Genesys Authentication Private Edition Guide

Deploy Genesys Authentication

6/4/2026

Contents

Learn how to deploy Genesys Authentication 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

Make sure to review Before you begin for the full list of prerequisites required to deploy Genesys Authentication.

Prepare your environment

To prepare your environment for the deployment, complete the steps in this section for Google Kubernetes Engine (GKE).

GKE

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

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

Create a new namespace for Genesys Authentication with a JSON file that specifies the namespace metadata. For example, **create-gauth-namespace.json**:

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

Execute the following command to create the namespace:

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

Confirm the namespace was created:

```
kubectl describe namespace gauth
```

AKS

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

```
az aks get-credentials --resource-group  --name  --admin
```

Create a new namespace for Genesys Authentication with a JSON file that specifies the namespace metadata. For example, **create-gauth-namespace.json**:

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

Execute the following command to create the namespace:

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

Confirm the namespace was created:

```
kubectl describe namespace gauth
```

Deploy

To deploy Genesys Authentication, you'll need the Helm package and your overrides file. Copy **values.yaml** and the Helm package (**gauth-.tgz**) to the installation location.

Additional Steps for deploying New Auth Service (gauth-service-authentication):

1. Use gauth Helm chart version >= 100.0.101+0258
2. Helm chart default value for new auth service is 100.0.001.**0192.** Update this to latest new auth service version >= 100.0.001.**0193** in your values.yaml

   ```
   gauth-service-authentication: 100.0.001.0193
   ```

3. For customers using gauth-infra-bg chart, Use gauth-infra-bg Helm chart version >= 100.0.101+36
4. For customers using dedicated external auth pods, to deploy this new auth service as external auth pod, Update your values.yaml

   ```
   services:
     useNewAuth: false
     service_auth:
       externalAuth:
         enabled: true    # Deploy new auth ext pods
   ```

5. **NOTE:** Update your old auth service version to your previous/existing old auth service version in your values.yaml

   ```
   gws-core-auth: 100.0.018.4405 # Do not use this version, update it with existing stable
    version
    gws-core-environment: 100.0.018.2294
    gws-ui-auth: 100.0.018.1686
    gauth-service-authentication: 100.0.001.0193
   ```

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

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

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

Now you're ready to deploy Genesys Authentication:

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

Configure external access

Follow the instructions for either GKE or AKS to make the Genesys Authentication services accessible from outside the cluster.

Provision ingresses for GKE or AKS

After deploying, make Genesys Authentication services accessible from outside the GKE or AKS cluster using the NGINX Ingress Controller.

Create a YAML file called **gauth-ingress.yaml** with the content below. **Note:** Replace **gws.** and **gauth.** with your GWS and Genesys Authentication domains, such as `gws.test.dev`.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gauth-gws-ingress
  namespace: gauth
  annotations:
    # add an annotation indicating the issuer to use.
    cert-manager.io/cluster-issuer: "selfsigned-cluster-issuer"
    # Custom annotations for NGINX Ingress Controller
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
  - host: gws. - e.g. gws.test.dev
    http:
      paths:
        - path: /ui/auth/.*
          backend:
            serviceName:  gauth-auth-ui
            servicePort: 80
        - path: /auth/.*
          backend:
            serviceName:  gauth-auth
            servicePort: 80
        - path: /environment/.*
          backend:
            serviceName:  gauth-environment
            servicePort: 80
  tls:
  - hosts:
    - gws. - e.g. gws.test.dev
    secretName: gauth-gws-ingress-cert
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gauth-gauth-ingress
  namespace: gauth
  annotations:
    # add an annotation indicating the issuer to use.
    cert-manager.io/cluster-issuer: "selfsigned-cluster-issuer"
    # Custom annotations for NGINX Ingress Controller
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
  - host: gauth. - e.g. gauth.test.dev
```

```
    http:
      paths:
        - path: /ui/auth/.*
          backend:
            serviceName:  gauth-auth-ui
            servicePort: 80
        - path: /auth/.*
          backend:
            serviceName:  gauth-auth
            servicePort: 80

