Genesys Info Mart Private Edition Guide

5/26/2022

Table of Contents

Contents

Find links to all the topics in this guide.

Related documentation:

-
-

Genesys Info Mart is the backend service behind historical reporting in the Genesys Multicloud CX private edition offering.

Overview

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

- About Genesys Info Mart
- Architecture
- High availability and disaster recovery

Configure and deploy

Find out how to configure and deploy the services in the Genesys Info Mart service group:

- GIM Stream Processor (GSP)
- GIM Config Adapter (GCA)
- GIM and the Info Mart database

- Before you begin GSP deployment
- Configure GSP
- Deploy GIM Stream Processor
- Upgrade, rollback, or uninstall GSP

- Before you begin GCA deployment
- Configure GCA
- Deploy GIM Config Adapter
- Upgrade, rollback, or uninstall GCA

- Before you begin GIM deployment
- Configure GIM
- Deploy GIM
- Upgrade, rollback, or uninstall GIM

Operations

Learn how to monitor Genesys Info Mart with metrics and logging.

- Observability in Genesys Info Mart
- GSP metrics and alerts
- GCA metrics and alerts
- GIM metrics and alerts

Info Mart database

Learn about the Info Mart database.

- Genesys Info Mart Historical Database Reference

About Genesys Info Mart

Contents

Learn about Genesys Info Mart and how it works in Genesys Multicloud CX private edition.

Related documentation:

-
-

Genesys Info Mart is the enterprise-level Genesys service behind the historical reports in your cloud deployment. Genesys Info Mart receives data from various upstream Genesys services, then processes the low-level data to produce a data mart that the Genesys Multicloud CX historical reporting presentation layer, called Genesys CX Insights (GCXI), uses for contact center historical reporting.

Genesys Info Mart comprises three services—GIM Stream Processor (GSP), GIM Config Adapter (GCA), and Genesys Info Mart (GIM). Upstream services responsible for managing contact center configuration, interaction activity, agent activity, and so on publish messages to Apache Kafka. Genesys Info Mart consumes the data in Kafka reporting topics and, on a regular ETL cycle, transforms the data into a form more suitable for data analysis, then loads the data into the Info Mart database.

The Info Mart database stores the processed data. In addition, a separate aggregation service called Reporting and Analytics Aggregates (RAA) aggregates or re-aggregates the processed data and stores the aggregate data in the Info Mart database. The historical reports available in your cloud deployment are based on this aggregate data.

Supported Kubernetes platforms

Genesys Info Mart is supported on the following cloud platforms:

- Google Kubernetes Engine (GKE)
- OpenShift Container Platform (OpenShift)

See the Genesys Info Mart Release Notes for information about when support was introduced.

Architecture

Contents

Learn about Genesys Info Mart architecture.

Related documentation:

-
-

Introduction

Genesys Info Mart is the historical reporting back-end service that performs extract, transform, and load (ETL) processing of data generated by contact center activity. Genesys Info Mart comprises the following services:

- GIM Config Adapter (GCA) — Reads configuration data from the contact center Configuration Server database and stores the data into Kafka topics for consumption during transformation processing. There is a separate GCA service for each tenant.
- GIM Stream Processor (GSP) — Processes GCA-sourced configuration data as well as raw event data streamed to Kafka from upstream services handling interaction and agent activity on Voice and Digital channels. GSP transforms the data into data structures suitable for a data mart for contact center historical reporting.
- GIM — Pulls the transformed data from Kafka and loads this data into the Info Mart database. The GIM service also performs other database-related tasks, such as a regular maintenance job and an export job to export Info Mart data. There is a separate GIM service and Info Mart database for each tenant.

All the Genesys Info Mart services are stateful. This has implications for high availability (HA) and disaster recovery (DR) (see High availability and disaster recovery) as well as for upgrade methods, because you cannot deploy mirrored services in parallel.

Unless you are using a single namespace for an OpenShift deployment, each Genesys Info Mart service is in its own namespace.

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

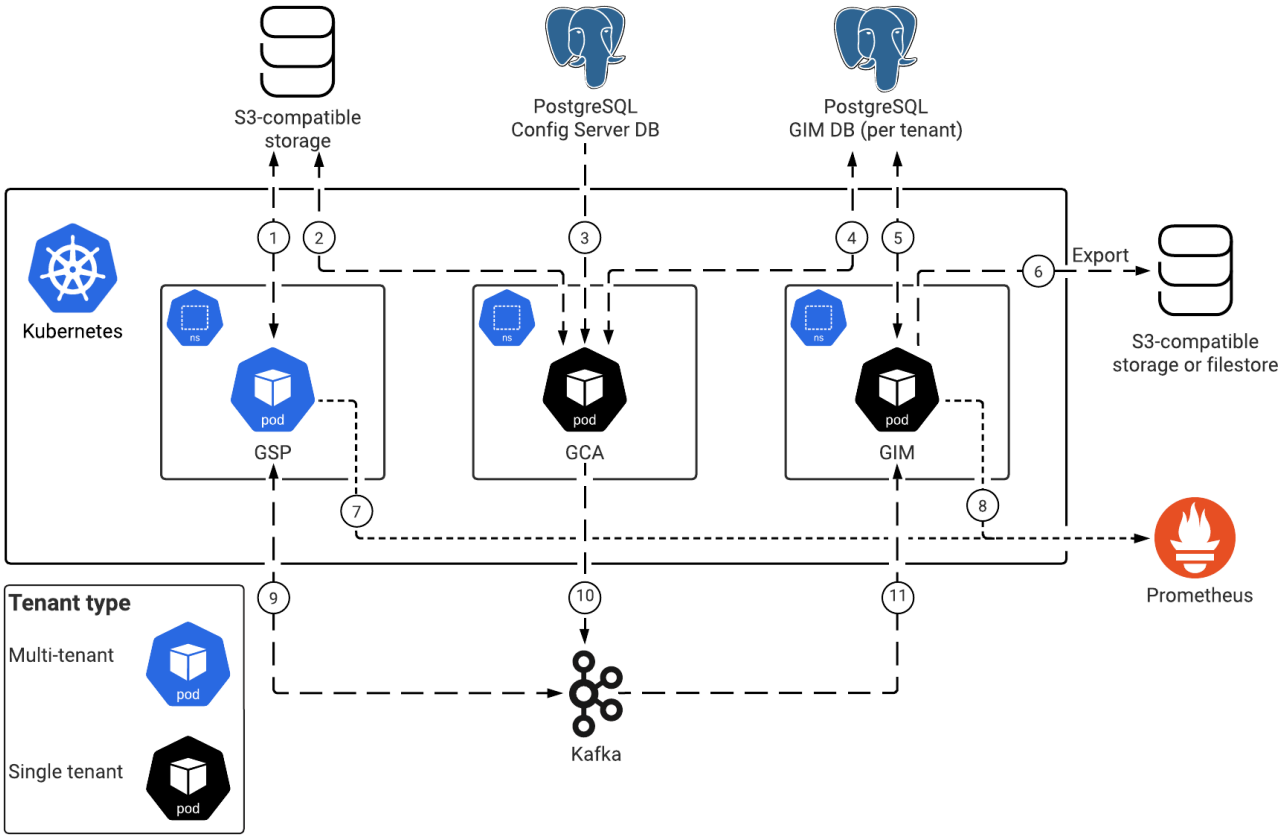
The following diagrams show the Genesys Info Mart architecture.

- Architecture diagram — Data flows is from the point of view of data.
- Architecture diagram — Connections is from the point of view of network connections.

