Genesys Customer Experience Insights Private Edition Guide

Deploy Genesys Customer Experience Insights

2/25/2026

Contents

Learn how to deploy Genesys Customer Experience Insights (GCXI) 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.

> **Important**
>
> Review Before you begin deploying GCXI for the full list of prerequisites required to deploy GCXI.

Prepare to deploy GCXI

Before you begin, ensure that:

- You are logged in to your OpenShift, GKE, or AKS cluster.
- For HA deployments, at least two worker nodes are available for GCXI pods.
- Each Worker machine meets the following minimum requirements:
  - 64-bit compatible CPU architecture. (2 or more CPUs.)

    - 10 GB for each GCXI container, and 2 GB for the PostgreSQL container. Production deployments commonly reserve between 16 GB and 64 GB of RAM for each container.
    - 40 GB of available disk space, if you are loading images from a repository.
- Helm-3 is installed on the host where the deployment will run.
- Images **gcxi** and **gcxi_control** are tagged and loaded on the registry. During deployment, OpenShift pulls the images from the registry to each OpenShift worker node.
- On each worker node, values are set for `kernel.sem` and `vm.max_map_count`, as required by MicroStrategy. For example:

```
echo "kernel.sem = 250 1024000 250 4096" >> /etc/sysctl.conf echo "vm.max_map_count =
 5242880" >> /etc/sysctl.conf sysctl -p
```

PVCs required by GCXI

Mount Name	Mount Path (inside container)	Description	Access Type	Default Mount Point on the Host (you can change this using values; ensure that the indicated directory pre-exists on your host, to accommodate the local provisioner)	Shared across Nodes?	Required Node Label (applies to default Local PVs setup)
gcxi-backup	/genesys/gcxi_shared/backup	Backup files Used by control container / jobs.	RWX	/genesys/gcxi/backup You can override this path using Values.gcxi.local.pv.backup.path	Not necessarily.	gcxi/local-pv-gcxi-backup = "true"
gcxi-log	/mnt/log	MSTR logs Used by main container. The Chart allows log volumes of legacy hostPath type. This scenario is the default.	RWX	/mnt/log/gcxi subPathExpr: $(POD_NAME) You can override this value using Values.gcxi.local.pv.log.path	Must not be shared across nodes.	gcxi/local-pv-gcxi-log = "true" If you are using hostPath volumes for logs, the Node label is not required.
gcxi-postgres	/var/lib/postgresql/data	Meta DB volume Used by Postgres	RWO	/genesys/gcxi/shared You can override this	Yes, unless you tie the PostgreSQL container to a particular	gcxi/local-pv-postgres-data = "true"

		container, if deployed.		value using Values.gcxi.local.pv.postgres.path	node	
gcxi-share	/genesys/gcxi_share	MSTR shared caches and cubes. Used by main container.	RWX	/genesys/gcxi/data subPathExpr: $(POD_NAME) You can override this value using Values.gcxi.local.pv.share.path	Yes	gcxi/local-pv-gcxi-share = "true"

Creating PVCs

Genesys recommends either of the following methods to create PVCs:

Create PVCs in advance

You can create PVCs in advance, and just configure the names in the values override file, for example:

```
pvc:
    log:
      volumeName: gcxi-log-pv
    backup:
      volumeName: gcxi-backup-pv
    share:
      volumeName: gcxi-share-pv
    postgres:
      volumeName: gcxi-postgres-pv
```

Create PVCs using Helm

You can configure the storage class name in the values override file, and let Helm create the PVCs, for example:

```
pvc:
    backup:
      capacity: 1Gi
      # if gcxi.deployment.deployLocalPV == true, this defaults to
gcxi.storageClass.local.name
      # otherwise must be set explicitly
      storageClassName: azure-files-retain #storage class name
      volumeName:
    log:
      capacity: 1Gi
      hostPath: /mnt/log/gcxi
      # if gcxi.deployment.deployLocalPV == true, this defaults to
gcxi.storageClass.local.name
      # otherwise must be set explicitly
      storageClassName: azure-files-retain #storage class name
      volumeName:
    postgres:
      capacity: 1Gi
      # if gcxi.deployment.deployLocalPV == true, this defaults to
gcxi.storageClass.local.name
```