        - path: /environment/.*
          backend:
            serviceName:  gauth-environment
            servicePort: 80
  tls:
  - hosts:
    - gauth. - e.g. gauth.test.dev
    secretName: gauth-gauth-ingress-cert
```

Create ingresses with the following command:

```
kubectl apply -f gauth-ingress.yaml -n gws
```

Provision GWS services to access New Auth Service

Refer Configure GWS Services

Provision New Auth Service to access GWS configuration service

Refer the Parameter `services.service_auth.env.GWS_AUTH_common_configService` in Override Helm chart values

Validate the deployment

Check the installed Helm release:

```
helm list
```

The results should show the Genesys Authentication deployment details. For example:

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

Check the **gauth** namespace status:

```
helm status gauth
```

The result should show the namespace details with a status of deployed:

```
NAME: gauth
LAST DEPLOYED: Thu May 20 11:56:32 2021
```

```
NAMESPACE: gauth
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Check the Genesys Authentication Kubernetes objects created by Helm:

```
kubectl get all -n gauth
```

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

Verify all new auth deployments are running:

```
kubectl get deployments -n gauth
```

Expected:

```
gauth-auth-                      2/2 # Already existing old auth pods
gauth-service-authentication-  2/2 # New auth pods
gauth-environment-              2/2
gauth-auth-ui-                  2/2
```

Customers using dedicated external auth pods, you will observe new auth external pods also deployed along with existing old auth external pods

```
gauth-auth-                           2/2
gauth-service-authentication-         2/2
gauth-environment-                    2/2
gauth-auth-ui-                        2/2
gauth-auth-ext-                       2/2  # Already existing old auth external pods
gauth-service-auth-ext-               2/2  # New auth external pods
```

Finally, verify that you can now access Genesys Authentication at the following URL: https:///ui/auth/sign-in.html

Genesys Service Authentication Cut Over Guide

Overview

Starting with `gauth` Helm chart version `100.0.101+0258` or later, the new authentication service `gws-service-authentication` is deployed alongside the existing authentication service `gauth-auth`. By default, traffic continues to route to the old authentication service.

The cutover switches authentication traffic from the old authentication service `gauth-auth` to the new authentication service `gws-service-authentication`.

Before cutover:

- Traffic routes to the old authentication service.
- Client data is copied from old database tables to new database tables.

- Existing authentication tokens remain in the old token format.

After cutover:

- `/auth` traffic routes to the new authentication service.
- New tokens are created in the new authentication token format.
- Existing tokens created by the old authentication service cannot be read by the new authentication service.
- Dependent services must be restarted, and users must re-login to obtain new tokens.

This guide further explains how to migrate client data, validate the new service, switch authentication traffic to the new service, validate the cutover, restart dependent services, and roll back to the old service if needed

Migration Process:

Auth Service has 2 persistent pieces of information.

1. Client Details
2. Authentication Token

1.Client Details:

The new authentication service (`gauth-service-authentication`) uses the same Postgres database and Redis cluster as the old auth service (`gauth-auth`), but with different table formats.

Data	Old Auth table name	New Auth Table name
Clients	oauth_client_details	oauth2_client_details
Contact Centers	client_contact_centers	client_contact_centers2

Automatic migration:

When the new authentication service starts, Flyway runs the migration script:

```
V1.7.1__CopyOauth2ClientDetailsFromOldTable.sql
```

This migration copies client data from the old tables to the new tables.

The migration runs once on the first startup of the new authentication service. On subsequent restarts, Flyway skips the migration because it has already been applied.

2. Authentication Token:

Authentication tokens from the old authentication service are not migrated. The old and new authentication services use incompatible token storage formats. Tokens created by the old authentication service cannot be read by the new authentication service. In order to regenerate the tokens, refer to step Restart Applicable Services of CutOver steps.