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

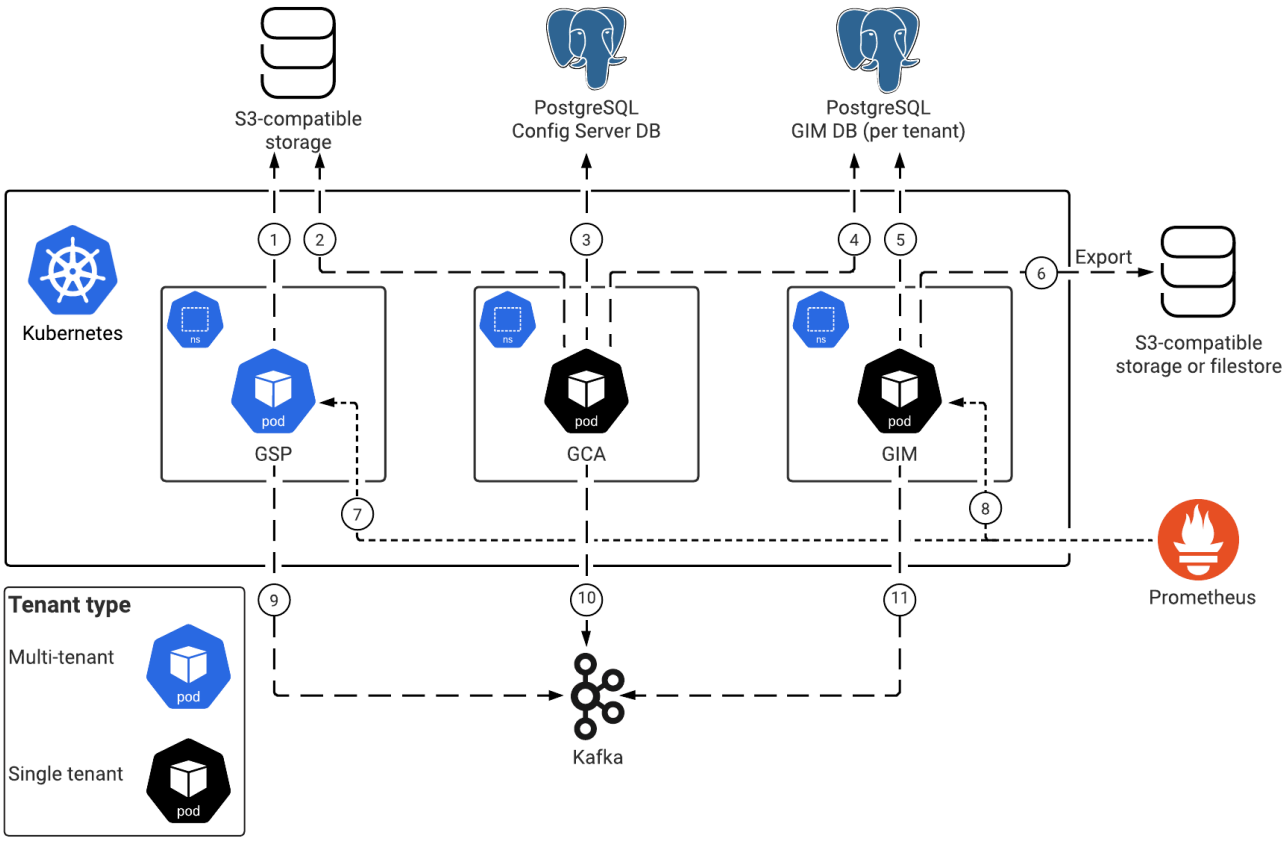
Architecture diagram — Data flows

The numbers on the connection lines refer to the connection numbers in the table that follows the diagrams. The direction of the arrows indicates the direction of data flows.



Architecture diagram — Connections

The numbers on the connection lines refer to the connection numbers in the table that follows the diagrams. 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 Info Mart as a service in the network.



Connections table

The connection numbers refer to the numbers on the connection lines in the diagrams. 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 Info Mart as a service in the network. *Egress* means the Genesys Info Mart service is the source, and *Ingress* means the Genesys Info Mart service is the destination. *Intra-cluster* means the connection is between services in the cluster.

Connection	Source	Destination	Protocol	Port	Connection Classification	Data that travels on this connection
1	GIM Stream Processor	Object storage	HTTPS	443	Egress	GSP state information, checkpoints, and other data used during processing

Connection	Source	Destination	Protocol	Port	Connection Classification	Data that travels on this connection
2	GIM Config Adapter	Object storage	HTTPS	443	Egress	Configuration data stored as a GCA snapshot used during processing
3	GIM Config Adapter	Configuration Database	TCP	5432	Egress	Contact center configuration data, which GCA uses to construct the GCA snapshot used during processing
4	GIM Config Adapter	Info Mart database	TCP	5432	Egress	Configuration data to synchronize the GCA snapshot with configuration data stored in the Info Mart database
5	GIM	Info Mart database	SSL	5432	Egress	Transformed reporting data to be stored in the Info Mart database. GIM also uses this connection to perform database maintenance.
6	GIM	Object storage	HTTP	443	Egress	Reporting data exported from the Info Mart database
7	Prometheus	GIM Stream Processor	HTTP	9249	Ingress	Metrics for monitoring and alerting
8	Prometheus	GIM	HTTP	8249	Ingress	Metrics for monitoring

Connection	Source	Destination	Protocol	Port	Connection Classification	Data that travels on this connection
						and alerting
9	GIM Stream Processor	Kafka	Kafka	9092	Egress	Message bus for receiving and sending reporting data
10	GIM Config Adapter	Kafka	Kafka	9092	Egress	Message bus for sending configuration data
11	GIM	Kafka	Kafka	9092	Egress	Message bus for receiving transformed reporting data

High availability and disaster recovery

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

Related documentation:

-
-

Service	High Availability	Disaster Recovery	Where can you host this service?
Genesys Info Mart	N = 1 (singleton)	Limited active spare	Primary or secondary unit

This information is under development: Flagged items aren't yet confirmed or have info coming soon; Checked items are valid.

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

Content coming soon

Before you begin GSP deployment

Contents

Find out what to do before deploying GIM Stream Processor (GSP).

Related documentation:

-
-

Limitations and assumptions

Not applicable

Download the Helm charts

GIM Stream Processor (GSP) is the only service that runs in the GSP Docker container. The Helm charts included with the GSP release provision GSP and any Kubernetes infrastructure necessary for GSP to run.

See Helm charts and containers for Genesys Info Mart for the Helm chart version you must download for your release.

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

Third-party prerequisites

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

The following table lists the third-party prerequisites for GSP.

Third-party services

Name	Version	Purpose	Notes
Kafka	2.x	Message bus.	The Kafka topics that GSP will consume and produce must exist in the Kafka configuration. See more details below.
Object storage		Persistent or shared data storage, such as Amazon S3, Azure Blob	Both GSP and GCA require persistent storage to store data

Name	Version	Purpose	Notes
		Storage, or Google Cloud Storage.	during processing. You can use the same storage account for both services.
A container image registry and Helm chart repository		Used for downloading Genesys containers and Helm charts into the customer's repository to support a CI/CD pipeline. You can use any Docker OCI compliant registry.	
Command Line Interface		The command line interface tools to log in and work with the Kubernetes clusters.	

Storage requirements

Like GCA, GSP uses S3-compatible storage to store data during processing. GSP stores data such as GSP checkpoints, savepoints, and high availability data. By default, GSP is configured to use Azure Blob Storage, but you can also use S3-compatible storage provided by other cloud platforms. Genesys expects you to use the same storage account for GSP and GCA.

To create S3-compatible storage, do one of the following:

- OpenShift: Create an Object Bucket Claim
- GKE: Create bucket storage

OpenShift: Create an Object Bucket Claim

Create an S3 Object Bucket Claim (OBC) if none exists.

1. Create a **gsp-obc.yaml** file:

    1. ```
       apiVersion: objectbucket.io/v1alpha1
       kind: ObjectBucketClaim
       metadata:
         name: gim
         namespace: gsp
       spec:
         generateBucketName: gim
         storageClassName: openshift-storage.noobaa.io
       ```

2. Execute the command to create the OBC:

    ```
    oc create -f gsp-obc.yaml -n gsp
    ```

    The following Kubernetes resources are created automatically:
    ```

    - An ObjectBucket (OB), which contains the bucket endpoint information, a reference to the OBC, and a reference to the storage class.
- A ConfigMap in the same namespace as the OBC, which contains the endpoint to which applications connect in order to consume the object interface
- A Secret in the same namespace as the OBC, which contains the key-pairs needed to access the bucket.

  Note the following:

- The name of the secret and the configMap are the same as the OBC name.
- The bucket name is created with a randomized suffix.

3. Get S3 data.
   You need to know details of your S3 object in order to populate Helm chart override values for the service.

   1. Execute the following command, where `gim` is the name of the configMap associated with the OBC:

      ```
      oc get cm gim -n gsp -o yaml -o jsonpath={.data}
      ```

      The result shows data such as BUCKET_HOST, BUCKET_NAME, BUCKET_PORT, and so on.

   2. Execute the following commands to get the values of the keys you require for access, where `gim` is the name of the secret associated with the OBC:

      - To get the value of the access key:

        ```
        oc get secret gim -n gsp -o yaml -o jsonpath={.data.AWS_ACCESS_KEY_ID} |
        base64 --decode
        ```

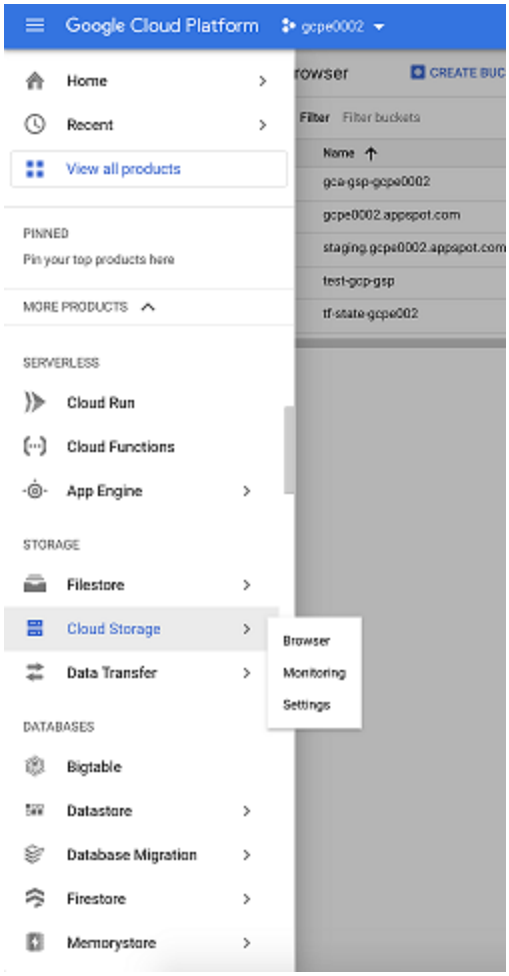
      - To get the value of the secret key:

        ```
        oc get secret gim -n gsp -o yaml -o jsonpath={.data.AWS_SECRET_ACCESS_KEY} |
        base64 --decode
        ```

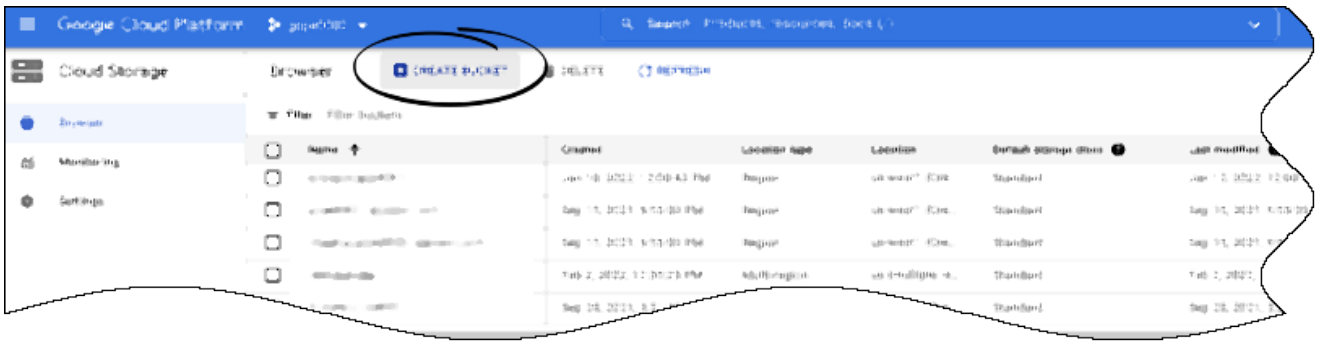
      Use the S3 data to populate storage-related Helm chart override values for the service (for GSP, see Configure S3-compatible storage).

      > **Tip**
      > You can also obtain the S3 data from the OpenShift console: Go to the **Object bucket claims** section under the **Storage** menu, and click on the required OBC resource. The data will be at the bottom of the page.

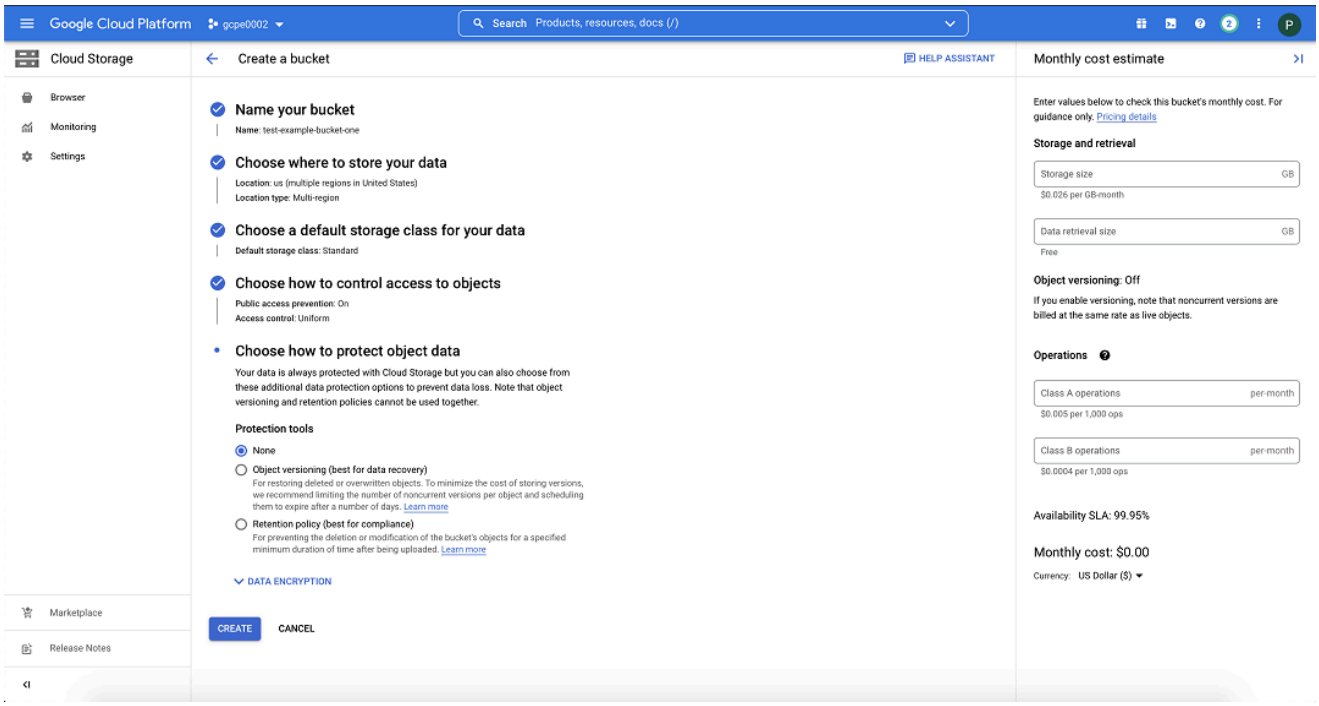
GKE: Create bucket storage



In the Google Cloud Platform (GCP) Cloud Console, select **Cloud Storage** and then choose **Browser** from the drop-down menu.



On the **Cloud Storage > Browser** screen, click **CREATE BUCKET**.



On the **Create a bucket** screen, specify the bucket details:

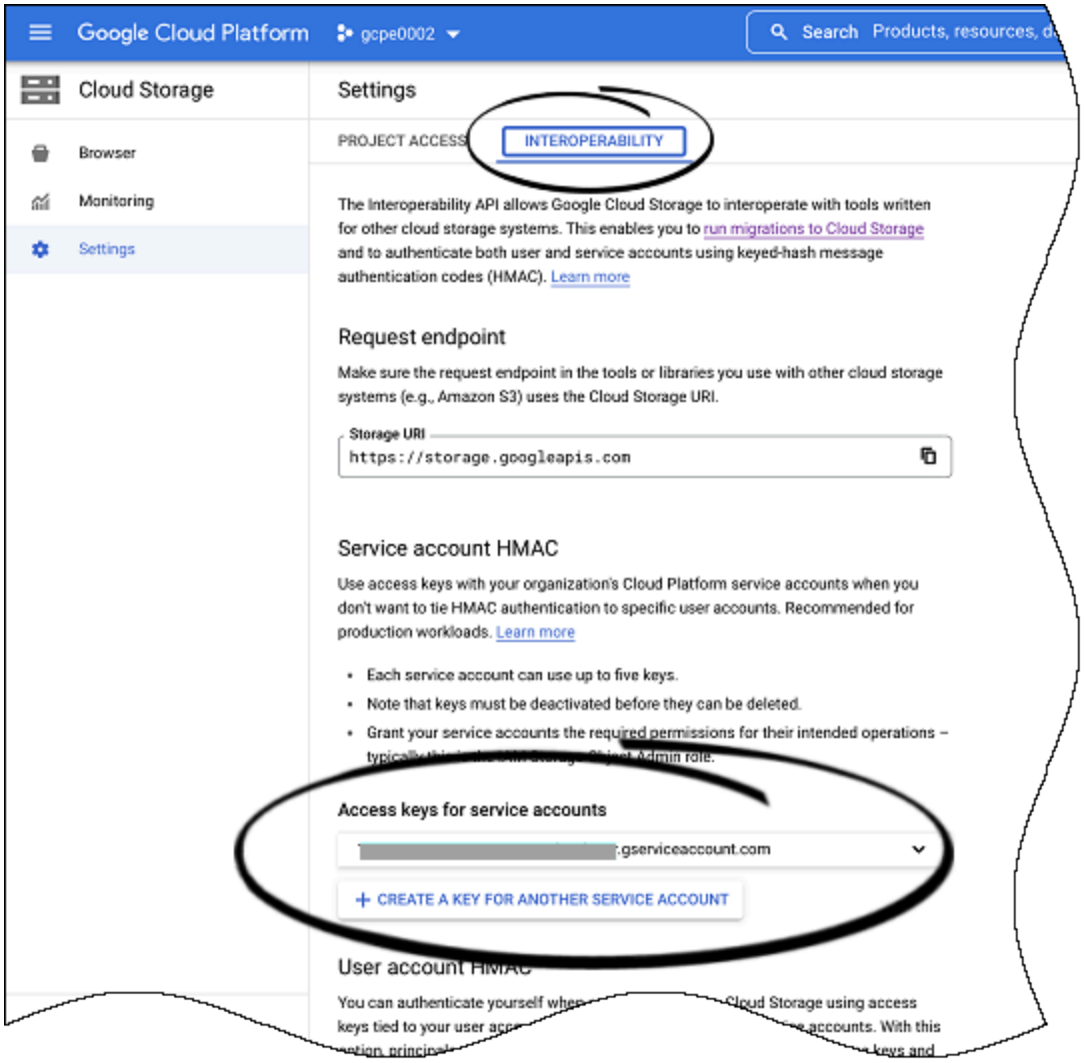
- **Name your bucket** — Enter a unique name for the bucket.
- **Choose where to store your data** — Select the geo-redundancy type (multi-region, dual-region, or region) and location where the data will be stored.

> **Important**
>
> You cannot change the location after the bucket is created.

