      secretName:  # This is the name of the secret within the Kubernetes Cluster
```

You can then mount the volume to GES. Set `.Values.ges.secrets.volumeMounts.enabled` to **true** and fill out a list entry. For example:

```
volumeMounts:
  enabled: true
  list:
    - mountPath:  # Best practise is just / but this can be different if requirements dictate
      name:
      readOnly: true
```

Finally, you must specify the folders that contain the configuration information for GES. To do this, enter the list of s in the form of a comma-separated string as the value for `.Values.ges.secrets.configFolders`.

While the examples here demonstrate how to provide secrets as typical Kubernetes secrets, GES also supports mounting secrets using the Container Storage Interface. Provided the CSI is configured correctly in the Kubernetes environment, you should be able to set this up with only some minor changes in the Helmvalues files.

Using secrets to integrate with other software

GES primarily uses secrets to create a secure mechanism through which it can obtain the credentials necessary to authenticate with services like Redis, GWS, the Postgres database, and the Voice Microservices Redis. This section provides information about how to accomplish this integration; this information is based on deployments on Azure.

REDIS

The REDIS-CACHEKEY must be supplied to GES through the use of an opaque secret. This is the password used to connect to the GES Redis instance. For this secret, the key is REDIS-CACHEKEY and the value is the password. You can provision this on its own or along with other secrets such as the database access information.

DB

The DB-USER and DB-PASSWORD must be supplied to GES through the use of an opaque secret. This is the username and password for GES to connect to the Postgres database. You can provision this on its own or along with other secrets such as the REDIS-CACHEKEY.

GWS

The AUTHENTICATION-CLIENT-ID and AUTHENTICATION-CLIENT-PASSWORD needed to authenticate with GWS must be supplied as an opaque secret. GES supports both encrypted and unencrypted client grants. However, if using an unencrypted grant, it is important that you encode the AUTHENTICATION-CLIENT-PASSWORD using the aes-128-ecb algorithm and an encryption key supplied by the password defined in the Helm value `.Values.ges.configMap.env.decrypt_password`.

ORS

The password to the ORS Redis instance must be supplied to GES through the use of an opaque secret. Unlike other secrets, which can be set up as simple key-value pairs, ORS REDIS information must be supplied to GES in a key-value pair where the key is `voice-redis-ors-stream` and the value is a JSON string with the following configuration:

```
{
    "password":"" # Optional,
    "rejectUnauthorized":"true" # Required,
    "servername":" # Optional
}
```

While this is the required method to supply the ORS REDIS password, other configuration information can be supplied as well. If configuration information is present, it will override any configuration supplied through the Config Maps.

Deploy Genesys Engagement Service

Contents

Learn how to deploy Genesys Engagement 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 information about downloading containers, see Downloading your Genesys Multicloud CX containers.

> **Important**
> Before deploying Genesys Engagement Service, review the Before you begin page for the full list of prerequisites.

Before you begin

Before you begin deployment, collect the following information – it is used during deployment:

- External GWS URL
- Internal GWS URL

- GES Redis host
- ORS Redis host
- GES internal URL
- GES external URL
- Postgres host

Deploy the service

1. Because Genesys Engagement Service (GES) relies on other services, you must ensure that the following have been deployed and configured successfully before you deploy GES:
   - Voice Microservices
   - Designer
   - GWS
   - Agent Setup (you require Agent Setup to create user logins for GES and to assign the necessary privileges)
2. Create dedicated Redis and Postgres instances for GES. Note the hostname and the credentials to authenticate and then provision Kubernetes secrets to store the following:
   - REDIS-CACHEKEY
   - DB-USER
   - DB-PASSWORD
3. Create the GES project.
4. Provision a set of GWS client credentials for GES.

   You need the Tenant ID for this step. You can use a `curl` command such as the following sample to find the Tenant ID:

   ```
   curl --location --request GET 'https://gauth-int./environment/v3/environments' \
    --header 'Content-Type: application/json' \
    --header 'Authorization: Basic ' \
   ```

   Create the GWS key:

   ```
   curl -X "POST" "http:///auth/v3/ops/clients" \
        -H 'Content-Type: application/json; charset=utf-8' \
        -u 'opsAdmin:opsPass' \
        -d $'{
     "data": {
       "authorities": [
         "ROLE_INTERNAL_CLIENT"
       ],
       "contactCenterIds": [
         ""
       ],
   ```

```
      "redirectURIs": [
        "https://ges-external-ges.apps./ges/ui/login.html",
        "http://ges-internal-ges.apps./ges/ui/login.html"
      ],
      "scope": [
        "*"
      ],
      "authorizedGrantTypes": [
        "client_credentials",
        "authorization_code",
        "refresh_token",
        "implicit",
        "password"
      ],
      "refreshTokenExpirationTimeout": 43200,
      "accessTokenExpirationTimeout": 43200,
      "clientType": "CONFIDENTIAL",
      "client_secret": "",  # NOTE: This is your unencrypted GWS client
  secret.Encrypted it will be your AUTHENTICATION-CLIENT-SECRET
      "name": "ges_client",                      # It is possible to change this name.
      "client_id": "ges_client"                  # It is possible to change this
  name. This is what is stored in AUTHENTICATION-CLIENT-ID
    }
  }'
```

In the preceding configuration, add values specific to your environment:

- : Specify the internal API of GWS.
- `contactCenterIds`: Specify your Tenant ID.
- `redirectURIs`: Specify the URL(s) that Designer uses.
- : Specify the domain address for the environment.
- `-u 'ops:ops'`: This is the default delivered for Tenants.
- `client_secret`: Specify any value (used in the secret).
- `name`: Specify any value.
- `client_id`: Specify your client ID (used in the secret).

5. Provision secrets for GES.
   - Create an infrastructure secret. This secret contains the username (DB), password (DB), and REDIS_CACHEKEY (GES Redis password; this is optional). Execute the following command:

     ```
     kubectl create secret generic ges-secrets-infra --from-literal=DB-USER= --from-
     literal=DB-PASSWORD= -- from-literal=REDIS_CACHEKEY=
     ```

   - Create an ORS Redis secret. This secret contains connection details for the Redis ORS stream. This has the servername (Redis Voice Cluster Hostname – redis-ors-stream), port (Redis Voice Cluster Port), and password (Redis Voice Cluster Password – if used). Execute the following command:

     ```
     kubectl create secret generic voice-redis-ors-stream --from-literal=voice-redis-
     ors-stream='{"password":"", "port":"", "rejectUnauthorized":"true",
     "servername":""}'
     ```

   - Create the GWS client secret. This secret contains the details for your GWS client (created earlier). Execute the following command:

     ```
     kubectl create secret generic gws-client-credentials --from-
     literal=AUTHENTICATION-CLIENT-ID= --from-literal=AUTHENTICATION-CLIENT-SECRET=
     ```

For more information about how GES leverages secrets and how they can be provisioned, see secrets and Using secrets to integrate with other software.

6. Download the required Helm chart release from the JFrog repository. For information about how to download the Helm charts, see Downloading your Genesys Multicloud CX containers. See Helm charts and containers for Callback for the Helm chart version you must download for your release.
7. Install the GES Helm charts:

```
helm install ges ./ -f ges-values.yaml
```

8. Using information in the Override Helm chart values section, create a Helm values file that describes the details of your GES deployment, secrets, and environment. A base Helm **values.yaml** file is included with the GES Helm charts that you receive from Genesys. The GES base file uses the filename, **ges-base-values.yaml**.
   Be sure to specify the following values in the **ges-values.yaml** overrides file for your GES deployment:

   - : Specify the data center region (Consul).
   - : Specify the address for the GES Redis server.
   - : Specify the port for the GES Redis server.
   - : Specify the address for the GES Postgres server.
   - : Specify the database name for GES Redis.
   - : Specify the internal cluster address for the gauth Authentication server.
   - : Specify the internal cluster address for the gauth Environment server.
   - : Specify the internal address for the GWS Service.
   - : Specify the external gauth address.
   - : Specify the address for the Redis-ORS-stream Redis server.
   - : Specify the port for the Redis-ORS-stream Redis server.
   - : Specify your "pull" secret name.

   This is a sample **ges-values.yaml** file:

```
ges:

  affinity: {}

  labels:
    globalLabels:
      version: "{{ .Release.Name }}"
    podLabels: {}
    serviceLabels: {}
  annotations:
    globalAnnotations: {}
    podAnnotations: {}

  configMap:
    env:
      extendEnv: "false"
      regionAffinity: ""
      devOpsAccessGroups: "Platform Read Only,Platform Provisioning,Platform
Internal Administrator"
```

```
      log:
        level: "DEBUG"
        max_len: 2048

      server:
        internal_port: 3050
        external_port: 3005
        internal_url: "ges.ges"

      ui:
        devops_username: "admin"
        devops_password: "letmein"


    integrations:

      # Omit if using secrets to supply Redis information
      redis:
        host:
        port:
        secure: "false"

      # Omit all but Secure if using secrets to supply DB information
      db:
        host:
        name:
        secure: "false"

      gws:
        auth:
        env:
        conf:
        public_auth:


      vmcs:
        redis_host:
        redis_port:
        redis_cluster_mode: "false"
        redis_stream_name: "NewCallbackStream"
        ors_redis_location: "{{ .Values.ges.configMap.integrations.vmcs.redis_host
}}:{{ .Values.ges.configMap.integrations.vmcs.redis_port | default 10000 }}"

  deployment:
    update_strategy:
      type: RollingUpdate
      rollingUpdate:
        maxUnavailable: 1
        maxSurge: 100%

    envFrom:
      - configMapRef:
          name: "{{.Values.ges.servicename}}-{{.Release.Name}}-configmap"

    min_ready_seconds: 300

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

    replicas: 1
```

```
  nodeSelector: {}

  image:
    imagePullSecrets: [  ]
    imagePullPolicy: "Always"
    registry: ""
    name: "genesys/ges"
    version: 9.0.048.00.build.205.rev.55fe5bd94

  hpa:
    enabled: true
    maximumReplicas: 5
    mimimumReplicas: 2
    targetMetrics:
      - type: Resource
        resource:
          name: cpu
          target:
            type: Utilization
            averageUtilization: 50
    behavior:
      scaleDown:
        stabilizationWindowSeconds: 60
        policies:
        - type: Pods
          value: 1
          periodSeconds: 30
      scaleUp:
        stabilizationWindowSeconds: 0
        policies:
        - type: Pods
          value: 1
          periodSeconds: 30
        selectPolicy: Max

priorityClassName: ""

resources:
  limits:
      cpu: "1250m"
      memory: "2048Mi"
  requests:
      cpu: "500m"
      memory: "512Mi"

ingress:
  ingressint:
    enabled: false
  networkPolicies:
    enabled: false

monitoring:
  prometheus:
    enabled: "false"
  newRelic:
    enabled: "false"

podDisruptionBudget:
  enabled: "true"
  strategy:
    minAvailable: 1

tolerations: {}
```

```
secrets:
  configFolders: "/secrets-infra,/secrets-ors,/secrets-gws"
  volumes:
    enabled: true
    list:
    - name: secrets-infra
      secret:
        secretName: ges-secrets-infra
    - name: secrets-ors
      secret:
        secretName: voice-redis-ors-stream
    - name: secrets-gws
      secret:
        secretName: gws-client-credentials
  volumeMounts:
    enabled: true
    list:
      - mountPath: /secrets-infra
        name: secrets-infra
        readOnly: true
      - mountPath: /secrets-ors
        name: secrets-ors
        readOnly: true
      - mountPath: /secrets-gws
        name: secrets-gws
        readOnly: true

service:
  port_internal: 80
  port_external: 8080
  type: "ClusterIP"