And below is the difference in token storage formats of both the authentication services,

Area	**Old Auth**	**New Auth**
Serialization	JDK binary	Jackson JSON
Key prefix	configurable (empty)	GWS:Auth:V3:
Token keys	plain token values	SHA-256 hashed
Data model	OAuth2AccessToken + OAuth2Authentication	OAuth2Authorization (unified)

Pre Validation

Complete the following checks before switching traffic to the new authentication service.

Step 1: Verify New Auth service health

Check that the new authentication pods are running:

```
kubectl get pods -n gauth -l gauth=service-auth
```

Check health from the pod:

```
kubectl exec -n gauth  -- curl -s http://localhost:8081/health
```

Expected response:

```
{"status":"UP"}
```

Verify that the service started successfully:

```
kubectl logs -n gauth  | grep -i "Started"
```

Expected log message:

```
Started AuthorizationServerApplication in X seconds
```

Step 2: Verify Client Data Migration

Check the new authentication service logs for Flyway migration status:

```
kubectl logs -n gauth  | grep -i "migrat"
```

Expected output on first startup:

```
Migrating schema "public" to version "1.7.1 - CopyOauth2ClientDetailsFromOldTable"
Successfully applied 1 migration(s) to schema "public"
```

Expected output if the migration was already applied:

```
Schema "public" is up to date. No migration necessary.
```

You can also verify migrated client data through the operations API on the new authentication pod.

Start port-forwarding:

```
kubectl port-forward -n gauth  8080:8080
```

Then call the GET Clients API:

```
curl -u : \
  http://localhost:8080/auth/v3/ops/clients
```

Expected result:

- Clients are returned from the new authentication service.
- If no clients are returned, do not proceed with cutover until migration is verified.

Step 3: Run a Pre-cutover functional test

Before switching traffic, verify the new authentication service directly through pod port-forwarding.

Start port-forwarding:

```
kubectl port-forward -n gauth  8080:8080
```

Call the token endpoint:

```
curl -X POST http://localhost:8080/auth/v3/oauth/token \
  -d "grant_type=client_credentials&client_id=&client_secret="
```

Expected response:

```
{
  "access_token": "...",
  "token_type": "...",
  "expires_in": ...
}
```

Cut Over

Step 1: Configure Helm Values

Refer Configure Genesys Authentication for below helm values configurations

Customers Using the `gauth` Chart Only

Set the following value:

```
services:
  useNewAuth: true
```

This value is required to switch traffic to the new authentication service.

If dedicated external authentication pods are required, also set:

```
services:
  service_auth:
    externalAuth:
      enabled: true
```

Customers Using Both `gauth` and `gauth-infra-bg` Charts

For the `gauth` chart, set:

```
services:
  useNewAuth: true
```

For the `gauth-infra-bg` chart, set:

```
active:
  useNewAuth: true
```

If dedicated external authentication is required, also set:

```
active:
  externalAuth: true
```

Important: Always set `services.useNewAuth` and `active.useNewAuth` to the same value. Mismatched values can result in two ingresses pointing to different authentication services, causing unpredictable routing behavior.

Step 2: Confirm Database Migration Is Enabled

Automatic migration of PostgreSQL data from old auth tables to new auth tables is enabled by default.

Example configuration:

```
services:
  useNewAuth: false

  secrets:
    useSecretProviderClass: false

  replicas: 3
  location: /

  db:
    init: true # enables automatic migration
    poolSize: 3
```

Step 3: Deploy the `gauth` Helm Chart

Deploy the updated `gauth` chart:

```
helm upgrade --install gauth ./gauth-.tgz \
  -f .yaml \
  -n gauth \
  --wait \
  --timeout 600s
```

Step 4: Verify Ingress Routes to New Auth

Verify that `/auth/` routes to the new authentication service:

```
kubectl get ingress -n gauth -o yaml | grep -A5 "path: /auth/"
```

Expected backend:

```
path: /auth/
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-service-authentication-
    port:
      number: 8080
```

If the backend still shows `gauth-auth-`, the cutover has not taken effect.

