Designer Private Edition Guide

12/22/2025

Table of Contents

Overview

- About Designer 6
- Architecture 9
- High availability and disaster recovery 14

Configure and deploy

- Before you begin 16
- Configure Designer 23
- Platform / Configuration Server and GWS settings for Designer 68
- Deploy Designer 72
- Enable optional features 84

Upgrade, roll back, or uninstall

- Upgrade, roll back, or uninstall Designer 92

Observability

- Observability in Designer 102
  - DES metrics and alerts 108
  - DAS metrics and alerts 112
- Logging 117

Kubernetes platform specific information

- Designer on GKE 119
- Designer on AKS 122

Contents

- 1 Overview
- 2 Configure and deploy
- 3 Upgrade, roll back, or uninstall
- 4 Operations
- 5 Kubernetes platform specific information

This document guides you through the process of deploying and configuring Designer and Designer Application Server (DAS) as a service in a Kubernetes (K8s) cluster.

Related documentation:

-
-

RSS:

- For private edition

Designer is a service available with the Genesys Multicloud CX private edition offering.

This document is intended for use primarily by system engineers and other members of an implementation team who will be involved in configuring and installing Designer and DAS, and system administrators who will maintain Designer and DAS installations.

To successfully deploy and implement applications in Designer and DAS, you must have a basic understanding of and familiarity with:

- Network design and operation
- Network configurations in your organization
- Kubernetes
- Genesys Framework architecture and functions

Overview

Learn more about Designer, its architecture, and how to support high availability and disaster recovery.

- About Designer
- Architecture
- High availability and disaster recovery

Configure and deploy

Find out how to configure and deploy Designer.

- Before you begin
- Configure Designer
- Platform / Configuration Server and GWS settings for Designer
- Deploy Designer
- Enable optional features

Upgrade, roll back, or uninstall

Find out how to upgrade, roll back, or uninstall Designer.

- Upgrade, roll back, or uninstall Designer

Operations

Learn how to monitor Designer with metrics and logging.

- Observability in Designer
- Designer metrics and alerts
- DAS metrics and alerts
- Logging

Kubernetes platform specific information

Learn more about settings specific to the Kubernetes platform or the container orchestration platform you are deploying Designer on.

- Designer on GKE
- Designer on AKS

About Designer

Contents

- 1 Designer
- 2 Designer Application Server (DAS)
- 3 Supported Kubernetes platforms

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

Related documentation:

-
-
-

RSS:

- For private edition

Designer

The Designer service provides a web UI to build and manage VXML and SCXML based self-service and assisted service applications for a number of media types. It stores data on the local file system and is synchronized across instances by using services like Network File System (NFS). Genesys customers can build applications using a simple drag and drop method, and assign contact points (Route Points and other media endpoints) to applications directly from the Designer UI. Insights into runtime behavior of applications and troubleshooting aid is provided by Designer Analytics, which includes a rich set of dashboards based on session detail records (SDR) from data stored in Elasticsearch.

Designer offers the following features:

- Applications for working with phone, chat, email, SMS (text messages), Facebook, Twitter, and open media types.
- Bots, ASR, TTS capabilities for self-service.
- Assisted service or routing.
- Callback.
- Business Controls.
- Audio, message management.
- Grammars management.
- Contact points management - route points, chat end points, email pop-client/mailboxes.
- Analytics dashboards through embedded Kibana.

Designer is an Express/Node.js application. The UI is designed using Angular powered Bootstrap.

Application data (SCXML and VXML) is stored as a file system. Designer Analytics and Audit data is stored in Elasticsearch.

Designer Application Server (DAS)

Designer Application Server (DAS) hosts and serves the Designer generated application files (SCXML and VXML), audio, and grammars. It also provides:

- Runtime evaluation of Business Controls (business hours, special days, emergency flags and data tables).
- Callback interface to GES.

DAS uses built-in NGINX to front requests. It consists of 3 modules: NGINX, PHP, and Node.js.

- Requests for static workspace content (SCXML, VXML, JS, audio, grammar, etc) are handled by the NGINX module.
- Requests for PHP content are processed by the FastCGI PHP module.
- SDR (Analytics) processing requests are handled by the DAS Node.js module.

> **Important**
>
> Files generated by Designer can be served only by DAS. Designer will work only with DAS.

Supported Kubernetes platforms

The Designer and DAS services are supported on the following Kubernetes platforms:

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

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

Architecture

Contents

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

Learn about Designer architecture

Related documentation:

-
-
-

RSS:

- For private edition

Introduction

The architecture diagram in this topic illustrates a sample deployment of Designer and DAS.

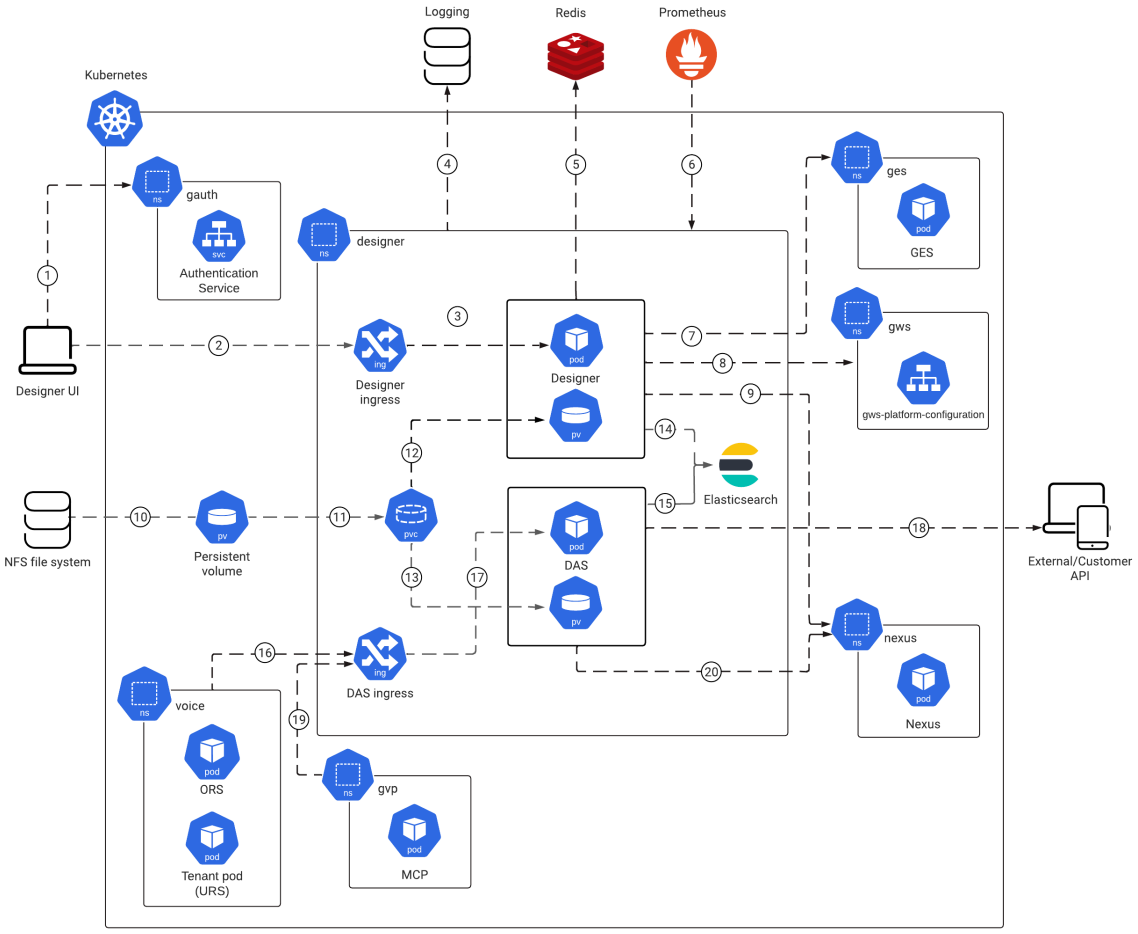
For more information on the Genesys Multicloud CX private edition architecture, refer to the Architecture topic in the *Setting up Genesys Multicloud CX private edition* document.

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

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

Architecture diagram — Connections

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



Connections table

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

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
1	Customer browser	Genesys Authentication	HTTPS	8095	Intra-cluster	Designer queries the Genesys Authentication Service to validate the user's identity.
2		Designer ingress	HTTP	443	Ingress	Web browser used to access the Designer UI.
3	Designer ingress	Designer	HTTP	8888	Egress	Incoming web traffic from the UI.
4	Designer	Logging	HTTP		Egress	Centralized logging.
5	Designer	Redis	HTTP	6380	Egress	Resource index caching and multi-user collaboration locks on Designer resources.
6	Prometheus	Designer	HTTP	8888	Ingress	Metrics for monitoring and alerting with Prometheus.
7	Designer	Genesys Engagement Service	HTTP	80		Publish callback services.
8	Designer	Genesys Web Services and Applications	HTTP	80		Authentication of Designer and configuration data access.
9	Designer replica set	Nexus	HTTP/HTTPS	80	Ingress	Fetch Designer Bot registry information.
10	Shared file system (NFS)	Persistent volume				NFS for workspace storage.
11	Persistent volume	Designer Persistent Volume				Data for workspace storage.

Connection	Source	Destination	Protocol	Port	Classification	Data that travels on this connection
		Claim (PVC)				
12	Designer Persistent Volume Claim (PVC)	Designer replica set persistent volume				Data for workspace storage.
13	Designer Persistent Volume Claim (PVC)	DAS replica set persistent volume				Data for workspace storage.
14	Designer	Elasticsearch	HTTP	9205	Egress	Query Designer Analytics data (Session Detail Records).
15	Designer Application Server	Elasticsearch	HTTP	9205	Egress	Store Designer Analytics data (Session Detail Records).
16	Voice Microservices	DAS ingress	HTTP	80	Ingress	Fetch Designer application pages (VXML, SCXML), JSON files, and so on.
17	DAS ingress	Designer Application Server	HTTP	8080	Ingress	HTTP traffic from DAS ingress.
18	External/ Customer	Designer Application Server	HTTPS	443	Egress	External customer API requests.
19	Genesys Voice Platform	DAS ingress	HTTP	80	Ingress	Fetch Designer audio resources.
20	Designer Application Server	Nexus	HTTP/HTTPS	80	Ingress	Fetch GES APIs for callback processing.

High availability and disaster recovery

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

Related documentation:

-
-
-

RSS:

- For private edition

Name	High Availability	Disaster Recovery	Where can you host this service?
Designer	N = N (N+1) *Or* N = 2 (active-active)	Pilot light	Primary unit only
Designer Application Server	N = N (N+1) *Or* N = 2 (active-active)	Active-spare	Primary or secondary unit

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

Designer and DAS must be deployed as highly available in order to avoid single points of failure. A minimum of 2 replicas of each service must be deployed to achieve HA.

The Designer and DAS service pods can be automatically scaled up or down based on metrics such as CPU and memory utilization. The Deployment configuration settings section provides more information on configuring HA and auto-scaling.

> **Important**
> The pilot-light DR or multi-region pattern for the Designer service is supported only for

> the primary region.

Refer to the Genesys Docker Deployment Guide for more information on general HA recommendation for Kubernetes.

Before you begin

Contents

- 1 Limitations and assumptions
- 2 Download the Helm charts
- 3 Third-party prerequisites
- 4 Storage requirements
- 5 Network requirements
- 6 Browser requirements
  - 6.1 Minimum display resolution
  - 6.2 Third-party cookies
- 7 Genesys dependencies
- 8 GDPR support

Find out what to do before deploying Designer.

Related documentation:

-
-
-

RSS:

- For private edition

Limitations and assumptions

Designer currently supports multi-tenancy provided by the tenant Configuration Server. That is, each tenant should have a dedicated Configuration Server, and Designer can be shared across the multiple tenants.

Before you begin:

1. Install Kubernetes. Refer to the Kubernetes documentation site for installation instructions. You can also refer to the Genesys Docker Deployment Guide for information on Kubernetes and High Availability.
2. Install Helm according to the instructions outlined in the Helm documentation site.

After you complete the above mandatory procedures, return to this document to complete deployment of Designer and DAS as a service in a K8s cluster.

> **Important**
> Designer applications cannot be used to handle default routed calls or voice interactions. IRD applications should be used for such scenarios until Designer adds support for handling default routed calls or voice interactions.

Download the Helm charts

Download the Designer related Docker containers and Helm charts from the JFrog repository.

See Helm charts and containers for Designer for the Helm chart and container versions you must download for your release.

For more information on JFrog, refer to the Downloading your Genesys Multicloud CX containers topic in the *Setting up Genesys Multicloud CX private edition* document.

Third-party prerequisites

The following section lists the third-party prerequisites for Designer.

- Kubernetes 1.19.x - 1.21.x
- Helm 3.0
- Docker
  - To store Designer and DAS docker images to the local docker registry.
- Ingress Controller
  - If Designer and DAS are accessed from outside of a K8s cluster, it is recommended to deploy/configure an ingress controller (for example, NGINX), if not already available. Also, the Blue-Green deployment strategy works based on the ingress rules.
  - The Designer UI requires *Session Stickiness*. Configure session stickiness in the *annotations* parameter in the **values.yaml** file during Designer installation.

For information about setting up your Genesys Multicloud CX private edition platform, including Kubernetes, Helm, and other prerequisites, see Software requirements.

Third-party services

Name	Version	Purpose	Notes
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.	
Load balancer		VPC ingress. For NGINX Ingress Controller, a single regional Google external network LB with a static IP and wildcard DNS entry will pass HTTPS traffic to NGINX Ingress Controller which will terminate SSL traffic and will be setup as part of the platform setup.	
Elasticsearch	7.x	Used for text searching and indexing. Deployed per service that needs Elasticsearch during	Elasticsearch 7.8.0 is used for Designer Analytics and audit trail.