servicename: "ges"
```

9. Validate the GES helm charts using:
   ```
   > helm upgrade --install --dry-run -n -f
   ```
   Review the output to ensure that the generated Helm manifest matches your expectations.

Validate the deployment

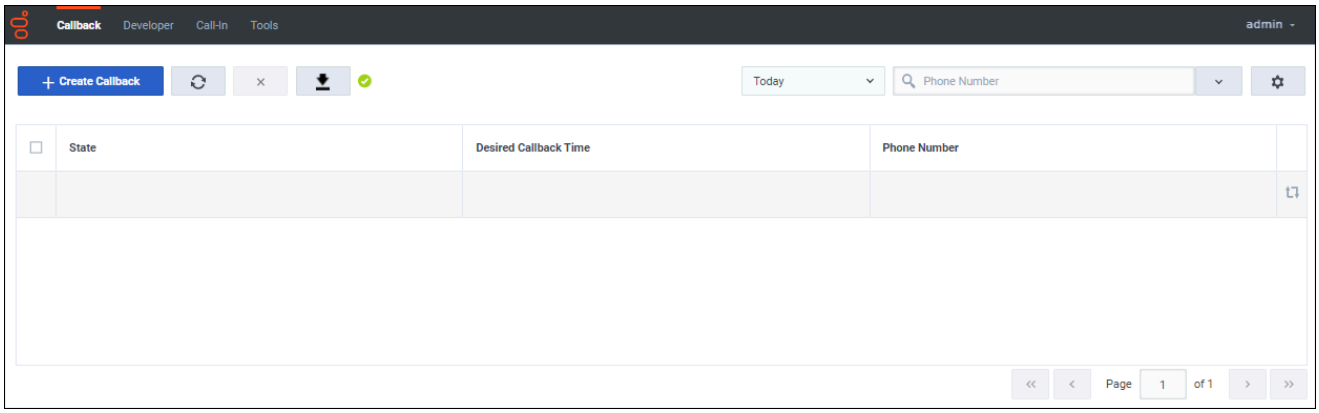
Once GES has been installed and provisioned following the processes outlined above, the best way to validate your deployment is to check that callbacks and other provisioned services work as intended. Logging in to the UI is a good way to check that privileges are working. Also, monitor callbacks, call-ins, and other requests to ensure that they complete successfully.

Validate the UI

Login to the UI.

With a user that has been assigned the **Callback Administrator** role, attempt to log into the Callback UI. You can do this at `/ges`.

If you successfully log in, then you see an empty **Callback** grid as the main feature of the screen:



Navigate to some of the other tabs and verify that no errors pop up in the lower corner of the screen as the system attempts to read and retrieve data.

If, at any point, data fails to load or you are locked out due to insufficient permissions, adjust the roles and privileges assigned to Callback UI users and try again.

Create a callback

There are two ways to create a callback request as an external user. You can create a callback request from the Callback UI or by using the /engagement API. For information about using the Callback UI to create a callback, see Managing callbacks. For information about using the APIs, see Genesys Multicloud CX REST APIs and tutorials for Callback.

Create Call-In request (optional)

The Click-to-Call-In scenario is optional functionality. If you plan to use the Click-to-Call-In feature, see the following pages for more information:

- Description of the Click-to-Call-In feature
- Provision the Click-to-Call-In scenario
- Configure Click-To-Call-In Groups
- View Click-To-Call-In records

Provision Genesys Engagement Service

Contents

- Administrator

Learn how to provision Genesys Engagement Service.

Related documentation:

-
-
-
-

RSS:

- For private edition

Create routes for GES to use

Create routes for Genesys Engagement Service (GES) only if routes were not created using the Helm chart while deploying the service.

Create an external route

Create an external route for GES. If you use the following sample as a guide, remember to replace with the value for your environment.

external-service.yaml:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ges-ext-ingress
  namespace: ges
  annotations:
    cert-manager.io/cluster-issuer:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        - ges.
      secretName: ges-ingress-ext
  rules:
    - host: ges.
      http:
        paths:
          - path: /.*
            pathType: ImplementationSpecific
            backend:
              service:
```

```
              name: ges
              port:
                number: 8080
```

Create the External Service spec for the external route:

```
kubectl create -f external-ingress.yaml
```

Create an internal route

Create an internal route for GES. If you use the following sample as a guide, remember to replace with the value for your environment.

internal-route.yaml:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ges-int-ingress
  namespace: ges
  annotations:
    cert-manager.io/cluster-issuer:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        - ges-int.
      secretName: ges-secret-int
  rules:
    - host: ges-int.
      http:
        paths:
          - path: /.*
            pathType: ImplementationSpecific
            backend:
              service:
                name: ges
                port:
                  number: 80
```

Create the External Service spec for the internal route:

```
kubectl create -f internal-ingress.yaml
```

Validate DesignerEnv in Agent Setup

Validate that the DesignerEnv transaction list is specified and correct for GES functionality. In Agent Setup, navigate to the DesignerEnv transaction on the **Transactions** page and review the **Annex** settings. The following settings must be specified (you might need to specify the flowsettings and gms sections and options):

- flowsettings > useSharedGms=true
- gms > _url=http://ges.ges.svc.cluster.local/genesys/

- reporting > ReportingURL=true

Currently, the ges section with key `gesurl` is not a required setting; it is not used in the system.

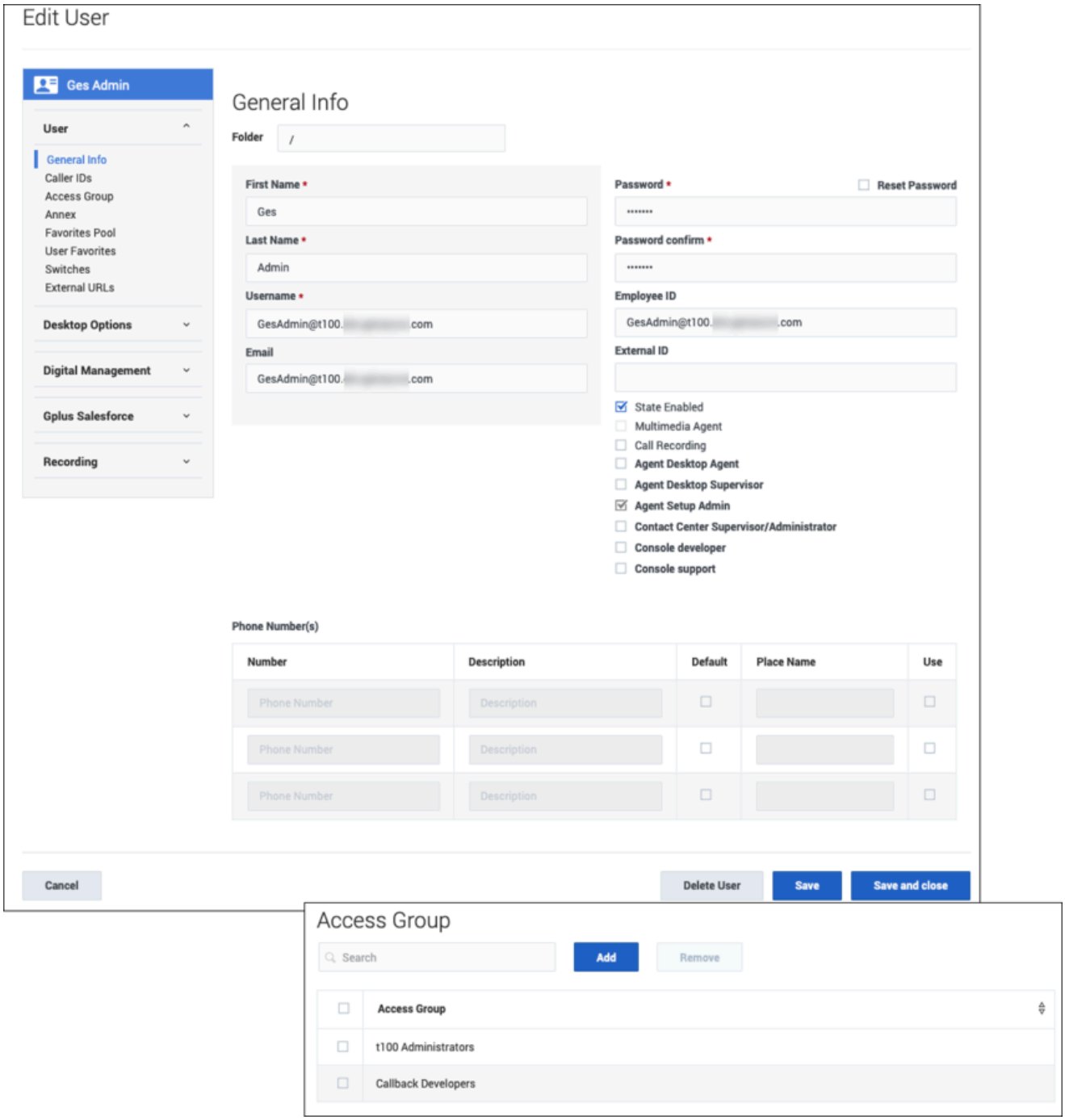
Annex tab options for DesignerEnv:

```
flowsettings
    customerfolder : Environment
    enablePTE : False
    httpProxy:"
    outboundTimeout : 20
    primarySwitch : SIP_Switch
    recordingType : FCR
    redundantHttpProxy:"
    useSharedGms : true

gms
    _url: http://ges.ges.svc.cluster.local/genesy/

reporting
    ReportingURL : true
```

Provision a User for GES

Use Agent Setup to provision an administrative User for GES. Be sure to configure appropriate Role-Based Access Control (RBAC) for the User. In other words, be sure to add the Admin User to the correct Access Groups for administrative-level access to the system or to GES:

- If you are configuring a general Admin User, use Access Group " Administrators". In our example, it is "t100 Administrators".
- If you are configuring an Admin User specifically for GES/callbacks, use Access Group "Callback Developers".



Provision GES in Designer

You provision GES/Callback in Designer. Log into Designer as a User who is a member of the **Designer Developers Group**. For a description of basic GES provisioning in Designer to support

callbacks, see Provisioning Callback in Designer.

> **Tip**
>
> When working with Voice Microservices, Genesys recommends that you create builds for your Designer applications.
>
> For your Default-type application, in the main Designer Application list view:
>
> - Assign the build that you created to the **Development** stream.
> - Associate a working DNIS to the **Development** stream.
>
> For your Callback-type application, in the main Designer Application list view:
>
> - Assign the build that you created to the **Development** stream so it can be used in the CALLBACK_SETTINGS data table.
>
> For more information about builds, see Manage Builds and Application workflow.

Provision the callback services (virtual queues)

The configuration parameters for callback services are stored in the Designer CALLBACK_SETTINGS data table. For detailed information about the CALLBACK_SETTINGS table, see Callback Settings Data Table in the Designer documentation.

You must make sure that the following prerequisites are completed before you add your queue to the CALLBACK_SETTINGS table:

- You must create the Callback-type Designer application before you can edit the CALLBACK_SETTINGS data table.
- You must configure the business hours, including the time zone, before making the following updates to the CALLBACK_SETTINGS data table. You cannot save the data table before the business hours are configured.
- The virtual queues that you will use for callback functionality must be created and saved in Platform Administration.

Configuration notes

When configuring settings in the CALLBACK_SETTINGS data table, use the following guidelines:

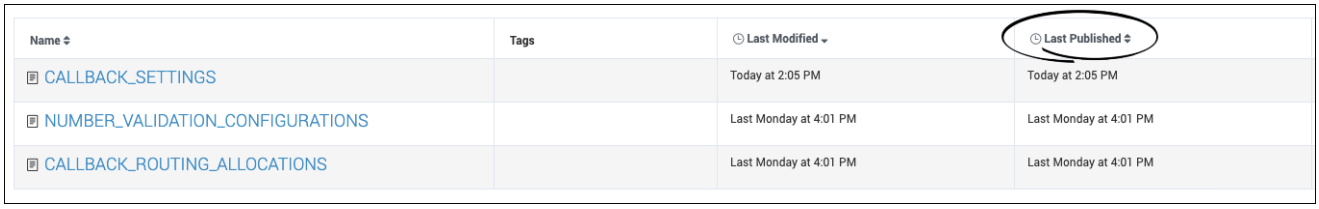
- **Call Display Numbe**r and **Callback Display Name** are optional settings.
- In the row in which you are configure the inbound virtual queue, select your Designer Callback-type application in the **Callback Application** column of the table.
- For the **Application Stream**, use the **Development** stream, as specified in the preceding Tip.
- The **Routing Point** setting is a free-form text box, but you must enter a valid routing point that is configured on the SIP switch.
- Whether or not you enter a **Dial Prefix** is dependent on the phone number format, dial plan set up, and so on. The actual outbound number used by Callback – before any transformations in the dial plan – is

in the format “dial prefix>callback phone number>”.

- If you deploy in AWS or Azure, use the “+” dial prefix, unless there is a very good reason not to. When used, the callback phone number> must be in E.164 format without the leading “+” (for example, 1xxxnnnnnnn format for North American numbers).
- For IVR-based callbacks, assuming the default Callback templates are used and number validation configurations are not used (the NUMBER_VALIDATION_CONFIGURATIONS data table):
  - If the ANI is properly detected (at the SIP level) and provided to the Designer strategy, and if it is in full E.164 format (for example, +1xxxnnnnnnn for North American numbers), the system states the ANI to the caller without the leading "+"; for example, "1xxxnnnnnnn".
  - If the ANI is available and the caller accepts the spoken phone number, then that is the phone number that the system uses to book the callback. For example, if the ANI is +12223334444 and the caller accepts that detected number, then the callback is booked with phone number 12223334444.
  - If the ANI is not detected, or if the caller chooses to enter a different phone number, the phone number that the caller enters is used to book the callback. For example, if the caller entered 2223334444 as the phone number, the callback is booked with the phone number 2223334444. In this example, note the difference between the detected ANI (12223334444) and the entered phone number (2223334444).
- Continuing with IVR callbacks, if you use the default Callback templates with the virtual queue that is configured in the NUMBER_VALIDATION_CONFIGURATIONS data table, then phone numbers are normalized into the E.164 format without the leading “+” sign, regardless if it is a detected ANI or one that is manually entered.
- For web callbacks, number validation configurations (in the NUMBER_VALIDATION_CONFIGURATIONS table) do not apply. Instead, the phone number that the caller provides when booking the callback is used as is.

Publishing data tables

After you complete configuration in the relevant data tables in Designer – for example, the CALLBACK_SETTINGS and NUMBER_VALIDATION_CONFIGURATIONS data tables – make sure you save and publish the tables. If you see tables in the list view that do not have a **Last Published** date and time, then publish them now.



Name	Tags	Last Modified	Last Published
CALLBACK_SETTINGS		Today at 2:05 PM	Today at 2:05 PM
NUMBER_VALIDATION_CONFIGURATIONS		Last Monday at 4:01 PM	Last Monday at 4:01 PM
CALLBACK_ROUTING_ALLOCATIONS		Last Monday at 4:01 PM	Last Monday at 4:01 PM

Configuring callback features

For information about how to configure Click-to-Call-In, CAPTCHA, Push Notifications, and other advanced use case scenarios, see the *Callback Administrator's Guide*.

Additional Roles and Access Groups

In addition to administrator-type users, you must assign users to specific Roles and Access Groups for access to the Callback UI. Genesys Callback uses Role-based-access-control (RBAC) to allow and restrict users' activities within the Callback UI.

For more information about Callback-related Roles and Access Groups, see Controlling user access.

Provision an API key for GES

Contents

Use the Environment Service API to assign API keys to a specific contact center and to validate that API keys were successfully assigned to the correct contact center.

Related documentation:

-
-
-
-

RSS:

- For private edition

If you plan to work with the Genesys Engagement Service (GES) APIs, then you must first provision an API key. An API key is a unique identifier for an application within a specific contact center. Genesys uses your GES API key to associate a Callback API request with your tenant. If an API key has not been provisioned for GES, then you cannot use the GES APIs. On the other hand, if you use basic in-queue callbacks only, you do not need to provision the API key.

The contact center-to-API key relationship is one-to-many. That is, one contact center can have many API keys assigned to it. An API key can be assigned to only one contact center.

To provision an API key for GES, you use the Environment Service API in the **gauth** namespace. For information about Genesys Authentication, see About Genesys Authentication.

All operations supported by the Environment Service API require authenticated access. You'll need client credentials and an authentication token to add an API key to the Environment Service. The token is sent with the request to the Environment Service to store the API key. For information, see Create an authentication token.

What you'll need to do to provision an API key

To provision an API key for GES, you'll do the following:

- Create an API key.
  The API key must be unique. You decide what you want to use for a key, but Genesys recommends that you generate a UUIDv4 and use that as your API key for GES.
- Find the contact center ID (CCID) for the contact center to which you'll assign the API key.
- Assign the API key to the contact center.
- Validate the API key. You can use a GET request to check that the API key is assigned to the correct CCID. You can also use the Callback UI to validate the API key.

If you need to remove an API key from a CCID, see Unassign an API key.

For more information about using the Environment Service API, including how to change CORS settings, or information about Callback APIs, see Additional resources.

Get the appropriate contact center ID

To manage the GES API key, you require the contact center ID (CCID) for the contact center where GES is deployed. Use a GET operation to request the CCID from the appropriate Environment pod. Include the GES Environment node and port information in the request. For example:

```
GET env:8091/environment/v3/contact-centers
```

The API responds with the CCID, Environment ID, domain, and authentication information.

If you don't have a CCID already assigned to the appropriate contact center, see Add a contact center.

Assigning an API key to a contact center

You create your own API key. Use an API key that is random and unique. The assignment of an API key fails if the API key already exists or is assigned to another contact center. Genesys recommends that you generate a UUIDv4 and use that as your API key.

Remember that all operations supported by the Environment Service API require authenticated access. You'll need client credentials and an authentication token to assign the API key to a contact center. The token is sent with the request to the Environment Service to store the API key. For information, see Create an authentication token.

To provision and associate an API key with your contact center, use the following POST request:

```
POST .../environment/v3/contact-centers/{ccid}/api-keys
```

Include the API key attribute (`apiKey`) in the body of the POST API request. For example:

```
curl --location --request POST 'env:8091/environment/v3/contact-centers/
ed4c03f3-6275-4419-8b2b-11d14af10655/api-keys' \

--header 'Content-Type: application/json' \
--header 'Authorization: Bearer f3aa2109-8889-4182-b2b7-d86917c53e4e' \
--data-raw '{
    "data": {
       "apiKey" : "78fd8f03-d9d0-4140-8aae-09f3f15d1392"
    }
}'
```

After you've assigned the API key to the contact center, the key is used as the value for `x-api-key` in the header of every request to the GES APIs.

Validating that the API key is correctly assigned to a contact center

To verify that the API key for GES was successfully assigned to the contact center where GES is deployed, use the API key that you created for GES to look up the CCID. You can also use the CCID to retrieve the collection of API keys assigned to the contact center; ensure the API key for GES is included.

To find the contact center to which a specific API key is assigned, use the following GET request. The API key must be URL-encoded. For information, see Google URL-Encoding.

```
GET .../environment/v3/contact-centers/api-key/
```

The API response provides the CCID, Environment ID, domain, and authentication information.

To retrieve the list of API keys assigned to a contact center, use the following GET request:

```
GET .../environment/v3/contact-centers/{ccid}/api-keys
```

The API returns the list of API keys assigned to the CCID.

Validating an API key using the Callback UI

For information about validating an API key using the Callback UI, see Validating your API key in the *Callback Administrator's Guide*.

Unassigning an API key

To remove an API key from a contact center, use the following DELETE operation, where {apiKey} is the API key you want to remove from the contact center:

```
DELETE .../environment/v3/contact-centers/{ccid}/api-keys/{apiKey}
```

Additional resources

When provisioning an API key for GES or working with GES APIs, you might find the following resources helpful:

- For detailed information about using the Genesys Authentication Environment Service API to create authentication tokens or to add a Genesys tenant, contact center, or data center, see Provision Genesys Authentication.
- The documentation for GES REST APIs is available through the Genesys Multicloud CX Developer Center. The GES/Callback APIs are:

  - Callbacks — Create, retrieve, cancel callbacks.
  - Estimated Wait Time —Retrieve Estimated Wait Time.
  - Availability — Retrieve time slots for a callback, matching Office Hours.
  - Call In — Request the phone number to call in.
  - Queue Status — Retrieve information about a queue's readiness to accept callbacks.
  - Statistics — Provides a proxy to the GWS Statistics API. To use the Callback Statistics API, you must first register your GWS credentials in the Callback UI (**Developer** > **Credential Management** > **GWS Credentials** tab).

- For web-based callbacks, you might need to configure or update CORS domains. For information, see Update CORS settings.

Upgrade, roll back, or uninstall Genesys Engagement Service

Contents

Learn how to upgrade, roll back, or uninstall Genesys Engagement 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

Genesys Callback supports the following upgrade strategies:

Service	Upgrade Strategy	Notes
Genesys Engagement 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.

Versions must match	Ensure that the version of GES and the Helm chart match. While Helm charts might work with other versions of GES, this is not tested and Genesys cannot guarantee success.

Provided there are no issues or errors when running the Helm command, you can expect GES to work with any updates supplied in the latest version of the Helm charts. However, in some instances, there might be changes to the GES Helm manifest that, due to restrictions in Kubernetes, require the existing GES deployment to be uninstalled and then re-installed, rather than simply upgraded in place. Consult the Callback Release Notes to see if this will be necessary.

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.

Upgrade the Helm deployment for GES during a prescribed maintenance window.

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 Genesys Callback).

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=
```

Example

```
> helm upgrade --install -n -f
```

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.