Step 5: Verify External Auth Ingress, If Used

For customers using dedicated external authentication, verify that `/auth/v3/oauth/token` routes to the new external authentication service:

```
kubectl get ingress -n gauth -o yaml | grep -A5 "path: /auth/v3/oauth/token"
```

Expected backend:

```
path: /auth/v3/oauth/token
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-service-auth-ext-
    port:
      number: 8080
```

If the backend still shows `gauth-auth-ext-`, the external authentication cutover has not taken effect.

Additional Steps for Customers Using `gauth-infra-bg`

Update the `gauth-infra-bg` chart values:

```
active:
  useNewAuth: true
  externalAuth: true
```

Set `externalAuth: true` only if dedicated external authentication is used.

Deploy the `gauth-infra-bg` chart:

```
helm upgrade --install gauth-infra ./gauth-infra-bg-.tgz \
  -f .yaml \
  -n gauth \
  --wait
```

Verify that `/auth/` routes to the new authentication service:

```
kubectl get ingress -n gauth -l service=gauth-infra-bg -o yaml | grep -A5 "path: /auth/"
```

Expected backend:

```
path: /auth/
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-service-authentication--active
    port:
      number: 80
```

If the backend still shows `gauth-auth--active`, the cutover has not taken effect. For dedicated external authentication, verify:

```
kubectl get ingress -n gauth -l service=gauth-infra-bg -o yaml | grep -A5 "path: /auth/v3/
oauth/token"
```

Expected backend:

```
path: /auth/v3/oauth/token
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-service-auth-ext--active
    port:
      number: 80
```

If the backend still shows `gauth-auth-ext--active`, the external authentication cutover has not taken effect.

Validate Cutover

Step 1: Test the External Ingress

Call the token endpoint through the external ingress URL:

```
curl -X POST https:///auth/v3/oauth/token \
  -d "grant_type=client_credentials&client_id=&client_secret="
```

Expected response:

```
{
  "access_token": "...",
  "token_type": "...",
  "expires_in": ...
}
```

Step 2: Test the Internal Ingress

Call the token endpoint through the internal ingress URL:

```
curl -X POST https:///auth/v3/oauth/token \
  -d "grant_type=client_credentials&client_id=&client_secret="
```

Expected response:

```
{
  "access_token": "...",
  "token_type": "...",
  "expires_in": ...
}
```

Ingress host values come from the Helm values files:

Chart	External Ingress Value	Internal Ingress Value
`gauth`	`ingress.frontend`	`internal_ingress.frontend`
`gauth-infra-bg`	`active.ingress.frontend`	`active.internal_ingress.frontend`

Restart Applicable Services

Restart dependent services after cutover.

This restart is required because existing cached tokens were created by the old authentication service. These old tokens cannot be validated by the new authentication service.

Example restart commands:

```
kubectl rollout restart deployment gws-service-configuration -n gws
kubectl rollout restart deployment gws-app-provisioning -n gws
```

Example applicable services include:

i. gws-service-configuration

ii. gws-app-provisoning

ii. GIR(Genesys Interaction Recording) - Speechminer Web service

iv. Nexus

v. CIWD

vi. UCSX (only Azure)

vii. UDM (CDDS-X)

Impact on Existing Logins

- Existing agent and user sessions are not terminated automatically during cutover.
- However, ongoing or new read/write operations may fail after cutover because old tokens use JDK binary serialization and cannot be read by the new authentication service, which uses Jackson JSON serialization.
- Users and agents must log in again to resume normal operations:

Examples:

User Type	Required Action
WWE users	Log out and log back in
Agent Setup users	Log out and log back in

Re-login creates new tokens in the new authentication service format. After re-login, operations should work normally.

Rollback (If needed)

After rollback:

- Traffic routes back to the old authentication service.
- Old auth pods are already running, so old auth pod restart is not required.
- Tokens created by the new authentication service become invalid.
- Dependent services must be restarted again so they obtain tokens from the old authentication service.
- Users and agents must log in again.