- **Choose a default storage class for your data** — Select the Standard storage class.
- **Choose how to control access to objects** — Check the box to enforce public access prevention on this bucket, to protect the bucket from being accidentally exposed to the public. When you enforce public access prevention, no one can make data in applicable buckets public through IAM policies or ACLs.
- **Choose how to protect object data** — GSP does not require any protection tools.

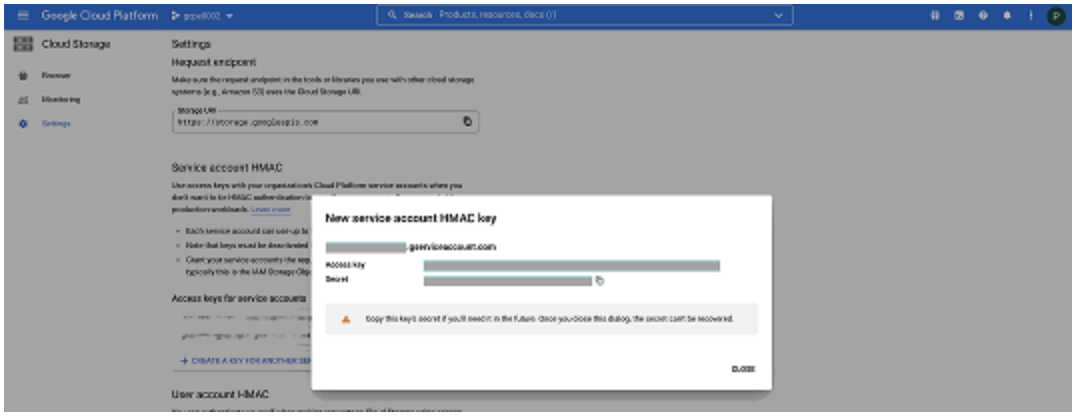
Click **CREATE BUCKET** to create the bucket.

The **Bucket details** screen displays. Verify and, if necessary, edit the details.



It is important to create a storage access key.

1. Go to **Settings > Interoperability**.
2. If the service account you want to use does not already exist, click **CREATE A KEY FOR SERVICE ACCOUNT**. Otherwise, click **CREATE A KEY FOR ANOTHER SERVICE ACCOUNT,** select the service account in the dialog box that displays, and click **CREATE KEY**.
   Note: The service account is the GCP service account associated with the Google Cloud project. The GCP service account enables applications to authenticate and access Google Cloud resources and services. It is not related to the Kubernetes service account created by the Helm chart. Depending on how you want to organize your Google Cloud resources, you can have multiple GCP projects and service accounts.



The Access Key and Secret are generated and displayed in a dialog box. Copy and securely save these details, which you use to populate storage-related Helm chart override values for the applicable service (see Configure S3-compatible storage).

> **Important**
> You cannot recover the Secret once the dialog box is closed.

Network requirements

No special network requirements. Network bandwidth must be sufficient to handle the volume of data to be transferred into and out of Kafka.

Browser requirements

Not applicable

Genesys dependencies

There are no strict dependencies between the Genesys Info Mart services, but the logic of your particular pipeline might require Genesys Info Mart services to be deployed in a particular order. Depending on the order of deployment, there might be temporary data inconsistencies until all the Genesys Info Mart services are operational. For example, GSP looks for the GCA snapshot when it starts; if GCA has not yet been deployed, GSP will encounter unknown configuration objects and resources until the snapshot becomes available.

There are other private edition services you must deploy before Genesys Info Mart. For detailed information about the recommended order of services deployment, see Order of services

deployment.

GDPR support

Not applicable, provided your Kafka retention policies have not been set to more than 30 days. GSP does not store information beyond ephemeral data used during processing.

Kafka configuration

Unless Kafka has been configured to auto-create topics, ensure that the Kafka topics GSP requires have been created in the Kafka configuration. The following table shows the topic names GSP expects to use. An entry in the **Customizable GSP parameter** column indicates that GSP supports using a customized topic name. If you use customized topic names, you must override the applicable values in the **values.yaml** file (see Override Helm chart values).

The topics represent various data domains. If a topic does not exist, GSP will never receive data for that domain. If the topic exists but the customizable parameter value is empty in the GSP configuration, data from that domain will be discarded.

Topic name	Customizable GSP parameter	Description
GSP consumes the following topics:		
voice-callthread		Name of the input topic with voice interactions
voice-agentstate		Name of the input topic with voice agent states
voice-outbound		Name of the input topic with outbound (CX Contact) activity
digital-itx	digitalItx	Name of the input topic with digital interactions
digital-agentstate	digitalAgentStates	Name of the input topic with digital agent states
gca-cfg	cfg	Name of the input topic with configuration data
GSP produces the following topics:		
gsp-ixn	interactions	Name of the output topic for interactions
gsp-sm	agentStates	Name of the output topic for agent states
gsp-outbound	outbound	Name of the output topic for outbound (CX Contact) activity
gsp-custom	custom	Name of the output topic for custom reporting
gsp-cfg	cfg	Name of the output topic for configuration reporting

Configure GSP

Contents

Learn how to configure GIM Stream Processor (GSP).

Related documentation:

-
-

Override Helm chart values

Download the GSP Helm charts from JFrog using your credentials. You must override certain parameters in the GSP **values.yaml** file to provide deployment-specific values for certain parameters.

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

If you want to use arbitrary UIDs in your OpenShift deployment, you must override the **securityContext** settings in the GSP **values.yaml** file, so that no user or group IDs are specified. For details, see Configure security, below.

To enable S3-compatible storage to store data that GSP requires during processing, see Configure S3-compatible storage, below.

At a minimum, you must override the following key entries in the GSP **values.yaml** file:

- `image:`
  `registry` — *the registry from which Kubernetes will pull images (*`pureengage-docker-staging.jfrog.io` *by default)*

  `tag` — *the container image version*
- `imagePullSecrets:`
  `pureengage-docker-dev` or `pureengage-docker-staging` — *the secret from which Kubernetes will get credentials to pull the image from the registry*
- `kafka:`
  `bootstrap` — *the Kafka address to align with the infrastructure Kafka*

If topic names in your Kafka configuration have been customized, you must also modify the `kafka:topic` parameter values to match. For more details about the required Kafka topics, see Kafka configuration.

> **Important**
> Treat your modified **values.yaml** file as source code, which you are responsible to

maintain so that your overrides are preserved and available for reuse when you upgrade.

Configure Kubernetes

Secrets

GSP requires the following secrets:

- `docker-registry` — Credentials to pull the image from the JFrog repository
- `kafka-secrets` — Credentials to access Kafka
- `gsp-s3` — Credentials to access S3-compatible storage

Except for `docker-registry` and `kafka-secrets`, which you must create manually (see the environment setup instructions on Deploy GIM Stream Processor), Helm creates the secrets based on values you specify in the **values.yaml** file.

Config Maps

Helm creates a number of Config Maps based on option values you specify in the **values.yaml** file. There are no Config Maps you can configure directly.

Kubernetes API requirements

GSP uses Apache Flink as the engine for stateful stream processing, with communications handled via the Kubernetes API.

GSP needs permissions to use the following verbs:

- `get`, `list`, `watch`, and `delete` on **jobs** resources in the **batch** API group. GSP uses these commands during upgrade and for a pre-upgrade hook to ensure that the previous version of GSP is stopped before upgrading to the new version.
- `update`, `get`, `create`, `watch`, `patch`, `delete`, `list` on **configmap** resources in the general API group (“”). GSP uses these commands:
  - To enable Flink's Kubernetes high availability (HA) services to function
  - To record the path to the savepoint periodically taken by the **take-savepoint** cron job and by the upgrade hook

Configure security

The security context settings define the privilege and access control settings for pods and containers.

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

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

containerSecurityContext: {}
```

Arbitrary UIDs in OpenShift

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

```
securityContext:
  runAsNonRoot: true
  runAsUser: null
  runAsGroup: 0
  fsGroup: null

containerSecurityContext: {}
```

Configure S3-compatible storage

To enable S3-compatible storage, modify the following entries in the **values.yaml** file:

- `azure:`
  `enabled:` false
- `storage:`
  `gspPrefix` — *the bucket name*

  `gcaSnapshots` — *the bucket name where the GCA snapshot is stored*

  `s3` — *the applicable details defined with the OBC or GCP bucket*

  Note: The `host` parameter is ignored.

OpenShift example

```
azure:
  enabled: false
..
storage:
    host: gspstate{{.Values.short_location}}{{.Values.environment}}.blob.core.windows.net
    #gspPrefix: wasbs://gsp-state@{{ tpl .Values.job.storage.host . }}/{{ .Release.Name }}/
    gspPrefix: "s3p://gim-3f7ac1ab-03b9-445b-ba12-137d4bbc3c38/{{ .Release.Name
}}/"
    #gcaSnapshots: wasbs://gca@{{ tpl .Values.job.storage.host . }}/
    gcaSnapshots: "s3p://gim-3f7ac1ab-03b9-445b-ba12-137d4bbc3c38/
gca/"
    checkpoints: '{{ tpl .Values.job.storage.gspPrefix . }}checkpoints'
    savepoints: '{{ tpl .Values.job.storage.gspPrefix . }}savepoints'
```

```
    highAvailability: '{{ tpl .Values.job.storage.gspPrefix . }}ha'
    s3:
      endpoint: "https://s3.openshift-storage.svc:443"
      accessKey: ""
      secretKey: ""
      pathStyleAccess: "true"
```

GKE example