You can verify that GES has upgraded successfully by either looking at dashboards (if provisioned) or using the health check APIs for GES (through the internal port query `ges:3050/ges/v1/health/detail`) to make sure that all dependencies are working as provided.

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
```

Depending on how services like Postgres and Redis are provisioned for GES, you will have to decommission those services separately. Discussion of that process is outside the scope of this document.

Observability in Genesys Callback

Contents

Learn about the logs, metrics, and alerts you should monitor for Genesys Callback.

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 Genesys Callback metrics, see:

-

See also System metrics.

Enable monitoring

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
	Supports both CRD (Service Monitor) and annotations	3050	/metrics	Real-time updates

Configure metrics

The metrics that are exposed by Genesys Engagement Service (GES) 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 page linked to above shows some of the metrics GES exposes. You can also query Prometheus directly or via a dashboard to see all the metrics available from GES.

What do Genesys Callback metrics monitor?

The GES/Callback metrics are divided into three categories:

- Metrics that have to do with the internal business logic of GES: This includes metrics that measure things like the number of callbacks booked, the number of click-to-call requests booked, callback monitor performance, and API usage.
- Metrics that measure the performance of GES: Disk, memory, and CPU usage, event loop lag, request handling times, and the health of connections to downstream services such as Redis, PostGres, GWS, ORS, and others.
- Alarms-type metrics, or alerts: These metrics are boolean flags that raise and lower whenever certain conditions are met.

Watching how the metrics change over time helps you understand the performance of a given GES deployment or pod. The sample Prometheus expressions show you how to use the basic metrics to gain valuable insights into your callback-related activity.

Health metrics

Health metrics – that is, those metrics that report on the status of connections from GES to dependencies such as Tenant Service (ORS), GWS, Redis, and Postgres – are implemented as a gauge that toggles between "0" and "1". For information about gauges, see the Prometheus Metric types documentation. When the connection to a service is down, the metric is "1". When the service is up, the metric is "0". Also see Alerting.

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 Genesys Callback alerts, see:

-

In a Kubernetes deployment, GES relies on Prometheus and Alertmanager to generate alerts. These alerts can then be fowarded to a service of your choice (for example, PagerDuty).

One of the key things to understand about alerts in GES is that, while GES leverages Prometheus, the application manually triggers alerts when certain criteria are met. This internal alert is then turned into a counter that is incremented each time the conditions to trigger the alert are met. The counter is available on the `/metrics` endpoint. Prometheus rules capture the metric data and trigger the alert on Prometheus; also see Configure alerts. For more information about counters, see the Prometheus Metric types documentation.

Because alerts are implemented as counters in GES, you can leverage metrics to analyze how the counters increase over time for a given deployment or pod. For a list of helpful Prometheus expressions to use for this purpose, see Sample Prometheus expressions.

The following example shows an alert used in an Azure deployment; an increase in instances of the alert firing over a certain period of time triggers the Prometheus alert.

```
- alert: GES_RBAC_CREATE_VQ_PROXY_ERROR
annotations:
  summary: "There are issues managing VQ proxy objects on {{ $labels.pod }}"
labels:
  severity: info
  action: email
  service: GES
expr: increase(RBAC_CREATE_VQ_PROXY_ERROR[10m]) > 5
```

Health alerts in GES work a little differently. They are gauges, rather than counters. The gauge toggles between "0" and "1"; "1" indicates that the service is down and "0" indicates that the service is up. Because GES has an automatic health check that runs approximately every 15-20 seconds, the health alerts are generated when a connection has been in the DOWN state for a given period of time. The following example shows the ORS_REDIS_DOWN alert.