Name	Version	Purpose	Notes
		runtime.	
Redis	6.x	Used for caching. Only distributions of Redis that support Redis cluster mode are supported, however, some services may not support cluster mode.	Redis is used for resource index caching and multi-user collaboration locks on Designer resources.

Storage requirements

The following storage requirements are mandatory prerequisites:

- Persistent Volumes (PVs)
  - Create persistent volumes for workspace storage (5 GB minimum) and logs (5 GB minimum)
  - Set the access mode for these volumes to *ReadWriteMany*.
  - The Designer manifest package includes a sample YAML file to create Persistent Volumes required for Designer and DAS.
  - Persistent volumes must be shared across multiple K8s nodes. Genesys recommends using NFS to create Persistent Volumes.
- Shared file System - NFS
  - For production, deploy the NFS server as highly available (HA) to avoid single points of failure. It is also recommended that the NFS storage be deployed as a Disaster Recovery (DR) topology to achieve continuous availability if one region fails.
  - By Default, Designer and DAS containers run as a Genesys user (`uid:gid 500:500`). For this reason, the shared volume must have permissions that will allow write access to `uid:gid 500:500`. The optimal method is to change the NFS server host path to the Genesys user: `chown -R genesys:genesys`.
  - The Designer package includes a sample YAML file to create an NFS server. Use this only for a demo/lab setup purpose.
  - Azure Files Storage - If you opt for Cloud storage, then Azure Files Storage is an option to consider and has the following requirements:
    A Zone-Redundant Storage for RWX volumes replicated data in zone redundant (check this), shared across multiple pods.
    - Provisioned capacity : 1 TiB
    - Baseline IO/s : 1424
    - Burst IO/s : 4000
    - Egress Rate : 121.4 MiBytes/s
    - Ingress Rate : 81.0 MiBytes/s

Network requirements

- If Designer and DAS are accessed from outside of a K8s cluster, it is recommended to deploy/configure an ingress controller (for example, NGINX), if not already available. Also, the Blue-Green deployment strategy works based on the ingress rules.
- The Designer UI requires Session Stickiness. Configure session stickiness in the annotations parameter in the values.yaml file during Designer installation.

Browser requirements

Unless otherwise noted, Designer supports the latest versions of the following browsers:

- Mozilla Firefox
- Google Chrome (see Important, below)
- Microsoft Edge
- Apple Safari

Internet Explorer (all versions) is not supported.

> **Important**
>
> For Google Chrome, Designer supports the *n*-1 version of the browser, i.e. the version prior to the latest release.

Minimum display resolution

The **minimum** display resolution supported by Designer is **1920 x 1080**.

Third-party cookies

Some features in Designer require the use of third-party cookies. Browsers must allow third-party cookies to be stored for Designer to work properly.

Genesys dependencies

The following Genesys dependencies are mandatory prerequisites:

- Genesys Web Services (GWS) 9.x
  - Configure GWS to work with a compatible version of Configuration Server.

- Other Genesys Components
  - Authentication Service
  - Voice Microservices

For the order in which the Genesys services must be deployed, refer to the Order of services deployment topic in the *Setting up Genesys Multicloud CX private edition* document.

GDPR support

Designer supports the European Union's General Data Protection Regulation (GDPR) requirements and provides customers the ability to export or delete sensitive data using ElasticSearch APIs and other third-party tools.

For the purposes of GDPR compliance, Genesys is a data processor on behalf of customers who use Designer. Customers are the data controllers of the personal data that they collect from their end customers, that is, the data subjects. Designer Analytics can potentially store data collected from end users in ElasticSearch. This data can be queried by certain fields that are relevant to GDPR. Once identified, the data can be exported or deleted using ElasticSearch APIs and other third-party tools that customers find suitable for their needs.

In particular, the following SDR fields may contain PII or sensitive data that customers can choose to delete or export as required:

- **ANI** - This SDR field contains the customer's phone number used to make voice calls handled by Designer applications.
- **variables.Contact** - This SDR field is an object and can have multiple properties, such as, name, email address, and other contact details. For example,

```
{
    "ContactId":"AAABBA1000000I9y",
    "EmailAddress":"john.doe@home.com",
    "FromPersonal":"John Doe ",
    "FromAddress":"john.doe@home.com",
    "FirstName":"John",
    "LastName":"Doe"
}
```

- Application variables defined in the main application flow are also stored in the SDR under the `variables` object. These variables depend on application logic and may capture sensitive information intentionally or unintentionally. It is recommended to mark such variables secure (see Securing Variables in *Designer Help* for more details). But if they are captured in analytics, they can also be used to identify candidate SDRs for deletion or retrieval. The same applies to `userdata` key value pairs attached to interaction data which is captured in the `calldata` object in the SDR.

> **Important**
>
> It is the customer's responsibility to remove any PII or sensitive data within 21 days or less, if required by General Data Protection Regulation (GDPR) standards.

For general information about Genesys support for GDPR compliance, see General Data Protection Regulation.

Configure Designer

Contents

- 1 Deployment configuration settings (Helm values)
- 2 Designer deployment settings
  - 2.1 Designer ConfigMap settings
- 3 DAS deployment settings
  - 3.1 DAS ConfigMap settings
- 4 Post deployment Designer configuration settings
  - 4.1 Flow settings
  - 4.2 Tenant settings
  - 4.3 DesignerEnv transaction list
  - 4.4 Post deployment configuration settings reference table
  - 4.5 Features
- 5 Adding a UI plugin to Designer

Learn how to configure Designer.

Related documentation:

-
-
-

RSS:

- For private edition

Deployment configuration settings (Helm values)

The following sections provide information on the various settings that have to be configured in Designer and DAS. The configuration settings listed below will be used during the deployment of Designer and DAS. That is, these settings will be used during initial deployment/upgrade. These settings can be configured in the **values.yaml** Helm file.

For more information about how to override Helm chart values, see Overriding Helm chart values in the *Setting up Genesys Multicloud CX Private Edition* guide.

> **Important**
> Depending on the Kubernetes platform or the container orchestration platform that you are deploying Designer on, you might have to carry out some additional steps specific to that platform. For more information, navigate to the required topic in the **Kubernetes platform specific information** section on the About page.

Designer deployment settings

The following table provides information on the Designer deployment settings. These settings are configured in the **designer-values.yaml** file.

Parameter	Description	Mandatory?	Default Value
`designer.deployment.replicaCount`	Number of service instances to be created.	Mandatory	`2`
`designer.deployment.maxReplicaCount`	The maximum number of replicas to be created. It is	Optional	`10`

	recommended to configure this setting if auto-scaling is used.		
`designer.deployment.strategy`	The deployment strategy to follow. This determines which type of resources are deployed. Valid values are: `rollingupdate, blue-green, blue-green-volume, blue-green-ingress, grafana`. • **rollingupdate** - default Kubernetes update strategy where resources will be updated using the rolling upgrade strategy. • **blue-green** - for deploying and upgrading the Designer service using the blue-green strategy. • **blue-green-volume** - for the blue/green upgrade, this is to create a Persistent Volume Claim (PVC) for the very first time. • **blue-green-ingress** - for the blue/green upgrade, this is to create an ingress for the first time and update the ingress during a service cutover. • **grafana** - for deploying the Grafana dashboard.	Mandatory	`rollingupdate`
`designer.deployment.color`	This is to deploy/ upgrade the Designer service in a blue-green upgrade strategy. Valid values are: `blue, green`.	Optional	
`designer.deployment.type`	This is to specify the	Optional	`Deployment`

	type of deployment. Valid value: `Deployment`.		
`designer.image.registry`	The registry that the organization uses for storing images.	Mandatory	
`designer.image.repository`	Docker repository that contains the images for Designer.	Mandatory	
`designer.image.tag`	Designer image version.	Mandatory	9.0.110.07.7
`designer.image.PullPolicy`	Designer image pull policy (imagePullPolicy). Valid values: `Always, IfNotPresent, Never`. • **Always** - always pull the image. • **IfNotPresent** - pull the image only if it does not already exist on the node. • **Never** - never pull the image.	Mandatory	`IfNotPresent`
`designer.image.imagePullSecrets`	Secret name containing credentials for authenticating access to the Docker repository.	Mandatory	
`designer.volumes.workspacePv.create`	`true` if a persistent volume for the Designer workspace must be created. This is used in case of static volume provisioning, where, the PV is created and then the PVC is bound to the specified PV. Currently, support to create PV only for Azure files (SMB) and NFS is present in the helm chart.		`false`
`designer.volumes.workspacePv.type`	Supports two types: `nfs` - Creates an NFS PV provided you have an NFS server/file share set up already. `azurefiles-smb` - Creates a PV for pre-existing SMB type Azure fileshares.		
`designer.volumes.workspacePv.name`	Name of the PV to be created. For example,		

	`designer-workspace-pv`.		
`designer.volumes.workspacePv.storage`	Size of the PV to be created. For example, `5Gi`.		
`designer.volumes.workspacePv.storageClass`	The storage class associated with the PV. For static volume provisioning to occur as expected, it is highly recommended to provide `""` (intentional empty double quotes) or any distinct storage class name that does not exist already.		
`designer.volumes.workspacePv.mountOptions`	Mount options to be given to the PV. Note: Mount options differ according to the underlying storage type used, that is, NFS or SMB. Using the same set of `mountOptions` with different storage types leads to volume mount errors.		
`designer.volumes.workspacePv.server`	The IP address or FQDN of the NFS server. Note: This field is only applicable for `nfs` type PVs.		
`designer.volumes.workspacePv.path`	The exported path from the NFS server. Note: This field is only applicable for `nfs` type PVs.		
`designer.volumes.workspacePv.shareName`	The azure fileshare name for which the PV must be created. Note: This field is only applicable for `azurefiles-smb` type PVs.		
`designer.volumes.workspacePv.createSecret`	`true` if secret with data to authenticate the Azure storage account must be created. Can be `false` if the secret is manually created. Note: This field is only applicable for `azurefiles-smb` type PVs.		
`designer.volumes.workspacePv.secretName`	The name to be given to		

	the secret created with the `designer.volumes.workspacePv.createSecret` field. For example, `designer-storage-secret`). Note: This field is only applicable for `azurefiles-smb` type PVs.		
`designer.volumes.workspacePv.`	Base64 encoded name of the storage account. This goes in the secret created with `designer.volumes.workspacePv.createSecret`. Note: This field is only applicable for `azurefiles-smb` type PVs.		
`designer.volumes.workspacePv.`	Base64 encoded access key of the storage account. This goes in the secret created with `designer.volumes.workspacePv.createSecret` Note: This field is only applicable for `azurefiles-smb` type PVs.		
`designer.volumes.workspacePvc.`	If a persistent volume is to be created, this value has to be `true`.	Mandatory	`true`
`designer.volumes.workspacePvc.`	The type of the volume provisioning to use: `static` - This type is used when a PV has been created either by using the helm values in `designer.volumes.workspacePv` or bmanually and the workspace PVC must be bound to it. `dynamic` - This type is used when a configured storage class will dynamically allocate a PV to the workspace PVC.	Mandatory	`dynamic`
`designer.volumes.workspacePvc.mountPath`	The path where the workspace volume is to be mounted inside the Designer container.	Mandatory	`/designer/workspace` Note: This is not a customizable value. The value MUST be `/designer/workspace` for the proper functioning of Designer.
`designer.volumes.workspacePvc.`	Persistent volume claim name for the workspace.	Mandatory	`designer-managed-disk`

`designer.volumes.workspacePvc.claimSize`	Size of the persistent volume claim for the workspace. The persistent volume must be equal to or greater than this size.	Mandatory	
`designer.volumes.workspacePvc.storageClass`	storageClassName provided in the persistent volume that is created for the Designer workspace (example, **nfs**).	Mandatory	
`designer.volumes.workspacePvc.volumeName`	The PV's name to which the PVC must be bound (applicable only when `designer.volumes.workspacePvc.type` is `static`).		
`designer.volumes.logsPvc.create`	If a PVC volume is to be created, this value has to be `true`, else `false`.	Mandatory	`true`
`designer.volumes.logsPvc.type`	The type of volume provisioning to use: `static` - This type is used when a PV has been created and PVC logs must be bound to it. `dynamic` - This type is used when a configured storage class will dynamically allocate a PV to the PVC logs. Note: The helm charts only have support for creating static PVs for the PVC workspace. For PVC logs, it is recommended to make use of dynamic provisioning and let the storage class do the PV allocation.		
`designer.volumes.logsPvc.mountPath`	The path where the Designer logs volume is to be mounted inside the Designer container.	Mandatory	`/designer/logs` Note: This is not a customizable value. The value MUST be `/designer/logs` for the proper functioning of Designer.
`designer.volumes.logsPvc.claim`	Persistent volume claim name for logs.	Mandatory	`designer-logs`
`designer.volumes.logsPvc.claimSize`	Size of the persistent volume claim for the Designer logs. The persistent volume must be equal to or greater than this size.	Mandatory	
`designer.volumes.logsPvc.storageClassName`	storageClassName	Mandatory	

	provided in the persistent volume that is created for the Designer logs (example, **nfs**). Note: In case of static volume provisioning, this field must match with the storage class of the PV. If the PV does not have a storage class, then it is mandatory to provide `""` for this field in the helm values. Otherwise, static volume provisioning will not occur as expected.		
`designer.volumes.logsPvc.appName`	The PV's name to which the PVC must be bound (applicable only when `designer.volumes.logsPvc.type` is static).		