```
azure:
  enabled: false
...
storage:
    host: gspstate{{.Values.short_location}}{{.Values.environment}}.blob.core.windows.net
    #gspPrefix: wasbs://gsp-state@{{ tpl .Values.job.storage.host . }}/{{ .Release.Name }}/
    gspPrefix: "s3p://test-example-bucket-one/{{ .Release.Name
}}/"
    #gcaSnapshots: wasbs://gca@{{ tpl .Values.job.storage.host . }}/
    gcaSnapshots: "s3p://test-example-bucket-one/
gca/"
    checkpoints: '{{ tpl .Values.job.storage.gspPrefix . }}checkpoints'
    savepoints: '{{ tpl .Values.job.storage.gspPrefix . }}savepoints'
    highAvailability: '{{ tpl .Values.job.storage.gspPrefix . }}ha'
    s3:
      endpoint: "https://storage.googleapis.com:443"
      accessKey: ""
      secretKey: ""
      pathStyleAccess: "true"
```

Deploy GIM Stream Processor

Contents

Learn how to deploy GIM Stream Processor (GSP) into a private edition environment.

Related documentation:

-
-

Assumptions

- The instructions on this page assume you are deploying the service in a service-specific namespace or OpenShift project, 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.

> **Important**
> Make sure to review Before you begin GSP deployment for the full list of prerequisites required to deploy GSP, including creation of the required S3-compatible storage (see Create object storage).

Set up your environment

To prepare your environment for the deployment, complete the steps in this section for:

- OpenShift
- GKE

Environment setup for OpenShift

1. Log in to the OpenShift cluster via CLI on the host where you will run the deployment:

   ```
   oc login --token  --server
   ```

2. (Optional) Check the cluster version:

```
oc get clusterversion
```

3. If the cluster administrator has not already done so, create a new project for GSP:

```
oc new-project gsp
```

4. Set the default project to gsp:

```
oc project gsp
```

5. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

```
oc create secret docker-registry  --docker-server= --docker-username= --docker-
password= --docker-email= -n gsp
```

Create a secret named `kafka-secrets` in order to access Kafka.

When Kafka is deployed without authentication:

```
oc create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"\"} -n gsp
```

When Kafka is deployed with authentication:

```
oc create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"\", \"username\":\"gsp\",\"password\":\"\"} -n gsp
```

For example:

```
oc create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"infra-kafka-cp-
kafka.infra.svc.cluster.local:9092\",\"username\":\"gsp\",\"password\":\"kafka-
password\"} -n gsp
```

Environment setup for GKE

1. Ensure that the gcloud CLI and required Helm version are installed on the host where you will run the deployment.
2. Log in to the GKE cluster from the host where you will run the deployment:

```
gcloud container clusters get-credentials
```

3. If the cluster administrator has not already done so, create a new namespace for GSP:
    1. Create a .json file specifying the namespace metadata. For example, **create-gsp-namespace.json**:

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "gsp",
    "labels": {
      "name": "gsp"
    }
  }
}
```

    2. Execute the following command to create the namespace:

```
kubectl apply -f apply create-gsp-namespace.json
```

3. Confirm namespace creation:

```
kubectl describe namespace gsp
```

4. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

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

5. Create a secret named `kafka-secrets` in order to access Kafka.
   When Kafka is deployed without authentication:

```
kubectl create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"\"} -n gsp
```

   When Kafka is deployed with authentication:

```
kubectl create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"\", \"username\":\"gsp\",\"password\":\"\"} -n gsp
```

   For example:

```
kubectl create secret generic kafka-secrets --from-literal=kafka-
secrets={\"bootstrap\":\"infra-kafka-cp-
kafka.infra.svc.cluster.local:9092\",\"username\":\"gsp\",\"password\":\"kafka-
password\"} -n gsp
```

Deploy

Execute the following command:

```
helm install gsp  -f  -n gsp
```

Validate the deployment

You can consider GSP deployment successful when the pod is running and in Ready state. Genesys Info Mart does not report the Ready state for pods until internal health checks are satisfied and the pods are operational. You can use standard `kubectl` commands like `list` and `get` to verify the successful deployment and readiness status of the Kubernetes objects.

However, from a functional point of view, you cannot validate GSP deployment unless GCA and GIM have been deployed as well. Do not expect consistent data until all three Genesys Info Mart services are up and running. When all three services have been deployed:

1. Make a few test calls employing different routing strategies under different scenarios, and verify that all the calls are correctly captured in the Info Mart database. For example:
   - The calls appear in the interaction-related tables.
   - The calls have been correctly assigned to agents and queues.
2. Review the logs to verify no errors.

3. Monitor the operations dashboard to verify that the services report their status as Ready, and pods are not continually restarting.

Upgrade, rollback, or uninstall GSP

Contents

Learn how to upgrade, rollback or uninstall GIM Stream Processor (GSP).

Related documentation:

-
-

Upgrade GSP

Content coming soon

Execute the following command to upgrade GSP:

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

Rollback GSP

Content coming soon

Uninstall GSP

Execute the following command to uninstall GSP:

```
helm uninstall gsp -n gsp
```

Before you begin GCA deployment

Contents

Find out what to do before deploying GIM Config Adapter (GCA).

Related documentation:

-
-

Limitations and assumptions

Instructions are provided for a single-tenant deployment.

Download the Helm charts

GIM Config Adapter (GCA) and GCA monitoring are the only services that run in the GCA Docker container. The Helm charts included with the GCA release provision GCA and any Kubernetes infrastructure necessary for GCA to run.

See Helm charts and containers for Genesys Info Mart for the Helm chart versions you must download for your release.

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

Third-party prerequisites

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

The following table lists the third-party prerequisites for GCA.

Third-party services

Name	Version	Purpose	Notes
Kafka	2.x	Message bus.	GCA publishes configuration data to the `gca-cfg` topic, which GSP consumes. The topic must exist in your Kafka configuration.

Name	Version	Purpose	Notes
Object storage		Persistent or shared data storage, such as Amazon S3, Azure Blob Storage, or Google Cloud Storage.	Both GCA and GSP require object storage to store data during processing. You can use the same storage account for both services.
A container image registry and Helm chart repository		Used for downloading Genesys containers and Helm charts into the customer's repository to support a CI/CD pipeline. You can use any Docker OCI compliant registry.	
Command Line Interface		The command line interface tools to log in and work with the Kubernetes clusters.	

Storage requirements

GCA uses object storage to store the GCA snapshot during processing. Like GSP, GCA supports using S3-compatible storage provided by OpenShift and Google Cloud Platform (GCP), and Genesys expects you to use the same storage account for GSP and GCA. If you want to use separate storage for GCA, follow the Create object storage instructions for GSP to create similar S3-compatible storage for GCA.

Network requirements

No special network requirements.

Browser requirements

Not applicable

Genesys dependencies

- Voice Tenant Service, which enables GCA to access the Configuration Server database. You must deploy the Voice Tenant Service before you deploy GCA.
    - Ensure that an appropriate user account is available for GCA to use to access the Configuration Database. The GCA user account requires at least read permissions.

    - You must also have your Tenant ID information available.
- There are no strict dependencies between the Genesys Info Mart services, but the logic of your particular pipeline might require Genesys Info Mart services to be deployed in a particular order. Depending on the order of deployment, there might be temporary data inconsistencies until all the Genesys Info Mart services are operational. For example, GSP looks for the GCA snapshot when it starts; if GCA has not yet been deployed, GSP will encounter unknown configuration objects and resources until the snapshot becomes available.

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

GDPR support

Not applicable. GCA does not store information beyond an ephemeral snapshot.

Configure GCA

Contents

Learn how to configure GIM Config Adapter (GCA).

Related documentation:

-
-

Override Helm chart values

Download the gca and gca-monitoring Helm charts from JFrog using your credentials. You must override certain parameters in the GCA **values.yaml** file to provide deployment-specific values for certain parameters.

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

If you want to use arbitrary UIDs in your OpenShift deployment, you must override the **securityContext** settings in the GCA **values.yaml** file, so that no user or group IDs are specified. For details, see Configure security, below.

To enable S3-compatible storage to store the GCA snapshot, see Configure S3-compatible storage, below.

At a minimum, you must override the following key entries in the GCA **values.yaml** file:

- `image:`
  `registry` — *the registry from which Kubernetes will pull images (*`pureengage-docker-staging.jfrog.io` *by default)*

  `tag` - *the container image version*
- `tenant_id` - *the TenantID of the tenant in use*
- `cfgdb` - *the applicable details for the Configuration Database, created before you deployed the Tenant service*
  `name` - *the name of the database*

  `host` - *the host on which the DBMS is running*

  `username` - *the user account for GCA to access the database. The user account must have at least read permissions.*

  `password` - *the password for the user account*
- `gimdb` - *the applicable details for the Info Mart database*
  `name` - *the name of the database*

  `host` - *the host on which the DBMS is running*

  `username` - *the user account created when you created the database (see Create the Info Mart*

database)

`password` - *the password for the user account*

- `kafka:`
  `bootstrap` - *the Kafka address to align with the infrastructure Kafka*

  `password` - *the Kafka password, if Kafka requires authentication*

Note: `tenant_id` and `kafka:password` (optional) are currently not included in the **values.yaml** file. Either add these parameters to your customized **values.yaml** file or else specify them in the command line when you install the Helm chart.