```
- alert: GES_ORS_REDIS_DOWN
        expr: ORS_REDIS_STATUS > 0
        for: 5m
        labels:
          severity: critical
          action: page
          service: GES
        annotations:
            summary: "ORS REDIS Connection down for {{ $labels.pod }}"
            dashboard: "See GES Performance > Health and Liveliness to track ORS Redis Health
over time"
```

Configure alerts

Private edition services define a number of alerts by default (for Genesys Callback, 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, Genesys Callback does not support customizing the alerts or defining additional **PrometheusRule** objects to create alerts based on the service-provided metrics.

Logging

For solution-level documentation about logging, see Logging overview and approaches.

GES outputs logs to standard output (stdout).

GES log size is highly dependent on usage. A high-traffic enterprise that accepts many callbacks daily will have much larger log sizes than an enterprise with only a handful of callbacks each day. For your reference, a single Create Callback operation, invoked from the Callback UI, generates 16 log messages in 1 second at full trace level. Therefore, at a rate of 10 callbacks/second, there are 160 log messages per second.

You can make changes to logging settings using the Helm values file. For information about the `log` values included in the Helm charts, see the configMap section.

No results metrics and alerts

Contents

Find the metrics *No results* exposes and the alerts defined for *No results*.

Related documentation:

-

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval
No results	Supports both CRD (Service Monitor) and annotations	3050	/metrics	Real-time updates

See details about:

- *No results* metrics
- *No results* alerts

Metrics

GES exposes some default metrics such as CPU usage, memory usage, and the state of the Node.js runtime, as well as metrics coming directly from the GES API such as the number of created callbacks, call-in requests, and so on. These basic metrics are created as counters, which means that the values will monotonically increase over time from the beginning of a GES pod's lifespan. For more information about counters, see Metric Types in the Prometheus documentation.

You might see metrics documented on this page that you cannot find on the endpoint or – if they exist – they might have no value. These are alert-type metrics. This type of metric is set when the condition it tracks is first encountered. For example, if GES has never experienced a DNS failure since it started, then no GES_DNS_FAILURE alert has ever been generated and the GES_DNS_FAILURE metric would not yet exist. For more information, see Alerting.

You might see metrics with almost identical names, except for case (upper or lower). Metrics with names ending in `_tolerance` are simply thresholds and exist at the level at which an alert is triggered; they are not the same as the metric used for monitoring. For more information, see Alerting.

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

Metric and description	Metric details	Indicator of
ges_callbacks_created	**Unit:** N/A	The number of callbacks booked in GES

Metric and description	Metric details	Indicator of
The number of callbacks booked in GES since the deployment went online.	**Type:** counter **Label:** tenant – The tenant for which the callback was booked. **Sample value:**	
ges_monitor_size The number of booked callbacks currently being monitored and managed in GES. This is a background task that both ensures that new callbacks are propagated to Redis and that callbacks are dispatched to ORS when appropriate. If this metric is consistently high, it might indicate issues with the GES deployment.	**Unit:** callback **Type:** gauge **Label:** type – The type of callback monitor. **Sample value:** 3	Latency related to starting scheduled callbacks
ges_push_notifications_sent The number of Push Notifications sent since the deployment went online. This tracks notifications that were both successfully and unsuccessfully dispatched.	**Unit:** N/A **Type:** counter **Label:** tenant – The tenant for which the Push Notification request was created. channel – The channel through which the Push Notification is delivered. Currently, this can be only Google FCM ("FCM"). result – Either "success" or "failure" based on whether the notification was successfully dispatched or not. **Sample value:** 3	How many Push Notifications that GES has dispatched
ges_http_requests_total The number of HTTP requests handled by GES since the deployment went online. This metric does not delineate between successful and unsuccessful requests.	**Unit:** N/A **Type:** counter **Label:** tenant – The tenant associated with the request. If no tenant can be identified, this defaults to "Unknown Tenant". path – The path of the request. If a private endpoint, then it is “Private API Endpoint”. **Sample value:**	Overall GES activity and usage
ges_callin_created The total number of Click-to-Call-In requests handled since the GES deployment went online.	**Unit:** N/A **Type:** counter **Label:** tenant – The tenant for which the Click-to-Call-In request was booked. **Sample value:**	The number of Click-to-Call-In requests GES has received
ges_http_failed_requests_total The amount of failed (4XX/5XX) requests handled by GES since the deployment came online.	**Unit:** N/A **Type:** counter **Label:** tenant – The tenant associated with the request. If no tenant can be identified, this defaults to "Unknown Tenant". path – The path of the request. httpCode – The HTTP code associated with the result. **Sample value:**	Dependent on which HTTP codes you observe. Excessive 500 codes might indicate an issue with configuration or with GES itself. Excessive 400 errors might indicate malicious behavior.
ges_build_info Displays the version of GES that is	**Unit:** N/A **Type:** gauge	Software version

Metric and description	Metric details	Indicator of
currently running. In the case of this metric, the labels provide the important information. The metric value is always 1 and does not provide any information.	**Label:** version – The version of GES that you are running in your deployment. **Sample value:** `ges_build_info{version="100.0.000.0000.build.69.rev.d07b89146"}` 1	
GES_HEALTH The overall health of the GES deployment; this is a composite of the connection statuses of GES and downstream services. Values are: 1 – healthy 0 – unhealthy If a value is not exported, assume that GES is healthy (unless the /metrics endpoint can't be reached).	**Unit:** N/A **Type:** gauge **Label:** version – The GES version that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	The overall health of the GES deployment and connections
GWS_CONFIG_STATUS The status of the connection to the GWS Configuration Service. Values are: 1 – healthy 0 – unhealthy	**Unit:** N/A **Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	Health of the connection to the GWS Configuration Service
GWS_ENV_STATUS The status of the connection to the GWS Environment Service. Values are: 1 – healthy 0 – unhealthy	**Unit:** N/A **Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	Health of the connection to the GWS Environment Service
GWS_AUTH_STATUS The status of the connection to the Genesys Authentication Service. Values are: 1 – healthy 0 – unhealthy	**Unit:** N/A **Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	Health of the connection to the GWS Authentication Service
ALL_URS_DOWN A flag that raises when connections to both the primary and secondary URS components are unhealthy. Values are: 1 – healthy 0 – unhealthy If the metric is not being exposed, assume that the value is 0 and that URS connections are in an unhealthy state.	**Unit:** N/A **Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	Health of the connection from GES to URS
REDIS_CONNECTION	**Unit:** N/A	Health of the connection to Redis

Metric and description	Metric details	Indicator of
Monitors the health of the connection between GES and its own Redis instance. Values are: 1 – healthy 0 – unhealthy Because GES is so dependent on Redis, you might have trouble confirming – with metrics – when Redis is actually down (GES might not respond to the `/metrics` query).	**Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	
ORS_REDIS_STATUS Monitors the health of the connection between GES and the ORS Redis instance. Values are: 1 – healthy 0 – unhealthy	**Unit:** N/A **Type:** gauge **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:** 1	Health of the connection to ORS Redis.
RBAC_CREATE_VQ_PROXY_ERROR The number of times GES has encountered issues when managing virtual queue proxy objects. When a callback service (also called a virtual queue, or VQ) is added to GES using the CALLBACK_SETTINGS data table in Designer, GES automatically creates a script object for line-of-business segmentation (see Line of Business segmentation). When the callback service (VQ) is removed from the CALLBACK_SETTINGS data table, GES automatically deletes the script object.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	The ability of GES to create or delete the script objects.