`designer.podVolumes`	Log and workspace persistent volume claim names and name of the volumes attached to the pod.	Mandatory	`designer:`
			`  podVolumes:`
			`    - name: designer-pv-volume`
			
			`persistentVolumeClaim:`
			`        claimName: designer-managed-disk`
			`    - name: designer-log-volume`
			
			`persistentVolumeClaim:`
			`        claimName: designer-logs`

`designer.volumeMounts`	Name and mount path of the volumes to be attached to the Designer pods.	Mandatory	`volumeMounts:`
			`    - name: designer-pv-volume`
			`      mountPath: /designer/workspace`
			`    - name: designer-log-volume`
			`      mountPath: /designer/logs`

`designer.livenessProbe.path`	Designer liveness probe API path.	Mandatory	`/health`
`designer.livenessProbe.containerPort`	Port running the container.	Mandatory	`8888`
`designer.livenessProbe.initialDelay`	The liveness probe will be started after a given delay as specified here.	Mandatory	`20`
`designer.livenessProbe.checkInterval`	The interval between each liveness probe request.	Mandatory	`5`

`designer.livenessProbe.failureCount`	Number of liveness probe failures after which to mark the container as unstable or restart.	Mandatory	`5`
`designer.readinessProbe.path`	Designer readiness probe API path.	Mandatory	`/health`
`designer.readinessProbe.containerPort`	Port running the container.	Mandatory	`8888`
`designer.readinessProbe.initialDelay`	The readiness probe will be started after a given delay as specified here.	Mandatory	`20`
`designer.readinessProbe.checkInterval`	The interval between each readiness probe request.	Mandatory	`5`
`designer.readinessProbe.failureCount`	Number of readiness probe failures after which to mark the container as unstable or restart.	Mandatory	`5`
`designer.designerSecrets.enabled`	This enables providing the GWS Client ID and Secret as an input to the Designer pods. Kubernetes Secrets is used to store the GWS client credentials.	Mandatory	`true`
`designer.designerSecrets.secrets`	GWS Client ID and GWS Client Secret. Create a new GWS Client if it does not exist. A link to information on creating a new GWS Client is provided in the *Platform settings* section.	Mandatory	
`designer.service.enabled`	Set to `true` if the service must be created.	Optional	`true`
`designer.service.type`	Service type. Valid values are: `ClusterIP`, `NodePort`, `LoadBalancer`.	Mandatory	`NodePort`
`designer.service.port`	The Designer service port to be exposed in the cluster.	Mandatory	`8888`
`designer.service.targetPort`	The Designer application port running inside the container.	Mandatory	`http`
`designer.service.nodePort`	Port to be exposed in case service type is `NodePort`.	Mandatory for `designer.service.type=NodePort`.	`30180`

`designer.service.termination_grace_period`	The period after which Kubernetes starts to delete the pods after service termination.	Optional	`30` seconds.
`designer.ingress.enabled`	Set to `true` to enable ingress. Ingress should be enabled for all cases except for a lab/ demo setup.	Mandatory	`true`
`designer.ingress.apiVer`	The **apiVersion** of the ingress manifest to be deployed. Currently, `networking.k8s.io/v1beta1` and `networking.k8s.io/v1` are supported.	Optional	`networking.k8s.io/v1`
`designer.ingress.ingressClassName`	The ingress class name for the ingress deployed. Applicable only when `designer.ingress.apiVersion` is `networking.k8s.io/v1`.	Optional	
`designer.ingress.annotations`	Annotations added for ingress. The Designer UI requires Session Stickiness if the replica count is more than 1. Configure Session Stickiness based on the ingress controller type. Configuration specific to ingress such as Session Stickiness can be provided here.	Optional	
`designer.ingress.paths`	Ingress path	Mandatory	`[/]`
`designer.ingress.hosts`	Hostnames to be configured in ingress for the Designer service.	Mandatory	- .example.com - .blue.example.com - .green.example.com
`designer.ingress.tls`	TLS configuration for ingress.	Optional	`[]`
`designer.resources.limits.cpu`	Maximum amount of CPU that K8s allocates for the container.	Mandatory	`600m`
`designer.resources.limits.memory`	Maximum amount of memory that K8s allocates for the container.	Mandatory	`1Gi`
`designer.resources.requests.cpu`	Guaranteed CPU allocation for the	Mandatory	`500m`

	container.		
`designer.resources.requests.memory`	Guaranteed memory allocation for the container.	Mandatory	`512Mi`
`designer.securityContext.runAsUser`	This setting controls which user ID the containers are run with. This can be configured to run Designer as a non-root user. You can either use the **Genesys** user or arbitrary UIDs. Both are supported by the Designer base image. `500` is the ID of the Genesys user. The file system must reside within the Genesys user account in order to run Designer as a Genesys user. Change the NFS server host path to the Genesys user: `chown -R genesys:genesys`.	Optional	
`designer.securityContext.runAsGroup`	Controls which primary group ID the containers are run with. This can be configured to run Designer as a non-root user. You can either use the **Genesys** userGroup (GID - `500`) or arbitrary GIDs. Both are supported by the Designer base image.	Optional	

`designer.nodeSelector`	To allow pods to be scheduled based on the labels assigned to the nodes.	Optional	Default value:
			`nodeSelector: {}`
			Sample value:
			`nodeSelector:`
			`  :`

`designer.affinity`	The K8s standard node affinity and anti-affinity configurations can be added here. Refer to the this topic in the Kubernetes documentation site for sample values.	Optional	`{}`
`designer.tolerations`	Tolerations work with taints to ensure that pods are not scheduled on to inappropriate nodes. Refer to the	Optional	`[]`

	Taints and Tolerations topic in the Kubernetes documentation site for sample values.		
`designer.podDisruptionBudget.enabled`	Set to `true` if a pod disruption budget is to be created.	Optional	`false`
`designer.podDisruptionBudget.minAvailable`	The number of pods that should always be available during a disruption.	Optional	`1`
`designer.dnsPolicy`	The DNS policy that should be applied to the Designer pods.	Optional	
`designer.dnsConfig`	The DNS configuration that should be applied to the Designer pods.	Optional	
`designer.priorityClassName`	The priority class name that the pods should belong to.	Optional	
`designer.hpa.enabled`	Enables K8s Horizontal Pod Autoscaler (HPA). It automatically scales the number of pods based on average CPU utilization and average memory utilization. For more information on HPA refer to this topic in the Kubernetes documentation site.	Optional	`false`
`designer.hpa.targetCPUPercent`	The K8s HPA controller will scale up or scale down pods based on the target CPU utilization percentage specified here. It scales up or scales down pods between the range - `designer.deployment.replicaCount` and `designer.deployment.maxreplicaCount`.	Optional	`70`
`designer.hpa.targetMemoryPercent`	The K8s HPA controller will scale up or scale down pods based on the target memory utilization percentage specified here. It scales up or scales down pods between the range - `designer.deployment.replicaCount` and `designer.deployment.maxreplicaCount`.	Optional	`70`

`designer.labels`	Labels that will be added to the Designer pods.	Optional	`{}`
`designer.annotations`	Annotations added to the Designer pods.	Optional	`{}`
`designer.prometheus.enabled`	Set to `true` if Prometheus metrics must be enabled.	Optional	`false`
`designer.prometheus.tagName`	Label key assigned to the pods/service to filter out.	Optional	`service`
`designer.prometheus.tagValue`	Label value assigned to the pods/service to filter out.	Optional	`designer`
`designer.prometheus.instance`		Optional	`{{instance}}`
`designer.prometheus.serviceMonitor.enabled`	Set to `true if a service monitor resource is needed to monitor the pods through the Kubernetes service.`	Optional	`false`
`designer.prometheus.serviceMonitor.path`	The path in which the metrics are exposed.	Optional	`/metrics`
`designer.prometheus.serviceMonitor.interval`	The scrape interval specified for the Prometheus server. That is, the time interval at which the Prometheus server will fetch metrics from the service.	Optional	`10s`
`designer.prometheus.serviceMonitor.labels`	Labels to be specified for the service monitor resource.	Optional	
`designer.prometheus.alerts.enabled`	Set to `true if Prometheus alerts must to be created.`	Optional	`false`
`designer.prometheus.alerts.customAlerts`	Any custom alerts that are created must be specified here.	Optional	
`designer.prometheus.alerts.labels`	Labels to be specified for the alerts resource.	Optional	

`designer.prometheus.alerts`	Scenarios for which alerts need to be created.	Optional	**designer.prometheus.alerts**
			
			`containerRestartAlert:`
			`interval: 3m`
			`threshold: 5`
			
			`AlertPriority:`
			`CRITICAL`
			
			`MemoryUtilization:`

			interval: 1m threshold: 70 AlertPriority: CRITICAL endpointAvailable: interval: 1m AlertPriority: CRITICAL CPUUtilization: interval: 1m threshold: 70 AlertPriority: CRITICAL containerReadyAlert: interval: 1m readycount: 1 AlertPriority: CRITICAL WorkspaceUtilization: interval: 3m threshold: 80 workspaceClaim: designer-managed-disk AlertPriority: CRITICAL AbsentAlert: interval: 1m AlertPriority: CRITICAL Health: interval: 3m AlertPriority: CRITICAL WorkspaceHealth: interval: 3m AlertPriority: CRITICAL ESHealth: interval: 3m AlertPriority: CRITICAL GWSHealth: interval: 3m AlertPriority: CRITICAL

`designer.grafana.enabled`	Set to `true` if the Grafana dashboard is to be created.	Optional	`true`
`designer.grafana.labels`	Labels that have to be added to the Grafana ConfigMap.	Optional	
`designer.grafana.annotations`	Annotations that have to be added to the Grafana ConfigMap.	Optional	
`annotations`	Enables Kubernetes Annotations and adds it to all the resources that have been created. For more information, refer to the Annotations topic in the Kubernetes documentation site.	Optional	`{}`
`labels`	Any custom labels can be configured here. It is a key and value pair, for example, key:"value". These labels are added to all resources.	Optional	`{}`
`podLabels`	Labels that will be added to all application pods.	Optional	`{}`
`podAnnotations`	Annotations that will be added to all application pods.	Optional	`{}`

Designer ConfigMap settings

The following table provides information on the environment variables and service-level settings stored in the Designer ConfigMap.

Parameter	Description	Mandatory?	Default Value
`designer.designerConfig.create`	This enables providing environment variables as an input to the Designer pods. It uses a ConfigMap to store the environment variables.	Mandatory	`true`
`designer.designerConfig.envSettings.DES_PORT`	Designer port for container (`"port"` in flowsettings.json). The input should be a string, within double quotes.	Mandatory	`"8888"`
`designer.designerConfig.envSettings.DAS_HOST`	DAS hostname (`"applicationHost"` in flowsettings.json).	Mandatory	`das`

`designer.designerConfig.envs.DES_APPSERVER_PORT`	DAS port (`"applicationPort"` in flowsettings.json). The input should be a string, within double quotes.	Mandatory	`"80"`
`designer.designerConfig.envs.DES_DEPLOY_URL`	This is normally not changed. It is the relative path to the workspace on DAS. The default `"/workspaces"` should be used always (`"deployURL"` in flowsettings.json).	Mandatory	`"/workspaces"`
`designer.designerConfig.envs.DES_USE_HTCC`	Set to `"true"` so Designer works with GWS. If set to `"false"`, Designer defaults to a local mode and may be used temporarily if GWS is unavailable (`"usehtcc"` in flowsettings.json). Input should be "true" or "false".	Mandatory	`"false"`
`designer.designerConfig.envs.DES_HTCC_SERVER`	GWS server host (`"htccserver"` in flowsettings.json). For example, "gws.genhtcc.com". The input should be a string, within double quotes.	Mandatory	`" "`
`designer.designerConfig.envs.DES_HTCC_PORT`	GWS server port (`"htccport"` in flowsettings.json). For example, `"80"`. The input should be a string, within double quotes.	Mandatory	`" "`
`designer.designerConfig.envs.DES_ENABLE_ANALYTICS`	To enable or disable Designer Analytics (`"enableAnalytics"` in flowsettings.json). Input should be "true" or "false".	Optional	`"false"`
`designer.designerConfig.envs.DES_ES_URL`	Elasticsearch URL (`"esUrl"` in flowsettings.json). For example, "http://des-service:9200". The input should be a string, within double quotes.	Optional	`" "`
`designer.designerConfig.envs.DES_ES_HOST_NAME`	Elasticsearch Server Host Name (`"esServer"` in flowsettings.json). For	Optional	`" "`

	example, `"es-service"`). The input should be a string, within double quotes.		
`designer.designerConfig.envs.DES_ES_PORT`	Elasticsearch port (`"esPort"` in flowsettings.json). For example, `"9200"`. The input should be a string, within double quotes.	Optional	`" "`
`designer.designerConfig.envs.DES_FILE_LOGGING_ENABLED`	Enable file logging. If not enabled, Designer writes logs to stdout. Input should be "true" or "false".	Mandatory	`"false"`
`designer.designerFlowSettings.create`	Set to `true` to include the contents of the flowsettings.yaml file in a separate ConfigMap. Input should be true or false.	Optional	`false`
`designer.designerFlowSettingsConfigMap`	The flowsettings.yaml file should contain these keys, so that the file's contents will be included in the ConfigMap. Refer to the Updating the flowsettings file section in the *Deploy Designer* topic for more information on this.	Optional	`{}`

DAS deployment settings

The following table provides information on the DAS deployment settings. These settings are configured in the **das-values.yaml** file. DAS Deployment Settings

Parameter	Description	Mandatory?	Default Value
`das.deployment.replicaCount`	Number of pods to be created.	Mandatory	`2`
`das.deployment.maxreplicaCount`	The maximum number of replicas to be created. It is recommended to configure this setting if auto-scaling is used.	Optional	`10`
`das.deployment.strategy`	The deployment strategy to follow. This determines which type	Mandatory	`rollingupdate`

	of resources are deployed. Valid values are: `rollingupdate, blue-green, blue-green-ingress, blue-green-service, canary`. • **rollingupdate** - default Kubernetes update strategy where resources will be updated using the rolling upgrade strategy. • **blue-green** - for deploying and upgrading the DAS service using the blue-green strategy. • **blue-green-ingress** - for the blue-green upgrade, this is to create an ingress for the first time. • **blue-green-service** - for the blue-green upgrade, this is to create a service for the first time, and update the service during a service cutover. • **canary** - to deploy canary pods along with the blue-green pods.		
`das.deployment.color`	This is to deploy/ upgrade the DAS service using the blue-green upgrade strategy. Valid values are: `blue, green`.	Mandatory for `blue-green` and `blue-green-service` strategies.	
`das.deployment.type`	Type of Kubernetes controller. Valid values is: `StatefulSet` • **StatefulSet** - if the Designer workspace is stored in a remote cloud storage system, such as	Optional	`StatefulSet`