> **Important**
>
> Treat your modified **values.yaml** file as source code, which you are responsible to maintain so that your overrides are preserved and available for reuse when you upgrade.

Configure Kubernetes

Secrets

GCA requires the following secrets:

- `docker-registry` — Credentials to pull the image from the JFrog repository
- `{{.Release.Name}}-kafka-secrets` — Credentials to access Kafka
- `{{.Release.Name}}-cfgdb-secrets` — Credentials to access the Configuration Database
- `{{.Release.Name}}-gimdb-secrets` — Credentials to access the Info Mart database
- `{{.Release.Name}}-storage-secret` — Credentials to access optional S3-compatible storage (for Data Export)

Except for `docker-registry`, which you must create manually (see the environment setup instructions on Deploy GIM Config Adapter), Helm creates the secrets based on values you specify in the **values.yaml** file.

Config Maps

There are no Config Maps you can configure directly.

Configure security

The security context settings define the privilege and access control settings for pods and containers.

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

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

containerSecurityContext: {}
```

Arbitrary UIDs in OpenShift

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

```
securityContext:
  runAsNonRoot: true
  runAsUser: null
  runAsGroup: 0
  fsGroup: null

containerSecurityContext: {}
```

Configure S3-compatible storage

If you are using S3-compatible object storage on OpenShift or GCP to store the GCA snapshot, modify the following `storage: s3` entries in the **values.yaml** file:

- `bucket` — *the bucket name*
- `gcaSnapshots` — *the volume or folder in the bucket where the GCA snapshot is stored*
- `accessKey` — *the access key created when you created the bucket*
- `secretKey` — *the secret created when you created the bucket*
- `endPoint` — *the bucket host*

OpenShift example

```
storage:
  ...
  s3:
    bucket: "gim-3f7ac1ab-03b9-445b-ba12-137d4bbc3c38"
    gcaSnapshots: "/gca"
    accessKey: ""
    secretKey: ""
    useSSL: true
    endPoint: "s3.openshift-storage.svc"
    port: 443
    Insecure: true
```

GKE example

```
storage:
  ...
  s3:
    bucket: "test-example-bucket-one"
    gcaSnapshots: "/gca"
    accessKey: ""
    secretKey: ""
    useSSL: true
    endPoint: "storage.googleapis.com"
    port: 443
    Insecure: true
```

Deploy GIM Config Adapter

Contents

Learn how to deploy GIM Config Adapter (GCA) into a private edition environment.

Related documentation:

-
-

Assumptions

- The instructions on this page assume you are deploying the service in a service-specific namespace or OpenShift project, 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.

> **Important**
> Make sure to review Before you begin GCA deployment for the full list of prerequisites required to deploy GCA, including creation of the required S3-compatible storage (see Create object storage).

Set up your environment

To prepare your environment for the deployment, complete the steps in this section for:

- OpenShift
- GKE

Environment setup for OpenShift

1. Log in to the OpenShift cluster from the remote host via CLI:

   ```
   oc login --token  --server
   ```

2. (Optional) Check the cluster version:

```
oc get clusterversion
```

3. If the cluster administrator has not already done so, create a new project for GCA:

```
oc new-project gca
```

4. Set the default project to GCA:

```
oc project gca
```

5. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

```
oc create secret docker-registry  --docker-server= --docker-username= --docker-
 password= --docker-email= -n gca
```

Environment setup for GKE

1. Ensure that the gcloud CLI and required Helm version are installed on the host where you will run the deployment.
2. Log in to the GKE cluster from the host where you will run the deployment:

```
gcloud container clusters get-credentials
```

3. If the cluster administrator has not already done so, create a new namespace for GCA:
   1. Create a .json file specifying the namespace metadata. For example, **create-gca-namespace.json**:

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "gca",
    "labels": {
      "name": "gca"
    }
  }
}
```

   2. Execute the following command to create the namespace:

```
kubectl apply -f apply create-gca-namespace.json
```

   3. Confirm namespace creation:

```
kubectl describe namespace gca
```

4. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

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

Deploy

Execute the following command to install GCA:

```
helm upgrade --install  -f  -n gca
```

Execute the following command to install GCA monitoring:

```
helm upgrade --install gca-monitoring  -n gca
```

Validate the deployment

You can consider GCA deployment successful when the pod is running and in ready state. Genesys Info Mart does not report the ready state for pods until internal health checks are satisfied and the pods are operational. You can use standard `kubectl` commands like `list` and `get` to verify the successful deployment and readiness status of the Kubernetes objects, including connection to the database.

However, from a functional point of view, you cannot validate deployment of GCA unless GSP and GIM have been deployed as well. Do not expect consistent data until all three Genesys Info Mart services are up and running. For more details about functional checks you can perform to validate GCA deployment, see the equivalent validation section on the Deploy GIM Stream Processor page.

Upgrade, rollback, or uninstall GCA

Contents

Learn how to upgrade, rollback or uninstall GIM Config Adapter (GCA).

Related documentation:

-
-

Upgrade GCA

Content coming soon

Execute the following commands to upgrade GCA and GCA monitoring:

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

Rollback GCA

Content coming soon

Uninstall GCA

Execute the following command to uninstall GCA:

```
helm uninstall gca -n gca
```

Execute the following command to uninstall GCA monitoring:

```
helm uninstall gca-monitoring -n gca
```

Before you begin GIM deployment

Contents

Find out what to do before deploying GIM.

Related documentation:

-
-

Limitations and assumptions

Instructions are provided for a single-tenant deployment.

Download the Helm charts

GIM and GIM monitoring are the only services that run in the GIM Docker container. The Helm charts included with the GIM release provision GIM and any Kubernetes infrastructure necessary for GIM to run.

See Helm charts and containers for Genesys Info Mart for the Helm chart version you must download for your release.

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

Third-party prerequisites

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

The following table lists the third-party prerequisites for GIM.

Third-party services

Name	Version	Purpose	Notes
PostgreSQL	11.x	Relational database.	You must create the Info Mart database (see below).
Kafka	2.x	Message bus.	The Kafka topics GIM will consume must exist in the Kafka configuration. GIM consumes the topics

Name	Version	Purpose	Notes
			GSP produces. For more information, see Kafka configuration.
Object storage		Persistent or shared data storage, such as Amazon S3, Azure Blob Storage, or Google Cloud Storage.	(Optional) For the Data Export feature, GIM uses object storage to store the exported data. Alternatively, you can elect to store the exported data in a directory on a local machine.
A container image registry and Helm chart repository		Used for downloading Genesys containers and Helm charts into the customer's repository to support a CI/CD pipeline. You can use any Docker OCI compliant registry.	
Command Line Interface		The command line interface tools to log in and work with the Kubernetes clusters.	

Storage requirements

GIM uses PostgreSQL for the Info Mart database and, optionally, uses object storage to store exported Info Mart data.

PostgreSQL — the Info Mart database

The Info Mart database stores data about agent and interaction activity, Outbound Contact campaigns, and other services usage in your contact center. A subset of tables and views created, maintained, and populated by Reporting and Analytics Aggregates (RAA) provides the aggregated data on which Genesys CX Insights (GCXI) reports are based.

A sizing calculator for Genesys Multicloud CX private edition is under development. In the meantime, the interactive tool available for on-premises deployments might help you estimate the size of your Info Mart database, see *Genesys Info Mart 8.5 Database Size Estimator*.

Genesys recommends a minimum 3 IOPS per GB.

For information about creating the Info Mart database, see Create the Info Mart database.

Create the Info Mart database

Use any database management tool to create the Info Mart ETL database and user.

1. Create the database.
2. Create a user for the Genesys Info Mart services to use, and grant full permissions to the user for that database.
   This user's account is used by Genesys Info Mart jobs to access the Info Mart database schema.

   The Info Mart schema name is `public`.

   > **Important**
   > Make a note of the database and user details, which you use to populate database-related Helm chart override values for GIM and GCA.

Object storage — Data Export packages

The GIM Data Export feature enables you to export data from your Info Mart database. Unless you elect to store your exported data in a local directory, your Info Mart data is exported to an object store. GIM supports export to Azure Blob Storage or S3-compatible storage provided by OpenShift and Google Cloud Platform (GCP).

If you want to use S3-compatible storage, follow the Create object storage instructions for GSP to create the S3-compatible storage for GIM.

> **Important**
> GSP and GCA use object storage to store data during processing. For safety and security reasons, Genesys strongly recommends that you use a dedicated object storage account for the GIM persistent storage, and do not share the storage account created for GSP and GCA.

As another alternative, you can configure GIM to store your exported data in a local directory. In this case, you do not need to create the object storage.

Network requirements

No special network requirements.

Browser requirements

Not applicable

Genesys dependencies

- You must have your Tenant ID information available.
- There are no strict dependencies between the Genesys Info Mart services, but the logic of your particular pipeline might require Genesys Info Mart services to be deployed in a particular order. Depending on the order of deployment, there might be temporary data inconsistencies until all the Genesys Info Mart services are operational. For example, GCA might try to access the Info Mart database to synchronize configuration data; if GIM has not yet been deployed, the Info Mart database will be empty.