LOGGING_FAILURE The number of times GES has encountered issues writing logs to standard output (stdout).	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	Typically indicates some sort of issue with the Kubernetes pod or the host
UNCAUGHT_EXCEPTION The number of times GES has encountered an uncaught exception while running.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	There is no specific problem that this metric indicates. Check the logs for more information.
GES_DNS_FAILURE The number of times GES has encountered a failure in performing DNS resolution.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment.	Certain configuration values such as the location of GWS, Redis, Postgres, or ORS might be incorrect

Metric and description	Metric details	Indicator of
	Sample value:	
GWS_INCORRECT_CLIENT_CREDENTIALS The number of times that authentication on GWS has failed because the client credentials that were supplied were incorrect.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	Incorrect client credentials are being supplied to GWS. Check that correct credentials have been made available in the secret.
NEXUS_ACCESS_FAILURE The number of times the GES deployment has failed to contact Nexus. This is only relevant if you use the Push Notification feature.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	Indicates issues with the Nexus deployment or the connection from GES to Nexus.
CB_SUBMIT_FAILED The number of times that GES has failed to submit a callback to ORS.	**Unit:** N/A **Type:** counter **Label:** version – The version of GES that you are running in your deployment. host – The hostname associated with the GES deployment. **Sample value:**	There might be issues with the ORS deployment. GES could also be supplying an incorrect URL to ORS. Change the GES_URL environment variable to fix the latter issue.

Alerts

The following alerts are defined for *No results*.

Alert	Severity	Description	Based on	Threshold
GES_UP	Critical	Fires when fewer than two GES pods have been up for the last 15 minutes.		Triggered when fewer than two GES pods are up for 15 consecutive minutes.
GES_CPU_USAGE	Info	GES has high CPU usage for 1 minute.	ges_process_cpu_seconds_total	Triggered when the average CPU usage (measured by ges_process_cpu_seconds_total) is greater than 90% for 1 minute.
GES_MEMORY_USAGE	Info	GES has high memory usage for	ges_nodejs_heap_space_size_used_bytes, ges_nodejs_heap_space_size_available_bytes	Triggered when

No results metrics and alerts

Alert	Severity	Description	Based on	Threshold
		a period of 90 seconds.		(measured as a ratio of Used Heap Space vs Available Heap Space) is above 80% for a 90-second interval.
GES-NODE-JS-DELAY-WARNING	Warning	Triggers if the base NodeJS event loop becomes excessive. This indicates significant resource and performance issues with the deployment.	application_ccecp_nodejs_eventloop_lag_seconds	Triggered when the event loop is greater than 5 milliseconds for a period exceeding 5 minutes.
GES_NOT_READY_CRITICAL	Critical	GES pods are not in the Ready state. Indicative of issues with the Redis connection or other problems with the Helm deployment.	kube_pod_container_status_ready	Triggered when more than 50% of GES pods have not been in a Ready state for 5 minutes.
GES_NOT_READY_WARNING	Warning	GES pods are not in the Ready state. Indicative of issues with the Redis connection or other problems with the Helm deployment.	kube_pod_container_status_ready	Triggered when 25% (or more) of GES pods have not been in a Ready state for 10 minutes.
GES_PODS_RESTART	Critical	GES pods have been excessively crashing and restarting.	kube_pod_container_status_restarts_total	Triggered when there have been more than five pod restarts in the past 15 minutes.
GES_HEALTH	Critical	One or more downstream components (PostGres, Config Server, GWS, ORS) are down. **Note:** Because GES goes into a crash loop when Redis is down, this does not fire when	GES_HEALTH	Triggered when any component is down for any length of time.

No results metrics and alerts

Alert	Severity	Description	Based on	Threshold
		Redis is down.		
GES_ORS_REDIS_DOWN	Critical	Connection to ORS_REDIS is down.	ORS_REDIS_STATUS	Triggered when the ORS_REDIS connection is down for 5 consecutive minutes.
GES_GWS_AUTH_DOWN	Warning	Connection to the Genesys Authentication Service is down.	GWS_AUTH_STATUS	Triggered when the connection to the Genesys Authentication Service is down for 5 minutes.
GES_GWS_ENVIRONMENT_DOWN	Warning	Connection to the GWS Environment Service is down.	GWS_ENV_STATUS	Triggered when the connection to the GWS Environment Service is down.
GES_GWS_CONFIG_DOWN	Warning	Connection to the GWS Configuration Service is down.	GWS_CONFIG_STATUS	Triggered when the connection to the GWS Configuration Service is down.
GES_GWS_SERVER_ERROR	Warning	GES has encountered server or connection errors with GWS.	GWS_SERVER_ERROR	Triggered when there has been a GWS server error in the past 5 minutes.
GES_HTTP_400_POD	Info	An individual GES pod is returning excessive HTTP 400 results.	ges_http_failed_requests_total, http_400_tolerance	Triggered when two or more HTTP 400 results are returned from a pod within a 5-minute period.
GES_HTTP_404_POD	Info	An individual GES pod is returning excessive HTTP 404 results.	ges_http_failed_requests_total, http_404_tolerance	Triggered when two or more HTTP 404 results are returned from a pod within a 5-minute period.

No results metrics and alerts

Alert	Severity	Description	Based on	Threshold
GES_HTTP_500_POD	Info	An individual GES pod is returning excessive HTTP 500 results.	ges_http_failed_requests_total, http_500_tolerance	Triggered when two or more HTTP 500 results are returned from a pod within a 5-minute period.
GES_HTTP_401_POD	Info	An individual GES pod is returning excessive HTTP 401 results.	ges_http_failed_requests_total, http_401_tolerance	Triggered when two or more HTTP 401 results are returned from a pod within a 5-minute period.
GES_SLOW_HTTP_RESPONSE_TIME	Warning	Fired if the average response time for incoming requests begins to lag.	ges_http_request_duration_seconds_sum, ges_http_request_duration_seconds_count	Triggered when the average response time for incoming requests is above 1.5 seconds for a sustained period of 15 minutes.
GES_RBAC_CREATE_VQ_PROXY_ERROR	Info	Fires if there are issues with GES managing VQ Proxy Objects.	RBAC_CREATE_VQ_PROXY_ERROR, rbac_create_vq_proxy_error_tolerance	Triggered when there are at least 1000 instances of issues managing VQ Proxy objects within a 10-minute period.
GES_LOGGING_FAILURE	Warning	GES has failed to write a message to the log.	LOGGING_FAILURE	Triggered when there are any failures writing to the logs. Silenced after 1 minute.
GES_UNCAUGHT_EXCEPTION	Warning	There has been an uncaught exception within GES.	UNCAUGHT_EXCEPTION	Triggered when GES encounters any uncaught exceptions. Silenced after 1 minute.

No results metrics and alerts

Alert	Severity	Description	Based on	Threshold
GES_INVALID_CONTENT_LENGTH	Info	Fires if GES encounters any incoming requests that have exceeded the maximum content length of 10mb on the internal port and 500kb for the external, public-facing port.	INVALID_CONTENT_LENGTH, invalid_content_length_tolerance	Triggered when one instance of a message with an invalid length is received. Silenced after 2 minutes.
GES_DNS_FAILURE	Warning	A GES pod has encountered difficulty resolving DNS requests.	DNS_FAILURE	Triggered when GES encounters any DNS failures within the last 30 minutes.
GES_CB_TTL_LIMIT_REACHED	Info	GES is throttling callbacks for a specific tenant.	CB_TTL_LIMIT_REACHED	Triggered when GES has started throttling callbacks within the past 2 minutes.
GES_CB_ENQUEUE_LIMIT_REACHED	Info	GES is throttling callbacks to a given phone number.	CB_ENQUEUE_LIMIT_REACHED	Triggered when GES has begun throttling callbacks to a given number within the past 2 minutes.
GES_CB_SUBMIT_FAILED	Info	GES has failed to submit a callback to ORS.	CB_SUBMIT_FAILED	Triggered when GES has failed to submit a callback to ORS in the past 2 minutes for any reason.
GES_GWS_INCORRECT_CLIENT_CREDENTIALS	Warning	The GWS client credentials provided to GES are incorrect.	GWS_INCORRECT_CLIENT_CREDENTIALS	Triggered when GWS has had any issue with the GES client credentials in the last 5 minutes.

No results metrics and alerts

Alert	Severity	Description	Based on	Threshold
GES_NEXUS_ACCESS_FAILURE	Warning	GES has been having difficulties contacting Nexus. This alert is only relevant for customers who leverage the Push Notification feature in Genesys Callback.	NEXUS_ACCESS_FAILURE	Triggered when GES has failed to connect or communicate with Nexus more than 30 times over the last hour.