	Azure Files.		
`das.image.repository`	Docker repository that contains the images for DAS.	Mandatory	
`das.image.tag`	DAS image version.	Mandatory	
`das.image.pullPolicy`	DAS image pull policy (imagePullPolicy). Valid values are: `Always, IfNotPresent, Never`. • **Always** - always pull the image. • **IfNotPresent** - pull the image only if it does not already exist on the node. • **Never** - never pull the image.	Optional	`IfNotPresent`
`das.image.imagePullSecrets`	Secret name containing the credentials for authenticating access to the Docker repository.	Mandatory	

`das.podVolumes`	Provides the name of the volume and name of the persistent volume claim to be attached to the pods	Mandatory	`das:`
			`podVolumes:`
			`- name: workspace`
			
			`persistentVolumeClaim:`
			` claimName: designer-managed-disk`
			` - name: logs`
			
			`persistentVolumeClaim:`
			` claimName: designer-logs`

`das.volumes.podPvc.create`	This volume is usually created to mount a local disk to a DAS container for syncing data in case cloud storage is used for storing Designer files. This value has to be `true` or `false` depending on whether the local disk is needed or not	Optional	`false`
`das.volumes.podPvc.mountPath`	The path where the workspace volume is to be mounted inside the DAS container.	Optional	

`das.volumes.podPvc.claim`	Persistent volume claim name for the volume.	Optional	`local-workspace`
`das.volumes.podPvc.claimSize`	Size of the persistent volume claim for the pod. The persistent volume must be equal to or greater than this size.	Optional	
`das.volumes.podPvc.storageClass`	storageClassName provided in the persistent volume that is created for DAS (example, **nfs**).	Optional	
`das.volumes.podPvc.accessModes`	The read/write priveleges and mount priveleges of the volume claim with respect to the nodes. Valid types are: `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`. • **ReadWriteOnce** - the volume can be mounted as read-write by a single node. • **ReadOnlyMany** - the volume can be mounted as read-only by many nodes. • **ReadWriteMany** - the volume can be mounted as read-write by many nodes. For more information, refer to the access modes topic in the Kubernetes documentation site.	Optional	`ReadWriteOnce`

`das.volumeMounts`	The name of the volume and the mount path to be used by the pods.	Mandatory	`volumeMounts:`
			`- mountPath: /das/www/workspaces`
			`name: workspace`
			`- mountPath: /das/log`
			`name: logs`

`das.dasSecrets.enabled`	Set to `true` if Kubernetes secrets must be created to store	Optional	`false`

	keys/credentials/tokens.		
`das.dasSecrets.secrets`	Key value pairs containing the secret, such as, username and password.	Optional	
`das.livenessProbe.path`	DAS liveness probe API path.	Mandatory	`/health`
`das.livenessProbe.containerPort`	Port running the container.	Mandatory	`8081`
`das.livenessProbe.startupDelay`	The liveness probe will be started after a given delay as specified here.	Mandatory	`10`
`das.livenessProbe.checkInterval`	The interval between each liveness probe request.	Mandatory	`5`
`das.livenessProbe.failureCount`	Number of liveness probe failures after which to mark the container as unstable or restart.	Mandatory	`3`
`das.readinessProbe.path`	DAS readiness probe API path.	Mandatory	`/health`
`das.readinessProbe.containerPort`	Port running the container.	Mandatory	`8081`
`das.readinessProbe.startupDelay`	The readiness probe will be started after a given delay as specified here.	Mandatory	`10`
`das.readinessProbe.checkInterval`	The interval between each readiness probe request.	Mandatory	`5`
`das.readinessProbe.failureCount`	Number of readiness probe failures after which to mark the container as unstable or restart.	Mandatory	`3`
`das.service.enabled`	Set to `true` if the service must be created.	Optional	`true`
`das.service.type`	Service type. Valid values are: `ClusterIP`, `NodePort`, `LoadBalancer`.	Mandatory	`NodePort`
`das.service.port`	The DAS service port to be exposed in the cluster.	Mandatory	`80`
`das.service.targetPort`	The DAS application port running inside the container.	Mandatory	`http`
`das.service.nodePort`	Port to be exposed in	Mandatory if	`30280`

	case service type is `NodePort`.	`das.service.type` is `NodePort`.	
`das.service.termination.grace.period`	The period after which Kubernetes starts to delete the pods in case of deletion.	Optional	`30` seconds.
`das.ingress.enabled`	Set to `true` to enable ingress. Ingress should be enabled for all cases except for a lab/ demo setup.	Optional	`false`
`das.ingress.apiVersion`	The **apiVersion** of the ingress manifest deployed. Supported versions are, `networking.k8s.io/v1beta1` and `networking.k8s.io/v1`.	Optional	`networking.k8s.io/v1`
`das.ingress.ingressClassName`	The ingress class name for the ingress deployed. Applicable only when `das.ingress.apiVersion` is `networking.k8s.io/v1`.	Optional	
`das.ingress.annotations`	Annotations added for the ingress resources.	Optional	
`das.ingress.paths`	Ingress path.	Optional	`[/]`
`das.ingress.hosts`	Hostnames to be configured in ingress for the DAS service.	Mandatory if ingress is enabled.	
`das.ingress.tls`	TLS configuration for ingress.	Optional	`[]`
`das.resources.limits.cpu`	Maximum amount of CPU that K8s allocates for the container.	Mandatory	`600m`
`das.resources.limits.memory`	Maximum amount of memory that K8s allocates for the container.	Mandatory	`1Gi`
`das.resources.requests.cpu`	Guaranteed CPU allocation for the container.	Mandatory	`400m`
`das.resources.requests.memory`	Guaranteed memory allocation for the container.	Mandatory	`512Mi`
`das.securityContext.runAsUser`	This setting controls which user ID the containers are run with and can be configured	Optional	

	to run DAS as a non-root user. You can either use the **Genesys** user or arbitrary UIDs. Both are supported by the DAS base image. `500` is the ID of the Genesys user. For more information refer to the Security Context topic in the Kubernetes documentation site.		
`das.securityContext.runAsGroup`	This setting controls which primary group ID the containers are run with and can be configured to run DAS as a non-root user. You can either use the **Genesys** userGroup (GID - `500`) or arbitrary GIDs. Both are supported by the DAS base image.	Optional	

`das.nodeSelector`	To allow pods to be scheduled based on the labels assigned to the nodes.	Optional	Default value:
			
			`nodeSelector: {}` Sample value:
			
			`nodeSelector:`
			`  :`

`das.affinity`	The K8s standard node affinity and anti-affinity configurations can be added here. Refer to the this topic in the Kubernetes documentation site for sample values.	Optional	`{}`
`das.tolerations`	Tolerations work with taints to ensure that pods are not scheduled on to inappropriate nodes. Refer to the Taints and Tolerations topic in the Kubernetes documentation site for sample values.	Optional	`[]`
`das.podDisruptionBudget.enabled`	Set to `true` if a pod disruption budget is to be created.	Optional	`false`
`das.podDisruptionBudget.minAvailable`	The number of pods that should always be available during a disruption.	Optional	`1`

`das.dnsPolicy`	The DNS policy that should be applied to the DAS pods.	Optional	
`das.dnsConfig`	The DNS configuration that should be applied to the DAS pods.	Optional	
`das.priorityClassName`	The priority class name that the pods should belong to.	Optional	
`das.hpa.enabled`	Set to `true` if a K8s Horizontal Pod Autoscaler (HPA) is to be created.	Optional	`false`
`das.hpa.targetCPUPercent`	The K8s HPA controller will scale up/down pods based on the target CPU utilization percentage specified. It scale up/down pods between the range `deployment.replicaCount` to `deployment.maxReplicas`	Optional	`75`
`das.hpa.targetMemoryPercent`	The K8s HPA controller will scale up or scale down pods based on the target CPU utilization percentage specified here. It scales up or scales down pods between the range - `deployment.replicaCount` and `deployment.maxReplicas`.	Optional	`70`
`das.labels`	Labels that will be added to the DAS pods.	Optional	`{}`
`das.annotations`	Annotations added to the DAS pods.	Optional	`{}`
`das.prometheus.enabled`	Set to `true` if Prometheus metrics must be enabled.	Optional	`false`
`das.prometheus.tagName`	Label key assigned to the pods/service to filter out.	Optional	`service`
`das.prometheus.tagValue`	Label key assigned to the pods/service to filter out.	Optional	`designer`
`das.prometheus.pod`		Optional	`{{pod}}`
`das.prometheus.instance`		Optional	`{{instance}}`
`das.prometheus.serviceMonitor`	Set to `true` if a service monitor	Optional	`false`

	resource is needed to monitor the pods through the Kubernetes service.		
das.prometheus.serviceMonitor.path	The path in which the metrics are exposed.	Optional	/metrics
das.prometheus.serviceMonitor.interval	The scrape interval specified for the Prometheus server. That is the time interval at which the Prometheus server will fetch metrics from the service.	Optional	10s
das.prometheus.serviceMonitor.labels	Labels to be specified for the service monitor resource.	Optional	
das.prometheus.alerts.enabled	Set to true if Prometheus alerts must to be created.	Optional	false
das.prometheus.alerts.labels	Labels to be specified for the alerts resource.	Optional	
das.prometheus.alerts.customalerts	Any custom alerts that are needed must be specified here.	Optional	
das.prometheus.alerts.	Scenarios for which alerts need to be created.	Optional	**das.prometheus.alerts.** containerRestartAlert: interval: 3m threshold: 5 AlertPriority: CRITICAL MemoryUtilization: interval: 1m threshold: 75 AlertPriority: CRITICAL endpointAvailable: interval: 1m AlertPriority: CRITICAL CPUUtilization: interval: 1m threshold: 75 AlertPriority: CRITICAL containerReadyAlert: interval: 5m readycount: 1

```
AlertPriority:
CRITICAL

rsyncContainerReadyAlert:
        interval: 5m
        readycount: 1

AlertPriority:
CRITICAL

WorkspaceUtilization:
        interval: 3m
        threshold: 70

workspaceClaim:
designer-managed-disk

AlertPriority:
CRITICAL
      AbsentAlert:
        interval: 1m

AlertPriority:
CRITICAL

LocalWorkspaceUtilization:
        interval: 3m
        threshold: 70

AlertPriority:
CRITICAL
      Health:
        interval: 3m

AlertPriority:
CRITICAL

WorkspaceHealth:
        interval: 3m

AlertPriority:
CRITICAL
      PHPHealth:
        interval: 3m

AlertPriority:
CRITICAL
      ProxyHealth:
        interval: 3m

AlertPriority:
CRITICAL
      PhpLatency:
        interval: 1m
        threshold: 10

AlertPriority:
CRITICAL
      HTTPLatency:
        interval: 1m
```

			threshold: 60 AlertPriority: CRITICAL HTTP4XXCount: interval: 5m threshold: 100 AlertPriority: CRITICAL HTTP5XXCount: interval: 5m threshold: 100 AlertPriority: CRITICAL
das.grafana.enabled	Set to `true` if the Grafana dashboard is to be created.	Optional	`true`
das.grafana.labels	Labels that must be added to the Grafana ConfigMap.	Optional	
das.grafana.annotations	Annotations that must be added to the Grafana ConfigMap.	Optional	
`annotations`	Enables Kubernetes Annotations and adds it to all the resources that have been created. For more information, refer to the Annotations topic in the Kubernetes documentation site.	Optional	`{}`
`labels`	Any custom labels can be configured here. It is a key and value pair, for example, key:"value". These labels are added to all resources.	Optional	`{}`
`podLabels`	Labels that will be added to all application pods.	Optional	`{}`
`podAnnotations`	Annotations that will be added to all application pods.	Optional	`{}`

DAS ConfigMap settings

Parameter	Description	Mandatory?	Default Value

`das.dasConfig.create`	This setting enables providing environment variables as an input to the DAS pods. It uses a ConfigMap to store the environment variables.	Mandatory	`true`
`das.dasConfig.envs.DAS_FILE_LOGGING_ENABLED`	Enables file logging. DAS supports only std out logging. This should always be set to `false`. Input should be "true" or "false".	Mandatory	`"false"`
`das.dasConfig.envs.DAS_STDOUT_LOGGING_LEVEL`	Enables log levels. Valid values are: `"FATAL", "ERROR", "WARN", "INFO", "DEBUG", "TRACE"`.	Optional	`"DEBUG"`
`das.dasConfig.envs.DAS_STDOUT_LOGGING_ENABLE`	Enables standard output console logging. Input should be "true" or "false".	Mandatory	`"true"`
`das.dasConfig.envs.DAS_SERVICES_ELASTICSEARCH_ENABLED`	To enable Designer Analytics. This configuration is required for DAS to initialize ES templates. Input should be "true" or "false".	Mandatory	`"false"`
`das.dasConfig.envs.DAS_SERVICES_ELASTICSEARCH_HOST`	Elasticsearch server host name with an http:// prefix. For example, `"http://es-service"`. The input should be a string within double quotes.	Optional	`" "`
`das.dasConfig.envs.DAS_SERVICES_ELASTICSEARCH_PORT`	Elasticsearch port. For example, `"80"`. The input should be a string, within double quotes.	Optional	`" "`
das.dasConfig.envs.DAS_ELASTIC_URL	Elasticsearch URL for basic authentication. It should contain the URL with an http or https prefix accompanied with the port number (for example, http://es-service:80). The input should be a string within double quotes. This setting is mandatory when `DAS_SERVICES_ELASTICSEARCH_ENABLED` is set to `true`.	Optional	`" "`
das.dasConfig.envs.DAS_ELASTIC_SEC_	Elasticsearch secondary	Optional	`" "`

	region URL for basic authentication. It should contain the URL with an http or https prefix accompanied with the port number (for example, http://es-service:80). The input should be a string within double quotes. is an integer starting from 1. This setting is mandatory when secondary regions are configured. For example, `das.dasConfig.envs.DAS_ELASTIC_URL_1`.		