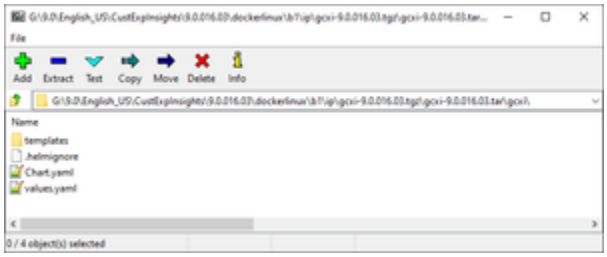
```
      # otherwise must be set explicitly
      storageClassName: azure-files
      volumeName:
    share:
      capacity: 1Gi
      # if gcxi.deployment.deployLocalPV == true, this defaults to
gcxi.storageClass.local.name
      # otherwise must be set explicitly
      storageClassName: azure-files-retain
      volumeName:
```

Install GCXI

The following procedures describe example steps to deploy GCXI with OpenShift, GKE, or AKS. The exact steps required vary depending on your environment.

1. Prepare for deployment

To prepare the environment and gather files needed for deployment, complete the steps in this section.



Contents of the Helm IP

Prerequisites

Within the Genesys Customer Experience Insights package for the Docker Linux OS, look for the Helm installation package (IP) — a small TGZ file (for example **gcxi-9.0.018.00.tgz**) that contains the Helm files. You require these files to complete this procedure.

1. On the host where the deployment will run, create a folder: **helm**.
2. Copy the Helm installation package (for example **gcxi-9.0.018.00.tgz**) into the **helm** folder, and extract the archive into a subfolder called **helm/gcxi**.
3. View the file **helm/gcxi/Chart.yaml**, and ensure that the appVersion is set to the appropriate GCXI version.
4. Open the file **helm/gcxi/values.yaml.** Follow the instructions provided in the file, and create a **values-test.yaml** file with content that is appropriate for your environment. Save the new file in the **helm** folder.

   Genesys provides the content in the following **values-test.yaml** file as an example. The example contains values that are appropriate for a simple deployment with key parameters, ingress, and the PersistentVolumes **gcxi-log-pv**, **gcxi-backup-pv**, and **gcxi-share-pv**. For information about PersistentVolumes, see PVCs required by GCXI.

```
gcxi:
     env:
         GCXI_GIM_DB:
              DSNDEF:
DSN_NAME=GCXI_GIM_DB;DB_TYPE=POSTGRESQL;DB_TYPE_EX=PostgreSQL;HOST=gim_db_host;PORT=5432;DB_NAME=gim_d
              LOGIN: gim_login
              PASSWORD: gim_password

         IWD_DB:
              DSNDEF:
DSN_NAME=IWD_DB;DB_TYPE=POSTGRESQL;DB_TYPE_EX=PostgreSQL;HOST=iwd_db_host;PORT=5432;DB_NAME=dm_gcxi;LO
            LOGIN: iwd_login
            PASSWORD: iwd_password
     deployment:
        deployLocalPV: false
        useDynamicLogPV: false
        hostIPC: false
     replicas:
        worker: 2
     pvc:
         log:
            volumeName: gcxi-log-pv
        backup:
             volumeName: gcxi-backup-pv
        share:
            volumeName: gcxi-share-pv
     ingress:
       # http path and annotations may be overriden for external and internal access
separately
       annotations:
         nginx.ingress.kubernetes.io/affinity: cookie
         nginx.ingress.kubernetes.io/affinity-mode: persistent
         nginx.ingress.kubernetes.io/proxy-body-size: "50m"
         nginx.ingress.kubernetes.io/proxy-buffer-size: "8k"
         nginx.ingress.kubernetes.io/ssl-redirect: "false"
      domain:
        external:
          annotations:
          host: gcxi.
          tls:
            enabled: true
            secretName: gcxi-ingress-ext
        internal:
          annotations:
          host: gcxi-int.
          tls:
            enabled: true
            secretName: gcxi-ingress-int
        path: /
```