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

GDPR support

GIM does not yet support GDPR compliance. For details about the Info Mart database tables and columns that potentially contain personally identifiable information (PII), see the description of the CTL_GDPR_HISTORY table in the Genesys Info Mart on-premises documentation.

Configure GIM

Contents

Learn how to configure GIM.

Related documentation:

-
-

Override Helm chart values

Download the gim and gim-monitoring Helm charts from JFrog using your credentials. You must override certain parameters in the gim **values.yaml** file to provide deployment-specific values for certain parameters.

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

If you want to use arbitrary UIDs in your OpenShift deployment, you must override the **securityContext** settings in the GIM **values.yaml** file, so that no user or group IDs are specified. For details, see Configure security, below.

To enable S3-compatible storage for the Data Export packages, see Configure S3-compatible storage, below.

At a minimum, you must override the following key entries in the GIM **values.yaml** file:

- `image:`
  `registry` — *the registry from which Kubernetes will pull images (*`pureengage-docker-staging.jfrog.io` *by default)*

  `tag` - *the container image version*
- `tenant_id` - *the TenantID of the tenant in use*
- `imagePullSecrets:`
  `pureengage-docker-dev` or `pureengage-docker-staging` — *the secret from which Kubernetes will get credentials to pull the image from the registry*
- `kafka:`
  `bootstrap` - *the Kafka address to align with the infrastructure Kafka*

  `password` - *the Kafka password, if Kafka requires authentication*
- `db` - *the applicable details for the Info Mart ETL database you created (see Create the Info Mart database)*

Note: `tenant_id` and `kafka:password` (optional) are currently not included in the **values.yaml** file. Either add these parameters to your customized **values.yaml** file or else specify them in the command line when you install the Helm chart.

If topic names in your Kafka configuration have been customized, you must also modify the `kafka:topic` parameter values to match. For more details about the required Kafka topics, see Kafka configuration.

> **Important**
>
> Treat your modified **values.yaml** file as source code, which you are responsible to maintain so that your overrides are preserved and available for reuse when you upgrade.

Configure Kubernetes

Secrets

GIM requires the following secrets:

- `docker-registry` — Credentials to pull the image from the JFrog repository
- `kafka-secrets` — Credentials to access Kafka
- `gim-secrets` — Credentials to access the Info Mart database
- `s3-storage-secrets` — Credentials to access optional S3-compatible storage (for Data Export)

Except for `docker-registry`, which you must create manually (see the environment setup instructions on Deploy GIM), Helm creates the secrets based on values you specify in the **values.yaml** file.

Config Maps

Helm creates a number of Config Maps based on option values you specify in the **values.yaml** file. There are no Config Maps you can configure directly.

Configure security

The security context settings define the privilege and access control settings for pods and containers.

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

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

containerSecurityContext: {}
```

Arbitrary UIDs in OpenShift

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

```
securityContext:
  runAsNonRoot: true
  runAsUser: null
  runAsGroup: 0
  fsGroup: null

containerSecurityContext: {}
```

Configure S3-compatible storage

If you are using S3-compatible object storage on OpenShift or GCP to store the Data Export packages, modify the following entries in the **values.yaml** file:

- `s3_storage_enabled:` true
- `s3_storage:`
  `account:`

  `accessKey` — *the access key created when you created the bucket*

  `secretKey` — *the secret created when you created the bucket*

  `region` — *the region in which the bucket was created*

  `entryPoint` — *URL to access bucket storage*
- `gim_export:`
  `output_directory` — *the bucket name*

The `s3_storage` parameters are used to construct the **s3_storage_secrets** secret.

OpenShift example

```
gim_export:
  ...
  output_directory:                       "gim-3f7ac1ab-03b9-445b-ba12-137d4bbc3c38"
  ...
s3_storage_enabled: true
s3_storage:
  account:
    accessKey: ""
    secretKey: ""
    region: ""
    entrypoint: "s3.openshift-storage.svc"
```

GKE example

```
gim_export:
  ...
  output_directory:                       "test-example-bucket-one"
```

```
  ...
s3_storage_enabled: true
s3_storage:
  account:
    accessKey: ""
    secretKey: ""
    region: ""
    entrypoint: "storage.googleapis.com"
```

Deploy GIM

Contents

Learn how to deploy GIM (GIM) into a private edition environment.

Related documentation:

-
-

Assumptions

- The instructions on this page assume you are deploying the service in a service-specific namespace or OpenShift project, 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.

> **Important**
> Make sure to review Before you begin GIM deployment for the full list of prerequisites required to deploy GIM, including creation of the Info Mart database and, if applicable, S3-compatible storage for your exported data (see (see Create object storage).

Set up your environment

To prepare your environment for the deployment, complete the steps in this section for:

- OpenShift
- GKE

Environment setup for OpenShift

1. Log in to the OpenShift cluster via CLI on the host where you will run the deployment:

   ```
   oc login --token  --server
   ```

2. (Optional) Check the cluster version:

```
oc get clusterversion
```

3. If the cluster administrator has not already done so, create a new project for GIM:

```
oc new-project gim
```

4. Set the default project to GIM:

```
oc project gim
```

5. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

```
oc create secret docker-registry  --docker-server= --docker-username= --docker-
password= --docker-email= -n gim
```

Environment setup for GKE

1. Ensure that the gcloud CLI and required Helm version are installed on the host where you will run the deployment.
2. Log in to the GKE cluster from the host where you will run the deployment:

```
gcloud container clusters get-credentials
```

3. If the cluster administrator has not already done so, create a new namespace for GIM:
   1. Create a .json file specifying the namespace metadata. For example, **create-gim-namespace.json**:

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "gim",
    "labels": {
      "name": "gim"
    }
  }
}
```

   2. Execute the following command to create the namespace:

```
kubectl apply -f apply create-gim-namespace.json
```

   3. Confirm namespace creation:

```
kubectl describe namespace gim
```

4. Create a secret for docker-registry in order to pull images from the Genesys JFrog repository:

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

Deploy

Execute the following command to install GIM:

```
helm upgrade --install gim  -f  -n gim
```

Execute the following command to install GIM monitoring:

```
helm upgrade --install gim-monitoring  -n gim
```

When the GIM service starts, it connects to the Info Mart database and checks if the GIM schema has been initialized. If the schema has not been initialized, the service runs a script to initialize the GIM schema.

Validate the deployment

You can consider GIM deployment successful when the pod is running and in Ready state. Genesys Info Mart does not report the Ready state for pods until internal health checks are satisfied and the pods are operational. For example, GIM does not report Ready until the first transformation job has completed successfully. In other words, if the GIM pod is running and in Ready state, it means that GIM successfully started, connected to Kafka and the Info Mart database, initialized the Info Mart database, and completed the first ETL cycle.

You can use standard `kubectl` commands like `list` and `get` to verify the successful deployment and readiness status of the Kubernetes objects, including connection to the database.

However, from a functional point of view, you cannot validate deployment of the GIM service and database unless GCA and GSP have been deployed as well. Do not expect consistent data until all three Genesys Info Mart services are up and running. For more details about functional checks you can perform to validate GIM deployment, see the equivalent validation section on the Deploy GIM Stream Processor page.

Upgrade, rollback, or uninstall GIM

Contents

Learn how to upgrade, rollback or uninstall GIM.

Related documentation:

-
-

Upgrade GIM

Content coming soon

Execute the following commands to upgrade GIM and GIM monitoring:

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

Rollback GIM

Content coming soon

Uninstall the GIM ETL service

Execute the following command to uninstall GIM:

```
helm uninstall gim -n gim
```

Execute the following command to uninstall GIM monitoring:

```
helm uninstall gim-monitoring -n gim
```

Observability in Genesys Info Mart

Contents

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

Related documentation:

-
-

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 Info Mart metrics, see:

- GIM Config Adapter metrics
- GIM metrics
- GIM Stream Processor metrics

See also System metrics.

Enable monitoring

The Genesys Info Mart services use PodMonitor custom resource definitions (CRDs), which are defined in the Helm charts by default. No additional service-level configuration is required to enable monitoring. See [[PrivateEdition/Current/Operations/General_instructions_on_monitoring_solution_for_services|]] for information about enabling monitoring for your private edition solution.

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
GIM Config Adapter	PodMonitor	9249	See selector details on the GIM Config Adapter metrics and alerts page	30 seconds

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
GIM	PodMonitor	8249	See selector details on the GIM metrics and alerts page	30 seconds
GIM Stream Processor	PodMonitor	9249	See selector details on the GIM Stream Processor metrics and alerts page	30 seconds

Configure metrics

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

The Metrics pages linked to above show some of the metrics the Genesys Info Mart services expose. You can also query Prometheus directly or via a dashboard to see all the metrics available from the Genesys Info Mart services.

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 Info Mart alerts, see:

- GIM Config Adapter alerts
- GIM alerts
- GIM Stream Processor alerts

Configure alerts

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

Logging

Genesys Info Mart uses a structured logging approach and outputs logs to standard output (stdout).

You can set the log levels for the Genesys Info Mart services in the respective values.yaml files. By default, the GSP and GIM logs are at the INFO level, while GCA logs are at DEBUG level.

See Logging overview and approaches and related links in the *Operations* guide for more information about logging in private edition, including deployment information and how you can use log collectors (such as Elasticsearch) to extract the logs and viewing tools (such as Kibana) to visualize the data.

GSP metrics and alerts

Find the metrics GSP exposes and the alerts defined for GSP.