Post deployment Designer configuration settings

Post deployment, Designer configuration is managed from the following 3 locations:

Flow settings

Flow Settings is used for controlling global Designer settings that are applicable to all tenants and it contains bootstrap configuration settings such as port, GWS info, and DAS URL.

Configuration path - `/workspace/designer/flowsettings.json`.

This will be configured using the helm install. Refer to the Update the flowsettings.json file section for information on updating the **flowsettings.json** file.

Tenant settings

These are tenant specific settings if the Designer service is configured with multi-tenancy .

Configuration path - `workspace//config/tenantsettings.json`.

The user should logout and log back in after any changes to the **tenantsettings.json** file. The Designer UI will continue to show the older features until the user logs out and logs back in.

Tenant specific settings are configured by directly editing the file in the above path.

DesignerEnv transaction list

The **DesignerEnv** transaction list is available in Configuration Server (`Tenant/Transactions/DesignerEnv`). This is mostly used to control the run-time settings. Any change to the **DesignerEnv** transaction list does not require the application to be published again or a new build for the application.

The user should log out and log back in for the changes to reflect in the Designer UI.

The **DesignerEnv** transaction list is configured using Agent Setup.

Post deployment configuration settings reference table

Category: Analytics					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`enableAnalytics` (optional)	Yes	Yes	No	This flag enables or disables the analytics feature.	Sample value: `true` Default value: `false`
`esUrl` (optional)	Yes	Yes	No	Elasticsearch URL	Sample value: http://es-spot.usw1.genhtcc.com:80
`esServer` (optional)	Yes	Yes	No	Elasticsearch server host name (for example, `es-service`).	Sample value: es-spot.usw1.genhtcc.com
`esPort` (optional)	Yes	Yes	No	Elasticsearch port.	Sample value: `80`
`ReportingURL` (optional)	No	No	Yes Section: `reporting`	URL of Elasticsearch where Designer applications will report data.	Sample value: http://es-spot.usw1.genhtcc.com:80
`esMaxQueryDuration` (optional)	Yes	Yes	No	The maximum time range (in days) to query in Designer Analytics. Each day's data is stored in a separate index in Elasticsearch.	Sample value: `90` Default value: `90`
`sdrMaxObjCount` (optional)	Yes	Yes	No	The maximum count of nested type objects that will be captured in SDRs. When set to -1, which is the default value, no objects will be trimmed. All	Sample value: `20`

				the *milestones* or *activities* visited in runtime are expected to be captured in an SDR.	
`SdrTraceLevel` (optional)	Yes	Yes	No	Value are: • **100** — Debug level and up. Currently, there are no Debug messages. • **200** — Standard level and up. This setting will show all blocks that are entered during a call in the `blocks` array. • **300** — Important level and up. This setting filters out all blocks from the `blocks` array, except those containing data that will change from call to call (such as the Menu block and User Input block).	Sample value: `300` Default value: `300`
Category: Audit					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`enableESAuditLogs` (optional)	Yes	Yes	No	Enable or disable audit logs captured in Elasticsearch.	Sample value: `false` Default value: `false`
`enableFSAuditLogs` (optional)	Yes	Yes	No	Enable or Disable audit logs captured in the file system under the **logs** directory or in standard output.	Sample value: `true` Default value: `true`
`maxAppSizeCompare` (optional)	Yes	Yes	No	The maximum size of data object for which a difference will be captured in the audit logs, value in bytes. That is, the difference between the Designer object's old value and new value.	Sample value: `1000000` Default value: `1000000`
`enableReadAuditLogs` (optional)	Yes	Yes	No	Control whether reading of Designer objects is captured in audit trails. If enabled any Designer object viewed in the UI will be recorded in the audit logs.	Sample value: `false` Default value: `false`
Category: Authorization					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`disableRBAC` (optional)	Yes	Yes	No	Controls if Designer reads and enforces permissions associated with the logged in user's roles.	Sample value: `false` Default value: `false`
`rbacSection` (optional)	Yes	Yes	No	In a Role object, the name of the	Sample value: `CfgGenesysAdministratorServer`

				section within the Annex where the privileges are stored.	Default value: `CfgGenesysAdministratorServer`
`disablePBAC` (optional)	Yes	Yes	No	Controls if Designer allows partitioning of the Designer workspace and restricts a user's access to Designer objects in the user's partitions.	Sample value: `false` Default value: `false`
Category: Collaboration					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`locking` (optional)	Yes	No	No	The type of locking used, in an editing session for applications, modules, or data tables. Valid values are: `file, redis, none`. • **none** - resources are not locked and can be edited simultaneously by multiple users which can result in one user overwriting another user's changes. • **file** - uses files to keep track of locks and relies on shared storage (for example, NFS) to	Sample value: `file` Default value: `file`

				make lock files available to each Designer pod. Lock files are stored in the same location as the user's Designer workspace. • **redis** - uses Redis for storing resource locks and is recommended for production environments.	
Category: DAS					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`applicationHost` (mandatory)	Yes	No	No	The server name Designer uses to generate the URL to the application. ORS and MCP fetch the application code and other resources from this URL.	Sample value: `das.usw1.genhtcc.com` Default value: `localhost`
`applicationPort`	Yes	No	No	The corresponding port to be used with applicationHost.	Sample value: `80` Default value: `80`
`deployURL`	Yes	No	No	This is normally not changed. It is the relative path to the workspace on DAS.	Sample value: `/workspace` Default value: `/workspace`
Category: Digital					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`rootsSRL` (optional)	Yes	Yes	No	If specified, this is used to filter which Root Categories to display when selecting Standard Responses.	Sample value: Any REGular EXpression (REGEX).
`maxFlowEntryCount` (optional)	Yes	No	Yes Section: `flowsettings`	Specify how many times the same application can process a specific digital interaction.	Sample value: `20` Default value: `20`
Category: External APIs					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`httpProxy` (optional)	Yes	Yes	Yes Secion: `flowsettings`	Specify the proxy used for external requests and nexus API calls (if `enable_proxy` is `true`).	Sample value: [http://vpcproxy-000-int.geo.genprim.
`redundantHttpProxy` (optional)	Yes	Yes	Yes Section: `flowsettings`	Specify the backup proxy used for external requests and nexus API calls (if `enable_proxy` is `true`), when httpProxy is down.	Sample value: [http://vpcproxy-001-int.geo.genprim.
Category: Features					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`features`	Yes	Yes	No	This is an object. See the **5.5 Features** section for a list of supported features.	Default value:
					
					`{`
					`  nexus: true,`
					
					`enableBulkAudioImport: true`
					`}`

Category: GWS					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`usehtcc`	Yes	No	No	Set to `true` so that Designer works with GWS. If set to `false`, Designer defaults to a local mode and may be used temporarily if GWS is unavailable.	Sample value: `true` Default value: `false`
`htccServer`	Yes	No	No	GWS Server	Sample value: gws-usw1-int.genhtcc.com Default value: gws-usw1-int.genhtcc.com
`htccport`	Yes	No	No	GWS port.	Sample value: `80` Default value: `80`
`ssoLoginUrl`	Yes	No	No	URL of GWS authentication UI. Designer redirects to this URL for authentication.	Sample value: https://gws-usw1.genhtcc.com Default value: https://gws-usw1.genhtcc.com
`maxConcurrentHTCCRequest` (optional)	Yes	No	No	For batch operations to GWS, the max number of concurrent requests that Designer will send to GWS.	Sample value: `5` Default value: `5`
`batchOperationResultTTL` (optional)	Yes	No	No	For batch operations to GWS, the time, in milliseconds, for which duration Designer stores the results of a batch operation on the server, before deleting them.	Sample value: `100000` Default value: `100000`
Category: Help					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`docsMicroserviceURL` (optional)	Yes	No	No	URL for Designer documentation.	Default value: https://docs.genesys.com/Documentation/

					PSAAS/Public/ Administrator/ Designer
Category: IVR					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`recordingType` (optional)	Yes	Yes	No	Specify the recording type to be used in Record block. Set as `GIR`. If the option is missing or blank, `Full Call Recording` type will be used.	Sample value: `GIR` Default value: `GIR`
Category: Logging					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`logging: { designer: { level: debug }, audit: { level: trace}, auditdebug: { level: debug }, cli: { level: debug } }`
 	
(optional)
	Yes	No	No	Specify Designer log levels. Each field has valid values: `trace, debug, info, warn, error, or fatal`.
				• **designer** - log level of Designer.
				• **audit** - log level of audit.
				• **auditdebug** - log level of audit debug, this will log detailed audit information.
				• **cli** - log level for cli commands executed on Designer.
					Sample value:
					`logging: { designer: { level: debug}, audit: { level: trace }, auditdebug: { level: debug}, cli: { level: debug } }`
					
					Default value:
					`logging: { designer: { level: debug }, audit: { level: trace }, auditdebug: { level: debug }, cli: { level: debug } }`

Category: Nexus					

Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`url` (optional)	No	No	Yes Section: `nexus`	URL of Nexus that typically includes the API version path. For example, https://nexus-server/nexus/api/v3.	Default value: http://nex-dev.usw1.genhtcc.com
`password` (optional)	No	No	Yes Section: `nexus`	The Nexus x-api-key created by Nexus deployment.	Default value: `dc4qeiro13nsof569dfn234smf`
`enable_proxy` (optional)	No	No	Yes Section: `nexus`	Boolean value to indicate if httpProxy is used to reach Nexus. Default value: `false`	
profile (optional)	No	No	Yes Section: `nexus`	Enable Contact Identification via Nexus (for example, to enable Last Called Agent routing).	
Category: Process					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`port`	Yes	No	No	Designer process port in the container. Normally, the default value should be left as is.	Sample value: `8888` Defualt value: `3000`
Category: Provisioning					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`primarySwitch` (optional)	Yes	Yes	No	Specify the primary switch name if more than one switch is defined for the tenant. Designer fetches and works with route points from this	Default value: `us-west-1`

				switch.	
Category: Routing					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`ewtRefreshTimeout` (optional)	NO	No	Yes Section: `flowsettings`	Specify the interval (in seconds) at which to refresh the Estimated Waiting Time when routing an interaction.	Sample value: `5` Default value: `1`
Category: Redis					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value

`redis: { host: "", port: "", tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }`
 	
(optional)
	Yes	No	No	Used by Designer for resource index caching and multi-user collaboration locks on Designer resources.
				
				It is a separate object that contains:
				
				• **host** - Redis host name.
				• **port** - Redis port.
				• **tlsEnabled** - TLS enabled or not.
				• **lockTimeout** - Timeout, in seconds, before a resource lock is released for an editing session of applications, modules, or data tables.
				• **listTimeout** - The cache expiry timeout (in
					Sample value:
					
					`redis: { host: "", port: "", tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }`
					
					Default value:
					
					`redis: { host: redis.server.genhtcc.com, port: 6379, tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }`

				seconds) of the application list and shared modules list. By default, it is 30 minutes. That is, any new application/ modules created in the UI will be seen in the listing page after 30 mins. It can be reduced to a smaller value. This is to improve the page loading performance of the Applications and Shared Modules page. A better performance is achieved with a higher value.	
Category: Security					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`zipFileSizeLimitInMegaBytes` (optional)	Yes	Yes	No	Defines the maximum zipFile size limit (in megabytes) during bulk audio import.	Sample value: `50`
`disableCSRF` (optional)	Yes	Yes	No	Disable CSRF attack protection. For more information, refer to this	Sample value: `false` Default value: `false`

				topic in the CWE site. By default, CSRF attack protection is enabled. It can be disabled by setting this flag to `true`.	
`disableSecureCookie` (optional)	Yes	No	No	Disables the secure cookies header.	Sample value: `false` Default value: `false`
Category: Session					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`idleTimeout` (optional)	Yes	Yes	No	Idle timeout, in seconds, before a user session is terminated while editing applications, modules, or data tables.	Sample value: `840` Default value: `840`
`lockTimeout` (optional)	Yes	Yes	No	Timeout, in seconds, before a resource lock is released, for an editing session of applications, modules, or data tables.	Sample value: `120` Default value: `120`
`lockKeepalive` (optional)	Yes	Yes	No	Interval, in seconds, before the client sends a ping to the server, to refresh the lock for an editing session of applications, modules, or data tables.	Sample value: `15` Default value: `15`
Category: Workflow					
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	Description	Value
`maxBuilds` (optional)	Yes	Yes	No	Specify the maximum number of builds allowed per application.	Sample value: `20` Default value: `20`

`enablePTE` (optional)	No	No	Yes Section: `flowsettings`	Boolean value to indicate if PTE objects are enabled at runtime.	Sample value: `true` Default value: `false`

Features

The features specified in this section are configured under the `features` object in the **flowsettings.json** file or the **tenantsettings.json** file.

For example,

```
"features": {
          "nexus": true,
..
}
```

> **Important**
>
> These features are configured only in the **flowsettings.json** file and the **tenantsettings.json** file, and not in the **DesignerEnv** transaction list.