Step 1: Switch the `gauth` Chart Back to Old Auth

Update the `gauth` chart values:

```
services:
  useNewAuth: false
```

Deploy the `gauth` chart:

```
helm upgrade --install gauth ./gauth-.tgz \
  -f .yaml \
  -n gauth \
  --wait \
  --timeout 600s
```

Verify that `/auth/` routes back to the old authentication service:

```
kubectl get ingress -n gauth -o yaml | grep -A5 "path: /auth/"
```

Expected backend:

```
path: /auth/
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-auth-
    port:
      number: 8080
```

For dedicated external authentication, verify:

```
kubectl get ingress -n gauth -o yaml | grep -A5 "path: /auth/v3/oauth/token"
```

Expected backend:

```
path: /auth/v3/oauth/token
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-auth-ext-
    port:
      number: 8080
```

Step 2: Switch the `gauth-infra-bg` Chart Back to Old Auth, **If Used**

Update the `gauth-infra-bg` chart values:

```
active:
  useNewAuth: false
```

If external authentication was enabled only for the new authentication service and must also be switched back, update the external authentication value as required by your deployment.

Deploy the `gauth-infra-bg` chart:

```
helm upgrade --install gauth-infra ./gauth-infra-bg-.tgz \
  -f .yaml \
  -n gauth \
  --wait
```

Verify that `/auth/` routes back to the old authentication service:

```
kubectl get ingress -n gauth -l service=gauth-infra-bg -o yaml | grep -A5 "path: /auth/"
```

Expected backend:

```
path: /auth/
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-auth--active
    port:
      number: 80
```

For dedicated external authentication, verify:

```
kubectl get ingress -n gauth -l service=gauth-infra-bg -o yaml | grep -A5 "path: /auth/v3/
oauth/token"
```

Expected backend:

```
path: /auth/v3/oauth/token
pathType: ImplementationSpecific
backend:
  service:
    name: gauth-auth-ext--active
    port:
      number: 80
```

Step 3: Restart Applicable Services After Rollback

Restart dependent services again after rollback.

This is required because services may hold tokens created by the new authentication service, and those tokens cannot be validated by the old authentication service.

Example commands:

```
kubectl rollout restart deployment gws-service-configuration -n gws
kubectl rollout restart deployment gws-app-provisioning -n gws
```

Restart any other applicable services listed in the Restart Applicable Services section.

Step 4: Validate Rollback

Call the token endpoint through the external ingress URL:

```
curl -X POST https:///auth/v3/oauth/token \
  -d "grant_type=client_credentials&client_id=&client_secret="
```

Expected response:

```
{
  "access_token": "...",
  "token_type": "...",
  "expires_in": ...
}
```

Call the token endpoint through the internal ingress URL:

```
curl -X POST https:///auth/v3/oauth/token \
  -d "grant_type=client_credentials&client_id=&client_secret="
```

Expected response:

```
{
  "access_token": "...",
  "token_type": "...",
  "expires_in": ...
}
```

Troubleshooting

Ingress Still Points to Old Auth After Cutover

If ingress still points to `gauth-auth-` or `gauth-auth-ext-`, the cutover did not take effect.

Check:

- `services.useNewAuth` is set to `true`
- `active.useNewAuth` is set to `true`, if using `gauth-infra-bg`
- Helm upgrade completed successfully
- The correct values file was used
- Ingress was updated after deployment

Token Requests Fail After Cutover

Check:

- Client migration completed successfully
- Client ID and client secret are valid
- Token request is reaching the new authentication service
- Dependent services were restarted
- Users have logged out and logged back in

Operations Fail for Logged-In Users After Cutover

Existing user sessions are not automatically terminated, but operations may fail because old tokens are incompatible with the new authentication service.

Resolution:

- Ask affected users to log out and log back in.
- Restart dependent services that cache service tokens.

Token Requests Fail After Rollback

After rollback, tokens created by the new authentication service are invalid for the old authentication service.

Resolution:

- Restart dependent services again.
- Ask users and agents to log out and log back in.
- Verify ingress routes back to the old authentication service.