Related documentation:

-

Contents

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval

GSP	PodMonitor	9249	Endpoint: /
			Selector:
			`matchLabels:`
			`  app: {{ template "gsp.fullname" . }}`
			where the value of `gsp.fullname` depends on deployment parameters such as Helm release name, `.Values.fullnameOverride`, and `.Values.nameOverride`.
				30 seconds

See details about:

- GSP metrics
- GSP alerts

Metrics

GSP exposes some standard Apache Flink and Kafka metrics as well as Genesys-defined metrics, which are exposed via the Flink API. Therefore, all GSP metrics start with the prefix **flink_** but in some cases the values are calculated by GSP.

You can query Prometheus directly to see all the metrics Flink and the Flink Kafka connector expose through GSP.

- For full information about the standard Flink metrics, see the Apache Flink documentation.
- For full information about the Kafka metrics, see the Apache Kafka or Confluent Kafka documentation.

The following metrics are likely to be particularly useful. The naming convention is _. Genesys does not commit to maintain other currently available GSP metrics not documented on this page.

Metric and description	Metric details	Indicator of
flink_taskmanager_job_task_operator_errors_numInvalidRecords Number of invalid input records.	**Unit:** **Type:** Gauge **Label:** **Sample value:** 0	Error
flink_jobmanager_numRunningJobs Number of running Flink jobs. If less than 1, there is a problem.	**Unit:** **Type:** Gauge **Label:** **Sample value:** 1	Error
flink_taskmanager_job_task_operator_user_errors_numOversizedMessages		

Metric and description	Metric details	Indicator of
Number of messages exceeding the **max.request.size** Kafka option.	**Type:** Gauge **Label:** • operator_name **Sample value:** 0	
flink_taskmanager_job_task_operator_tenant_error_total Number of issues encountered, such as errors or warnings.	**Unit:** **Type:** Gauge **Label:** • operator_name • tenant • error **Sample value:**	Error
flink_taskmanager_job_task_operator_currentInputWatermark The last watermark received by this operator/task, in milliseconds since the Unix Epoch (00:00:00 UTC on 1 January 1970). **Note:** For operators/tasks with two inputs, this is the earlier of the last received watermarks.	**Unit:** milliseconds **Type:** Gauge **Label:** • operator_name **Sample value:**	Latency

flink_taskmanager_job_task_operator_currentOutputWatermark
The last watermark this operator has emitted, in milliseconds since the Unix Epoch.
	Unit: milliseconds
	Type: Gauge
	Label:
	• operator_name:
	  • Sink:_Agent_State_Facts
	  • Sink:_Interaction_Facts
	Sample value:
		Latency

flink_taskmanager_job_task_operator_records_lag_max The maximum lag in terms of the number of records for any partition in this window. An increasing value over time is your best indication that the consumer group is not keeping up with the producers.	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Latency
flink_taskmanager_job_task_operator_records_consumed_rate The average number of records consumed per second.	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_taskmanager_job_task_operator_numCallsCreated Total number of EventCallCreated events GSP received since it started processing.	**Unit:** **Type:** Gauge **Label:**	Traffic

Metric and description	Metric details	Indicator of
	Sample value:	
flink_taskmanager_job_task_operator_numCallsCreatedPerSecond Number of EventCallCreated events per second (CPS).	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_taskmanager_job_task_operator_numThreadsCreated Total number of CallThreads GSP received since it started processing.	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_taskmanager_job_task_operator_numCallThreadsCreatedPerSecond Number of CallThreads per second (CTHPS).	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_taskmanager_job_task_operator_numChainsProcessed Total number of EventOCSChainStartProcessing events GSP received since it started processing.	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_taskmanager_job_task_operator_numChainsProcessedPerSecond Number of EventOCSChainStartProcessing events per second (CPS).	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Traffic
flink_(job|task)manager_Status_JVM_CPU_Load The recent CPU usage for the JVM process. The value is a double in the [0.0,1.0] interval, where a value of 0.0 means that none of the CPUs were running threads from the JVM process, while a value of 1.0 means that all CPUs were actively running threads from the JVM 100% of the time during the recent period being observed. A negative value means usage data is not available. For more information, see https://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html#getProcessCpuLoad().	**Unit:** **Type:** Gauge **Label:** • pod **Sample value:**	Saturation
flink_(job|task)manager_Status_JVM_Memory_Direct_TotalCapacity The total capacity of all buffers in the direct buffer pool.	**Unit:** bytes **Type:** Gauge **Label:** • pod **Sample value:**	Saturation
flink_(job|task)manager_Status_JVM_Memory_Direct_MemoryUsed The amount of memory used by the JVM for the direct buffer pool.	**Unit:** bytes **Type:** Gauge **Label:**	Saturation

Metric and description	Metric details	Indicator of
	• pod **Sample value:**	
flink_(job|task)manager_Status_JVM_Memory_NonHeap_Max The maximum amount of non-heap memory that can be used for memory management.	**Unit:** bytes **Type:** Gauge **Label:** • pod **Sample value:**	Saturation
flink_(job|task)manager_Status_JVM_Memory_NonHeap_Used The amount of non-heap memory currently used.	**Unit:** bytes **Type:** Gauge **Label:** • pod **Sample value:**	Saturation
flink_(job|task)manager_Status_JVM_Memory_Heap_Max The maximum amount of heap memory that can be used for memory management.	**Unit:** bytes **Type:** Gauge **Label:** • pod **Sample value:**	Saturation
flink_(job|task)manager_Status_JVM_Memory_Heap_Used The amount of heap memory currently used.	**Unit:** bytes **Type:** Gauge **Label:** • pod **Sample value:**	Saturation

Alerts

The alerts are based on Flink and Kubernetes cluster metrics.

The following alerts are defined for GSP.

Alert	Severity	Description	Based on	Threshold
GspFlinkJobDown	Critical	Triggered when the GSP Flink job is not running (number of running jobs	flink_jobmanager_numRunningJobs	For 5 minutes

Alert	Severity	Description	Based on	Threshold
		equals to 0 or metric is not available)		
GspOOMKilled	Critical	Triggered when a GSP pod is restarted because of OOMKilled	kube_pod_container_status_restarts_total	0
GspNoTmRegistered	Critical	Triggered when there are no registered TaskManagers (or metric not available)	flink_jobmanager_numRegisteredTaskManagers	For 5 minutes
GspUnknownPerson	High	Triggered when GSP encounters unknown person(s)	flink_taskmanager_job_task_operator_tenant_error_tota	For 5 minutes

GCA metrics and alerts

Find the metrics GCA exposes and the alerts defined for GCA.

Related documentation:

-

Contents

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval

GCA	PodMonitor	9249	`selector:`
			`  matchLabels:`
			`  app: {{ template "fullname" . }}`
			
			where the value of `fullname` depends on `.Values.tenant_id`.
				30 seconds

See details about:

- GCA metrics
- GCA alerts

Metrics

Use standard Kubernetes metrics to monitor the GCA service. For information about standard Kubernetes and other system metrics to monitor services, see System metrics.

Alerts

Alerts are based on Kubernetes cluster metrics.

The following alerts are defined for GCA.

Alert	Severity	Description	Based on	Threshold
GcaPodCrashLooping	Critical	Triggered when a GCA pod is crash looping.	kube_pod_container_status_restarts_total	The restart rate is greater than 0 for 5 minutes
GcaOOMKilled	Critical	Triggered when a GCA pod is restarted because of OOMKilled.	kube_pod_container_status_restarts_total and kube_pod_container_status_last_terminated_reason	1

GIM metrics and alerts

Find the metrics GIM exposes and the alerts defined for GIM.

Related documentation:

-

Contents

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval

GIM	PodMonitor	8249	Endpoint: /metrics
			Selector:
			`selector:`
			`  matchLabels:`
			`  app: {{ template "fullname" . }}`
			where the value of `fullname` depends on `.Values.tenant_id`.
				30 seconds

Metrics

The following metrics are defined for the GIM service.

Metric and description	Metric details	Indicator of
gim_server_state The state of the GIM server. Valid values are: • 0 — starting • 1 — running • -1 — failing once • -2 — failing twice • -3 — failing three or more times in a row	**Unit:** **Type:** Gauge **Label:** • applicationName • gim_version • built • schema_version **Sample value:**	Error
gim_failed_jobsteps Number of failed job steps.	**Unit:** **Type:** Gauge **Label:** **Sample value:**	Error
gim_number_failed_kafkajob Number of times the KAFKA step in the transform job failed.	**Unit:** **Type:** Gauge **Label:** • applicationName **Sample value:**	Error
gim_issues_total Number of encountered issues.	**Unit:** **Type:** Counter **Label:**	Error

Metric and description	Metric details	Indicator of
	• applicationName • issue **Sample value:**	
gim_kafka_timestamp_millis Maximum Kafka timestamp per topic per partition, in milliseconds since the Unix Epoch.	**Unit:** milliseconds **Type:** Gauge **Label:** **Sample value:**	Latency
gim_kafka_timestamp_behind Kafka latency per topic per partition, in milliseconds. Latency is calculated as (current system time) minus (maximum timestamp received). Therefore, this metric can grow if a topic or partition is idle and no records are received.	**Unit:** milliseconds **Type:** Gauge **Label:** • applicationName • topic • partition **Sample value:**	Latency

Alerts

No alerts are defined for GIM.