Category	Feature Setting Name	Mandatory	flowsettings.json	tenantsettings.json	Description	Default Value
Audio	`enableBulkAudioImport`	Optional	Yes	Yes	Enable/ disable the bulk audio import feature.	`false`
	`grammarValidation`	Optional	Yes	yes	If this feature is enabled, Designer will validate invalid grammar files during grammar upload and you can upload only valid grammar files (GRXML or Nuance compiled binary grammar files).	`false`

	`externalAudioSupport`	Optional	Yes	Yes	If this feature is enabled, a new audio type, External Audio, is available in the Play Message block. It accepts a single variable that contains a URL to the audio resource. MCP will fetch this resource directly and play it. The only supported value of **Play As** is *Audio URI*. There is no automatic language switching for this audio type.	`false`
Nexus	`nexus`	Optional	Yes	Yes	Enable/ disable the Nexus feature.	`false`
Survey	`survey`	Optional	Yes	Yes	Enable/ disable the survey feature.	`true`
UI Plugins	`plugins`	Optional	Yes	Yes	Plugin configuration details. (Steps are given below the table.)	`{}`
	`plugins`	Optional	Yes	Yes	Enable or disable the plugin feature.	`false`
Milestone	`enableImplicitOptionalMilestones`	Optional	Yes	Yes	Enable reporting	`false`

					each Shared Module call as an internal milestone. If disabled, Shared Module calls will not generate a milestone.	
Bots	`enableDialogflowCXBot`	Optional	Yes	Yes	When enabled, Dialogflow CX bot type is added to the bot registry and available for selection in the **Bot provider** drop-down when you configure a new bot.	`false`
Multisite Routing	`multisiteRouting`	Optional	Yes	Yes	Enables the **Override DN** option in the **Advanced** > **Targeting** section of the **Route Call** block to Force Route the interaction to a specified DN.	`false`

Adding a UI plugin to Designer

1. Add the `plugins` array object in the **flowsettings.json** file (*/ofs/designer/flowsettings.json*).
   The `plugins` object contains all the input properties for the plugin app. This is a required property.
   Whenever there is a change in this object, refresh the browser for the changes to take effect.
   Example:

```
"plugins": [
    {
      "url": "http://genesysexample.com/",
      "displayName": "Nexus PII Management",
```

```
        "placement": "messageCollections",
        "id":   "nexuspii",
        "mappings": {
              "prod": {
                  "G1-AUS4": "https://genesysexample.com/admin/ux"
              },
              "staging": {
                  "G1-USW1": "http://genesysexample.com/"
              },
         }
    },
    {
       ...
    }]
```

2. Add the `csplist` array object in the **flowsettings.json** file (*/ofs/designer/flowsettings.json*).
   The `cspList` object contains the URL forms to be allowed by Designer's security policy. This is a required property. Whenever there is a change in this object, re-start the node container for the changes to take effect.
   Example:
   If the URL is `http://genesysexample.com/`, the `cspList` would be:
   `"cspList": ["*.genexample1.com:*", "*.genexample2.com:*", "*.genexample3.com:*"]`
3. Turn on the `plugins` and `nexus` feature flags in the Designer **tenantSettings.json** file (*/ofs//config/tenantSettings.json*).
   This is a required property. Whenever there is a change in this object, log out of Designer and log in again for the changes to take effect.

   > **Important**
   > If you want to enable the plugins feature for all tenants, add this feature flag in the **flowsettings.json** file. The feature is enabled for all the tenants under that bucket.

   Example:

```
{
    "features": {
        "plugins": true,
        "nexus": true
}}
```

4. Add the `url_` property under the `plugins` section, in Agent Setup. If there is no `plugins` section, create one. This section is for the tenant URL override. If the `DesignerEnv` setting (*Transactions/Internal/DesignerEnv*) is not provided, the plugin URL from the **flowsettings.json** file is considered.
   This is an optional property. Whenever there is a change in this object, log out of Designer and log in again for the changes to take effect.
   Example:
   `{"url_" : "https://plugin-genesysexample.com"}`

Platform / Configuration Server and GWS settings for Designer

Contents

- 1 Create roles for Designer
- 2 Update the DesignerEnv transaction list
- 3 Platform settings
- 4 GWS configuration

- Administrator

Learn about the Configuration Server objects and settings required for Designer.

Related documentation:

-
-
-

RSS:

- For private edition

Create roles for Designer

Designer uses roles and access groups to determine permissions associated with the logged-in user. To enable this, you must make these changes in GAX or CME.

Designer supports a number of bundled roles suitable for various levels of users.

- **Designer Developer** - Most users fall into this category. These users can create Designer applications, upload audio, and create business controls. They have full access to Designer Analytics.
- **Designer Business User** - These users cannot create objects but they can manage them (for example, upload audio, change data tables, and view analytics).
- **Designer Analytics** - These users only have access to Designer Analytics.
- **Designer Admin** - These users can set up and manage partitions associated with users and Designer objects.
- **Designer Operations** - Users with this role have full access to all aspects of the Designer workspace. This includes the **Operations** menu (normally hidden), where they can perform advanced operational and maintenance tasks.

To create these roles, import the **.conf** files included in the **Designer Deployment** package. They are located in the **packages/roles/** folder.

In addition, ensure the following for user accounts that need access to Designer:

- The user must have read permissions on its own Person object.
- Users must be associated with one or more roles via access groups.
- The on-Premises user must have at least read access on the user, access group(s), and roles(s).
- The access groups must have read/write permissions to the Agent Setup folders - Scripts and Transactions.

Update the DesignerEnv transaction list

Designer requires a transaction list for configuration purposes as described in other sections of this document. The **DesignerEnv** transaction list is automatically created in on logging onto Designer.

1. Edit any values according to the descriptions provided in the Post deployment configuration settings reference table.
2. Save the list.
3. Ensure Designer users have at least read access to the **DesignerEnv** transaction list.

Platform settings

The platform settings listed below must be configured if the Designer application is used for voice calls.

Component	Config Key	Value	Description
SIP Switch -> Voip Services -> msml service	userdata-map-format	sip-headers-encoded	Option needs to set to pass JSON data as user data in SIPS.
SIP Switch -> Voip Services -> msml service	userdata-map-filter	*	To allow userdata passing to MSML service.
SIPServer --> TServer	divert-on-ringing	false	RONA is handled by the platform.
	agent-no-answer-timeout	12	
	agent-no-answer-action	notready	
	agent-no-answeroverflow	""	No value, empty.
	after-routing-timeout	24	
	sip-treatments-continuous	true	
	msml-record-support	true	To allow routed calls recording via the Media Server.
Switch object annex --> gts	ring-divert	1	
ORS --> orchestration	new-session-on-reroute	false	Required for SIPS Default Routing (Default Routing handling (Voice)).
MCP	[vxmli] transfer.allowed	TRUE	Required for Transfer block (allows VXML Transfer in MCP).

MCP	[cpa] outbound.method	NATIVE	Required for Transfer block (allow CPA detection for Transfer).
UCS	[cview] enabled	TRUE	Enables Customer Context Services.

GWS configuration

Ensure that the following steps are performed in GWS:

- **Add Contact Center**—Create a contact center in GWS, if it is not already created.
- **Create API Client**—Create new GWS client credentials, if they are not already created.

For more information, see Provision Genesys Web Services and Applications in the GWS documentation.

Deploy Designer

Contents

- 1 Assumptions
- 2 Preparation
  - 2.1 Set up Ingress
  - 2.2 Set up Application Gateway (WAF) for Designer
  - 2.3 Storage
  - 2.4 Set up Secrets
- 3 Deployment strategies
- 4 Rolling Update deployment
  - 4.1 Designer
  - 4.2 DAS
- 5 Blue-Green deployment
  - 5.1 Designer
  - 5.2 DAS
- 6 Canary
  - 6.1 Deployment
  - 6.2 Cleaning up
- 7 Validations and checks
- 8 Post deployment procedures
  - 8.1 Updating the flowsettings file

Learn how to deploy Designer 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.

Preparation

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

Before you deploy Designer and DAS using Helm charts, complete the following preparatory steps:

1. Ensure the Helm client is installed.
2. Set up an Ingress Controller, if not already done.
3. Setup an NFS server, if not already done.
4. Create Persistent Volumes - a sample YAML file is provided in the Designer manifest package.
5. Download the Designer and DAS docker images and push to the local docker registry.

6. Download the Designer package and extract to the current working directory.
7. Configure Designer and DAS value overrides (**designer-values.yaml** and **das-values.yaml**); ensure the mandatory settings are configured. If the Blue-Green deployment process is used, Ingress settings are explained in the Blue-Green deployment section.

> **Important**
>
> Depending on the Kubernetes platform or the container orchestration platform that you are deploying Designer on, you might have to carry out some additional steps specific to that platform. For more information, navigate to the required topic in the **Kubernetes platform specific information** section on the About page.

Set up Ingress

Given below are the requirements to set up an Ingress for the Designer UI:

- Cookie name - designer.session.
- Header requirements - client IP & redirect, passthrough.
- Session stickiness - enabled.
- Allowlisting - optional.
- TLS for ingress - optional (should be able to enable or disable TLS on the connection).

Set up Application Gateway (WAF) for Designer

Designer Ingress must be exposed to the internet using Application Gateway enabled with WAF.

When WAF is enabled, consider the following exception in the WAF rules for Designer:

- Designer sends a JSON payload with data, for example, `{profile . {} }`. Sometimes, this is detected as `OSFileAccessAttempt`, which is a false positive detection. Disable this rule if you encounter a similar issue in your WAF setup.

Storage

Designer storage

Designer requires storage to store designer application workspaces. Designer storage is a shared file storage that will be used by the Designer and DAS services.

> **Important**
>
> This storage is critical. Ensure you take backups and snapshots at a regular interval,

> probably, each day.

A Zone-Redundant Storage system is required to replicate data from the RWX volumes and must be shared across multiple pods:

- Capacity - 1 TiB
- Tier - Premium
- Baseline IO/s - 1424
- Burst IO/s - 4000
- Egress Rate - 121.4 MiBytes/s
- Ingress Rate - 81.0 MiBytes/s

DAS storage

If the Designer workspace is stored in a cloud storage system such as Azure Files, then the data must be synced to the DAS pods using the Designer-Sync service. In this case, DAS must use the `StatefulSet` deployment type. In the DAS StatefulSet pods, each pod must be attached to a premium SSD disk to store the workspace.

- Size - > 500GiB
- Max IOPS (Max IOPS w/ bursting) - 2,300 (3,500)
- Max throughput (Max throughput w/ bursting) - 150 MB/second (170 MB/second)

Permission considerations for Designer and DAS storage

NFS

For NFS RWX storages, the mount path should be owned by `genesys:genesys`, that is, `500:500` with `0777` permissions. It can be achieved by one of the below methods:

- From the NFS server, execute the **chmod -R 777** and **chown -R 500:500** commands to set the required permissions.
- Create a dummy Linux based pod that mounts the NFS storage. From the pod, execute the **chmod -R 777** and **chown -R 500:500** commands. This sets the required permissions. However, this method might require the Linux based pods to be run as privileged.

SMB / CIFS

For SMB / CIFS based RWX storages, for instance, Azure file share, the below `mountOptions` must be used in the **StorageClass** or the **PersistentVolume** template:

`mountOptions`

```
- dir_mode=0777
- file_mode=0777
```

```
- uid=500
- gid=500
- mfsymlinks
- cache=strict
```

Set up Secrets

Secrets are required by the Designer service to connect to GWS and Redis (if you are using them).

GWS Secrets:

- GWS provides a Client ID and secrets to all clients that can be connected. You can create Secrets for the Designer client as specified in the *Set up secrets for Designer* section below.

Redis password:

- If Designer is connected to Redis, you must provide the Redis password to Designer to authenticate the connection.

Set up Secrets for Designer

Use the `designer.designerSecrets` parameter in the **values.yaml** file and configure Secrets as follows:

```
designerSecrets:
  enabled: true
  secrets:
    DES_GWS_CLIENT_ID: xxxx
    DES_GWS_CLIENT_SECRET: xxxx
    DES_REDIS_PASSWORD: xxxxx
    DES_ELASTIC_USERNAME: "xxxx"
    DES_ELASTIC_PASSWORD: "xxxxx"
```

Set up Secrets for DAS

Use the `das.dasSecrets` parameter in the **values.yaml** file and configure Secrets as follows:

```
dasSecrets:
  enabled: true
  secrets:
        DAS_ELASTIC_USERNAME : "xxxxx"
        DAS_ELASTIC_PASSWORD : "xxxxx"
        DAS_ELASTIC_USERNAME_1 : "xxxxx"
        DAS_ELASTIC_PASSWORD_1 : "xxxxx"
        DAS_ELASTIC_USERNAME_2 : "xxxxx"
        DAS_ELASTIC_PASSWORD_2 : "xxxxx"
```

Deployment strategies

Designer supports the following deployment and upgrade strategies:

- Rolling Update (default).
- Blue-Green (recommended).

DAS (Designer Application Server) supports the following deployment and upgrade strategies:

- Rolling Update (default).
- Blue-Green (recommended).
- Canary (must be used along with Blue-Green and is recommended in production).

For full descriptions of the deployment and upgrade strategies, see Upgrade strategies in the *Setting up Genesys Multicloud CX Private Edition* guide.

Rolling Update deployment

The rolling deployment is the standard default deployment to Kubernetes. It works slowly, one by one, replacing pods of the previous version of your application with pods of the new version without any cluster downtime. It is the default mechanism of upgrading for both Designer and DAS.

Designer

To perform the initial deployment for a rolling upgrade in Designer, use the Helm command given below. The values.yaml file can be created as required.

- `helm upgrade --install --namespace designer designer -f designer-values.yaml designer-100.0.112+xxxx.tgz --set designer.image.tag=9.0.1xx.xx.xx`

The values.yaml overrides passed as an argument to the above Helm upgrade command:

`designer.image.tag=9.0.1xx.xx.xx` - This is the new Designer version to be installed, for example, `9.0.111.05.5`.

If you are using the `--set` flag in the helm install to populate the `designer.designerConfig.envs` values, use `--set-string`, for example:

`--set-string designer.designerConfig.envs.DES_ES_PORT="8080"`.