2. Deploy GCXI

This procedure provides steps you can use to deploy GCXI in environments without LDAP. For environments that include LDAP or other features not supported in **values.yaml,** you can pass container environment variables such MSTR_WEB_LDAP_ON=true using the **gcxi.envvars** file, for example: `--set-file gcxi.envext=gcxi.envvars`.

1. Log in to the OpenShift, GKE, or AKS cluster from the host where you will run deployment.

2. For debug purposes, run the following command to render templates without installing:

   ```
   helm template --debug -f values-test.yaml gcxi-helm gcxi/
   ```

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

3. To deploy GCXI, execute the following command:

   ```
   helm install --debug --namespace gcxi --create-namespace -f values-test.yaml gcxi-oc
   gcxi/
   ```

   Genesys recommends that you avoid using helm install with the `--wait` option when deploying GCXI. If you use `--wait`, the Helm architecture post-install hook (which in this case is the gcxi_init job) won't be triggered properly. For more information, see the Helm documentation.

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

4. To check the installed Helm release, run the following command:

   ```
   helm list –n gcxi
   ```

5. To check GCXI OpenShift objects created by Helm, run the following command:

   ```
   kubectl get all -n gcxi
   ```

6. This step is applicable only to OpenShift deployments. Complete this step only if you have not enabled ingress in the values override file to make GCXI accessible from outside the cluster, using the standard HTTP port. For production environments, Genesys recommends that you create secure routes as discussed on the OpenShift website. For testing or development environments, perform the following steps:

   1. To make available the gcxi service, run the following command:

      ```
      oc expose service gcxi --port web --name web
      ```

   2. To verify that the new route exists in the gcxi project, run the following command:

      ```
      oc get route -n gcxi
      ```

      Route information appears, similar to the following:

      ```
      NAME      HOST/PORT                              PATH   SERVICES   PORT
      TERMINATION   WILDCARD
      web       web-gcxi.                               gcxi
      web                     None
      ```

      where is the host name generated by OpenShift.

7. Verify that you can now access GCXI at the following URL:

   ```
   http://web-gcxi./MicroStrategy/servlet/mstrWeb
   ```

Validate the deployment

Check to ensure that you can now view reports and dashboards.

Viewing Historical Reports

To view (or edit) the GCXI historical reports, open MicroStrategy Web by pointing your web browser to `http://:/MicroStrategy/servlet/mstrWeb*`, where `:` are the server name and port provided by your administrator.

Verify that an assortment of report folders are present, such as Agents, Business Results, and Callback, and that each one contains one or more reports. Before you can view a report, you must run the report to populate data. For more information, see Generate historical reports.

Maintenance and troubleshooting

Use the instructions in this section only if you encounter errors or other difficulties. Problems with the deployment are most often associated with the following three kinds of objects:

- PVs
- PVCs
- pods

Use the following steps to examine events associated with these objects:

1. To list the objects that can cause problems, run the following commands:

   ```
   kubectl get pv -o wide

   kubectl get pvc -o wide -n gcxi

   kubectl get po -o wide -n gcxi
   ```

2. Examine the output from each **get** command. If any of the objects have a non-ready state (for example, **Unbound** (PVCs only), **Pending**, or **CrashLoop**) run the following command to inspect the object more closely using **kubectl describe**:

   ```
   kubectl describe   For example: kubectl describe po gcxi-0
   ```

3. In the **describe** output, inspect the section **Events**.