DAS

To perform the initial deployment for a rolling upgrade in DAS, use the Helm command given below. The values.yaml file can be created as required.

- `helm upgrade --install --namespace designer designer-das -f designer-das-values.yaml designer-das-100.0.112+xxxx.tgz --set das.image.tag=9.0.1xx.xx.xx`

The values.yaml overrides passed as an argument to the above Helm upgrade command:

`das.image.tag=9.0.1xx.xx.xx` - This is the new DAS version to be installed, for example, `9.0.111.05.5`.

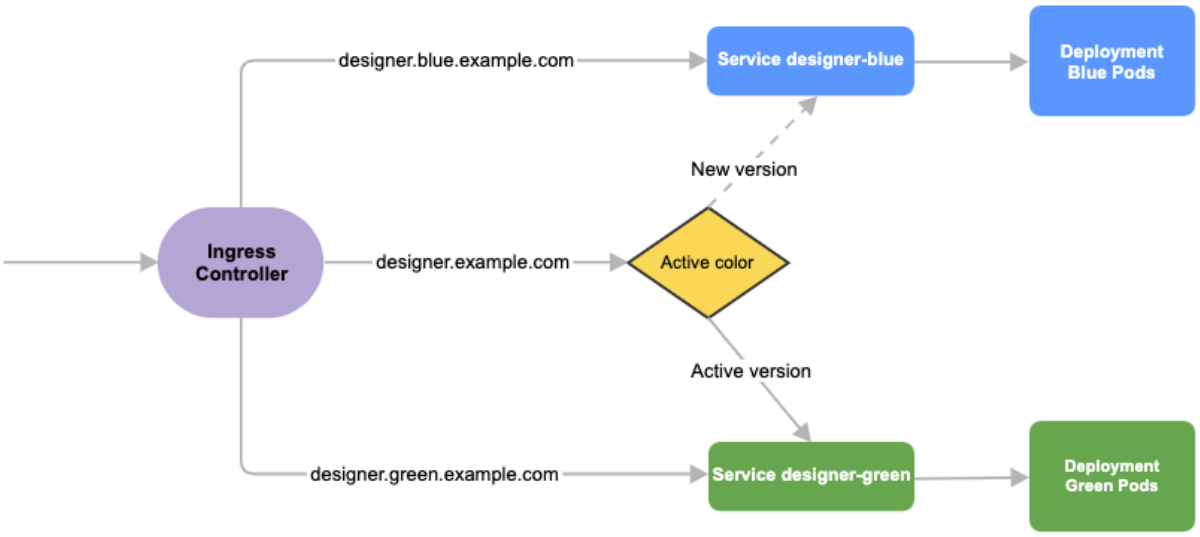
If you are using the `--set` flag in the helm install to populate the `das.dasConfig.envs values,` values, use `--set-string`, for example:

`--set-string das.dasConfig.envs.DAS_SERVICES_ELASTICSEARCH_PORT="9200"`.

Blue-Green deployment

Blue-Green deployment is a release management technique that reduces risk and minimizes downtime. It uses two production environments, known as Blue and Green or active and inactive, to provide reliable testing, continuous no-outage upgrades, and instant rollbacks. When a new release needs to be rolled out, an identical deployment of the application will be created using the Helm package and after testing is completed, the traffic is moved to the newly created deployment which becomes the active environment, and the old environment becomes inactive. This ensures that a fast rollback is possible by just changing route if a new issue is found with live traffic. The old inactive deployment is removed once the new active deployment becomes stable.

Service cutover is done by updating the Ingress rules. The diagram below shows the high-level approach to how traffic can be routed to Blue and Green deployments with Ingress rules.



Designer

Before you deploy Designer using the blue-green deployment strategy, complete the following preparatory steps:

1. Create 3 hostnames as given below. The blue service hostname must contain the string *blue*. For example, *designer.blue.example.com* or *designer-blue.example.com*. The green service hostname must contain the string *green*. For example, *designer.green.example.com* or *designer-green.example.com*. The blue/green services can be accessed separately with the blue/green hostnames:
   - `designer.example.com` - For the production host URL, this is used for external access.
   - `designer.blue.example.com` - For the blue service testing.

- `designer.green.example.com` - For the green service testing.

2. Configure the hostnames in the **designer-values.yaml** file under `ingress`. Annotations and paths can be modified as required.

```
ingress:
  enabled: true
  annotations: {}
  paths: [/]
  hosts:
    - designer.example.com
    - designer.blue.example.com
    - designer.green.example.com
```

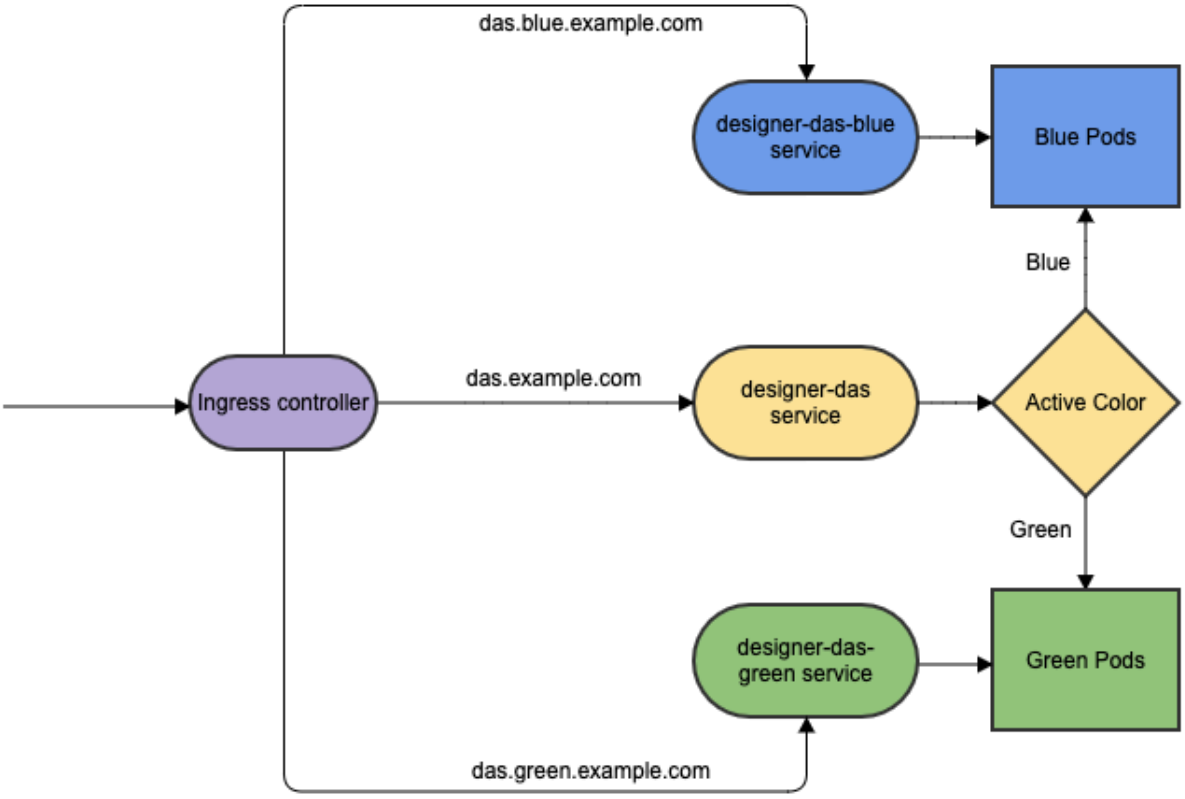
Deployment

The resources - ingress and persistent volume claims (PVC) - must be created initially before deploying the Designer service as these resources are shared between blue/green services and they are required to be created at the very beginning of the deployment. These resources are not required for subsequent upgrades. The required values are passed using the `--set` flag in the following steps. Values can also be directly changed in the values.yaml file.

1. Create Persistent Volume Claims required for the Designer service (assuming the volume service name is `designer-volume`).
   `helm upgrade --install --namespace designer designer-volume -f designer-values.yaml designer-9.0.xx.tgz --set designer.deployment.strategy=blue-green-volume`
   The values.yaml overrides passed as an argument to the above Helm upgrade command:
   `designer.deployment.strategy=blue-green-volume` - This denotes that the Helm install will create a persistent volume claim in the blue/green strategy.
2. Create Ingress rules for the Designer service (assuming the ingress service name will be designer-ingress):
   `helm upgrade --install --namespace designer designer-ingress -f designer-values.yaml designer-100.0.112+xxxx.tgz --set designer.deployment.strategy=blue-green-ingress --set designer.deployment.color=green`
   The values.yaml overrides passed as an argument to the above Helm upgrade command:
   `designer.deployment.strategy=blue-green-ingress` - This denotes that the Helm install will create ingress rules for the Designer service.
   `designer.deployment.color=green` - This denotes that the current production (active) color is green.
3. Deploy the Designer service color selected in step 2. In this case, green is selected and assuming the service name is `designer-green`:
   `helm upgrade --install --namespace designer designer-green -f designer-values.yaml designer-100.0.112+xxxx.tgz --set designer.deployment.strategy=blue-green --set designer.image.tag=9.0.1xx.xx.xx --set designer.deployment.color=green`

DAS

As with Designer, the Blue-Green strategy can be adopted for DAS as well. The Blue-Green architecture used for DAS is given below. Here, the cutover mechanism is controlled by Service, the Kubernetes manifest responsible for exposing the pods. The Ingress, when enabled, will point to the appropriate service based on the URL.



Deployment

The Ingress must be created initially before deploying the DAS service since it is shared between blue/green services and it is required to be created at the very beginning of the deployment. The Ingress is not required for subsequent upgrades. The required values are passed using the `-- set` flag in the following steps. Values can also be directly changed in the values.yaml file.

1. Deploy initial DAS pods and other resources by choosing an active color, in this example, green. Use the below command to create a `designer-das-green` service:
   `helm upgrade --install --namespace designer designer-das-green -f designer-das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.deployment.strategy=blue-green --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=green`
   The values.yaml overrides passed as an argument to the above Helm upgrade command:
   `das.deployment.strategy=blue-green` - This denotes that the DAS service will be installed using the blue-green deployment strategy.
   `das.image.tag=9.0.1xx.xx.xx` - This denotes the DAS version to be installed, for example, `9.0.111.04.4`.
   `das.deployment.color=green` - This denotes that the green color service is installed.
2. Once the initial deployment is done, the pods have to be exposed to the `designer-das` service. Execute the following command to create the `designer-das` service:
   `helm upgrade --install --namespace designer designer-das designer-das-100.0.106+xxx.tgz -f designer-das-values.yaml --set das.deployment.strategy=blue-green-service --set das.deployment.color=green`
   The values.yaml overrides passed as an argument to the above helm upgrade
   `das.deployment.strategy=blue-green-service` - This denotes that the designer-das service will be

installed and exposed to the active color pods.
`das.deployment.color=green` - This denotes that the designer-das service will point to green pods.

NodePort Service

The `designer-das-green` release creates a service called `designer-das-green` and the `designer-das-blue` release creates a service called `designer-das-blue`. If you are using NodePort services, ensure that the value of `designer.service.nodePort` is not the same for both the releases. In other words, you should assign dedicated node ports for the releases. The default value for `designer.service.nodePort` is **30280**. If this was applied to `designer-das-green`, use a different value for `designer-das-blue`, for example, **30281**. Use the below helm command to achieve this:
`helm upgrade --install --namespace designer designer-das designer-das-100.0.106+xxx.tgz -f designer-das-values.yaml --set das.deployment.strategy=blue-green-service --set das.deployment.color=green --set das.service.nodePort=30281`

Canary

Canary is optional and is only used along with Blue-Green. It is recommended in production. Canary pods are generally used to test new versions of images with live traffic. When you are installing the Designer and DAS services for the first time, you will not use Canary pods. Only when upgrading the services after initial deployment, you will use Canary pods for testing the new versions.

Deployment

1. Identify the current production color by checking the `designer-das` service selector labels (`kubectl describe service designer-das`). Green is the production color in the below example as the selector label is `color=green`.

   kubectl describe service designer-das

   ```
   Selector:                 color=green
   ```

2. To deploy canary pods, the `das.deployment.strategy` value must be set to `canary` in the **designer-das-values.yaml** file or using the `--set` flag as shown in the command below:
   `helm upgrade --install --namespace designer designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=green`
   The values.yaml overrides passed as an argument to the above Helm upgrade command:
   `das.deployment.strategy=canary` - This denotes that the Helm install will create canary pods.
   `das.deployment.color=green` - This denotes that the current production (active) color is green.

   > **Important**
   > To make sure Canary pods receive live traffic, they have to be exposed to the `designer-das` service by setting `das.deployment.color=`, which is obtained from step 1.

3. Once canary pods are up and running, ensure that the designer-das service points to the canary pods

using the `kubectl describe svc designer-das` command.

```
Endpoints: 10.206.0.101:8081,10.206.0.162:8081,10.206.0.90:8081
```

The IP address present in the Endpoints must match the IP address of the canary pod. The canary pod's IP address is obtained using the `kubectl describe pod` command.

```
IP: 10.206.0.90
IPs:
  IP: 10.206.0.90
```

Cleaning up

After completing canary testing, the canary pods must be cleaned up.

The `das.deployment.replicaCount` must be made zero and the release is upgraded. It can be changed in the **designer-das-values.yaml** file or through the `--set` flag as follows:

- `helm upgrade --install --namespace designer designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=blue --set das.deployment.replicaCount=0`

Validations and checks

Here are some common validations and checks that can be performed to know if the deployment was successful.

- Check if the application pods are in running state by using the `kubectl get pods` command.
- Try to connect to the Designer or DAS URL as per the ingress rules from your browser. You must be able to access the Designer and DAS webpages.

Post deployment procedures

Updating the flowsettings file

Post deployment, the **flowsettings.json** file can be modified through a Helm install as follows:

1. Extract the Designer Helm Chart and find the **flowsettings.yaml** file under the *Designer Chart* > *Config* folder.
2. Modify the necessary settings (refer to the *Post deployment configuration settings reference table* for

the different settings and their allowed values).

3. Execute the below Helm upgrade command on the non-production color service. It can be done as part of the Designer upgrade by passing the **flowsettings.yaml** file using the `--values` flag. In this case, a new Designer version can be used for the upgrade. If it is only a **flowsettings.json** update, the same Designer version is used.
```
helm upgrade --install --namespace designer designer-blue -f designer-values.yaml -f flowsettings.yaml designer-9.0.xx.tgz --set designer.deployment.strategy=blue-green --set designer.image.tag=9.0.1xx.xx.xx --set designer.deployment.color=blue
```
4. Once testing is completed on the non-production service, perform the cutover step as mentioned in the Cutover section (Designer Blue-Green deployment). After cutover, the production service will contain the updated settings. The non-active color Designer must also be updated with the updated settings after the cutover.

Enable optional features

Contents

- 1 Enable Designer Analytics and Audit Trail
  - 1.1 Designer
  - 1.2 DAS
- 2 Enable Personas
  - 2.1 Deploy personas.json
  - 2.2 Update Designer flowsettings.json
  - 2.3 Update application settings
  - 2.4 Adding voice definitions

Learn how to enable optional features in Designer post deployment.

Related documentation:

-
-
-

RSS:

- For private edition

Enable Designer Analytics and Audit Trail

Post Designer deployment, features such as Analytics and Audit Trail can be enabled by performing the below steps.

> **Important**
> Ensure Elasticsearch is deployed before proceeding.

Designer

1. Configure the following settings in flowsettings override (**flowsettings.yaml**) - Refer to the table in the Post deployment Designer configuration settings section for option descriptions.
   - enableAnalytics: true
   - enableESAuditLogs: true
   - esServer
   - esPort
   - esUrl
2. Configure the below setting in the DesignerEnv transaction list:
   `ReportingURL` in the **reporting** section.
3. Perform the steps in the *Updating the flowsettings file* section in Post deployment procedures.

DAS

1. Configure the following settings in the helm **das-values.yaml** file. For setting descriptions, refer to the

DAS deployment settings section in Deployment configuration settings.
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_ENABLED = true`
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_HOST`
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_PORT`

2. Execute the steps in the Upgrade section in the DAS deployment process for the Blue-Green strategy. The same DAS version running in production can be used for the upgrade.
3. Execute the steps in the Cutover section in the DAS deployment process for the Blue-Green strategy.

Enable Personas

You can enable the Personas feature in Designer by following the below steps.

Deploy personas.json

- Deploy the **personas.json** file in the workspace location, `/workspace/{tenantID}/workspace/personas/personas.json.`
- Create the **personas** directory if it does not exist.

Given below is a sample **personas.json** file:

```
[
    {
        "id": "1",
        "name": "Samantha",
        "gender": "female",
        "tags": ["female", "middle-age", "default"],
        "displayPersona": "female, 30-40s",
        "voice": [{
                "name": "samantha",
                "language": "en-US",
                "ttsname": "Samantha",
                "ttsengine": "NuanceTTS",
                "displayName": "Samantha"
            }, {
                "name": "karen",
                "language": "en-AU",
                "ttsname": "Karen",
                "ttsengine": "NuanceTTS",
                "displayName": "Karen"
            }, {
                "name": "amelie",
                "language": "fr-CA",
                "ttsname": "Amelie",
                "ttsengine": "NuanceTTS",
                "displayName": "Amelie"
            }, {
                "name": "paulina",
                "language": "es-MX",
                "ttsname": "Paulina",
                "ttsengine": "NuanceTTS",
                "displayName": "Paulina"
            }
        ],
        "digital": {},
```

```
        "email": {},
        "chat": {},
        "web": {}
    },
    {
        "id": "2",
        "name": "Tom",
        "gender": "male",
        "tags": ["male", "middle-age"],
        "displayPersona": "male, 30-40s",
        "voice": [{
                "name": "tom",
                "language": "en-US",
                "ttsname": "Tom",
                "ttsengine": "NuanceTTS",
                "displayName": "Tom"
            }, {
                "name": "lee",
                "language": "en-AU",
                "ttsname": "Lee",
                "ttsengine": "NuanceTTS",
                "displayName": "Lee"
            }, {
                "name": "felix",
                "language": "fr-CA",
                "ttsname": "Felix",
                "ttsengine": "NuanceTTS",
                "displayName": "Felix"
            }, {
                "name": "javier",
                "language": "es-MX",
                "ttsname": "Javier",
                "ttsengine": "NuanceTTS",
                "displayName": "Javier"
            }
        ],
        "digital": {},
        "email": {},
        "chat": {},
        "web": {}
    },
    {
        "id": "3",
        "name": "Gabriela",
        "gender": "female",
        "tags": ["female", "young"],
        "displayPersona": "female, 20-30s",
        "voice": [{
                "name": "gabriela",
                "language": "en-US",
                "ttsname": "en-US-Standard-E",
                "ttsengine": "GTTS",
                "displayName": "Gabriela"
            }, {
                "name": "sheila",
                "language": "en-AU",
                "ttsname": "en-AU-Standard-A",
                "ttsengine": "GTTS",
                "displayName": "Sheila"
            }, {
                "name": "lili",
                "language": "fr-CA",
                "ttsname": "fr-CA-Standard-A",
```

```
                "ttsengine": "GTTS",
                "displayName": "Lili"
            }
        ],
        "digital": {},
        "email": {},
        "chat": {},
        "web": {}
    },
    {
        "id": "4",
        "name": "Michael",
        "gender": "male",
        "tags": ["male", "young"],
        "displayPersona": "male, 20-30s",
        "voice": [{
                "name": "michael",
                "language": "en-US",
                "ttsname": "en-US-Standard-B",
                "ttsengine": "GTTS",
                "displayName": "Michael"
            }, {
                "name": "royce",
                "language": "en-AU",
                "ttsname": "en-AU-Standard-B",
                "ttsengine": "GTTS",
                "displayName": "Royce"
            }, {
                "name": "alexandre",
                "language": "fr-CA",
                "ttsname": "fr-CA-Standard-B",
                "ttsengine": "GTTS",
                "displayName": "Alexandre"
            }
        ],
        "digital": {},
        "email": {},
        "chat": {},
        "web": {}
    },
    {
        "id": "5",
        "name": "Diane",
        "gender": "female",
        "tags": ["female", "mature"],
        "displayPersona": "female, 40-50s",
        "voice": [{
                "name": "diane",
                "language": "en-US",
                "ttsname": "en-US-Standard-C",
                "ttsengine": "GTTS",
                "displayName": "Diane"
            }, {
                "name": "muriel",
                "language": "en-AU",
                "ttsname": "en-AU-Standard-C",
                "ttsengine": "GTTS",
                "displayName": "Muriel"
            }, {
                "name": "chloe",
                "language": "fr-CA",
                "ttsname": "fr-CA-Standard-C",
                "ttsengine": "GTTS",
```

```
                "displayName": "Chloe"
            }
        ],
        "digital": {},
        "email": {},
        "chat": {},
        "web": {}
    },
    {
        "id": "6",
        "name": "David",
        "gender": "male",
        "tags": ["male", "mature"],
        "displayPersona": "male, 40-50s",
        "voice": [{
                "name": "david",
                "language": "en-US",
                "ttsname": "en-US-Standard-D",
                "ttsengine": "GTTS",
                "displayName": "David"
            }, {
                "name": "austin",
                "language": "en-AU",
                "ttsname": "en-AU-Standard-D",
                "ttsengine": "GTTS",
                "displayName": "Austin"
            }, {
                "name": "pierre",
                "language": "fr-CA",
                "ttsname": "fr-CA-Standard-D",
                "ttsengine": "GTTS",
                "displayName": "Pierre"
            }
        ],
        "digital": {},
        "email": {},
        "chat": {},
        "web": {}
    }
]
```


Update Designer flowsettings.json

1. Enable the persona `feature` flag in the **flowsettings.json** override file.

```
"features": {
  "persona": true
```

2. Perform the steps in the Updating the flowsettings file section for the changes to take effect.

Update application settings

Perform the following steps to enable the persona in the required Designer application:

1. Open the required Designer application and navigate to the **Settings** tab.
2. In the **Application Settings**, select the **Enable Persona** checkbox in the **Persona** tab.
3. If you are using a Google TTS custom voice, select **Enable Custom Voices**.

4. Re-publish the application and create a new build.

Adding voice definitions

> **Important**
>
> Additional voice definitions can be added by Genesys. Contact your Genesys representative for more information.

Designer supports Nuance and Google (standard and custom) TTS voice definitions. This example of a voice definition contains both a standard and custom Google TTS voice:

```
"voice": [
            {                                                  // Example of a standard Google TTS
voice definition.
                "name": "fatima",
                "language": "ar-SA",
                "ttsname": "ar-XA-Wavenet-A",
                "ttsengine": "GTTS",
                "displayName": "Fatima"
            },
            {                                                 // Example of a Custom Google TTS voice
definition.
                "name": "ursula",
                "language": "de-DE",
                "ttsname": "de-DE-Wavenet-A",
                "ttsengine": "GTTS",
                "displayName": "Ursula",
                "ttsCustomVoice" : true,
                "ttsCustomVoiceURI: : ""
            }
]
```

Voice definitions must include the following details:

Name	Value description
`name`	Name of this voice.
`language`	Language that matches the **Language** system variable.
`ttsname`	Voice name used for this Language.
`ttsengine`	Specifies the TTS service provider for this voice. Designer supports the following TTS engines: • Enter `NuanceTTS` for Nuance voices. • Enter `GTTS` for Google voices.
`displayName`	Name of this voice as displayed in the Designer UI.
Note: The following values are required only if you are defining a *custom* Google TTS voice. Otherwise, they do not need to be included in the voice definition.	

Name	Value description
`ttsCustomVoice`	Enter `true` for this setting. This tells Designer that the voice is custom (that is, unique to your environment) and to ignore the `ttsname` value.
`ttsCustomVoiceURI`	Specifies the location of the custom voice.

> **Important**
>
> To use custom Google TTS voices in an application, **Enable Custom Voices** must be selected on the Persona tab in the application settings.

Upgrade, roll back, or uninstall Designer

Contents

- 1 Supported upgrade strategies
- 2 Timing
  - 2.1 Scheduling considerations
- 3 Monitoring
- 4 Preparatory steps
- 5 Rolling Update
  - 5.1 Rolling Update: Upgrade
  - 5.2 Rolling Update: Verify the upgrade
  - 5.3 Rolling Update: Rollback
  - 5.4 Rolling Update: Verify the rollback
- 6 Blue/Green
  - 6.1 Blue/Green: Upgrade Designer
  - 6.2 Blue/Green: Rollback Designer
  - 6.3 Blue/Green: Upgrade DAS
  - 6.4 Blue/Green: Rollback DAS
- 7 Canary
  - 7.1 Cleaning up
- 8 Post-upgrade procedures
  - 8.1 Upgrading the Designer workspace
  - 8.2 Elasticsearch maintenance recommendations
- 9 Uninstall

Learn how to upgrade, roll back, or uninstall Designer.

Related documentation:

-
-
-

RSS:

- For private edition

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

Supported upgrade strategies

Designer supports the following upgrade strategies:

Service	Upgrade Strategy	Notes
Designer	• Rolling Update • Blue/Green • Canary	Canary is used in combination with Blue/Green.
Designer Application Server	• Rolling Update • Blue/Green • Canary	Canary is used in combination with Blue/Green.

The upgrade or rollback process to follow depends on how you deployed the service initially. Based on the deployment strategy adopted during initial deployment, refer to the corresponding upgrade or rollback section on this page for related instructions.

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

Timing

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

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

Scheduling considerations

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

Monitoring

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

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

Preparatory steps

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

Each time you upgrade a service:

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

Rolling Update

Rolling Update: Upgrade

Execute the following command to upgrade :

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

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

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

Follow the same instructions to upgrade Designer and DAS. For example, the respective commands are:

- Designer:

  ```
  helm upgrade --install --namespace designer designer -f designer-values.yaml
   designer-100.0.112+1401.tgz  --set designer.image.tag=100.0.112.11
  ```

- DAS:

  ```
  helm upgrade --install --namespace designer designer-das -f designer-das-values.yaml
   designer-das-100.0.112+1401.tgz  --set das.image.tag=9.0.111.05.5
  ```

If you are using the --set flag in the helm upgrade command to populate the `designer.designerConfig.envs` or `das.dasConfig.envs` values, use `--set-string`, for example:

- Designer: `--set-string designer.designerConfig.envs.DES_ES_PORT="8080"`
- DAS: `--set-string das.dasConfig.envs.DAS_SERVICES_ELASTICSEARCH_PORT="9200"`

Rolling Update: Verify the upgrade

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

Rolling Update: Rollback

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

```
helm rollback
```

or, to roll back to an even earlier version:

```
helm rollback
```

Alternatively, you can re-install the previous package:

1. Revert the image version in the `.image.tag` parameter in the **-values.yaml** file. If applicable, also

revert any configuration changes you implemented for the new release.

2. Execute the following command to roll back the upgrade:

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

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

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

Follow the same instructions to roll back Designer and DAS. For example, the respective commands are:

- Designer:

  ```
  helm upgrade --install --namespace designer designer -f designer-values.yaml
  designer-100.0.112+1401.tgz  --set designer.image.tag=100.0.112.11
  ```

- DAS:

  ```
  helm upgrade --install --namespace designer designer-das -f designer-das-values.yaml
  designer-das-100.0.112+1401.tgz  --set das.image.tag=9.0.111.05.5
  ```

Rolling Update: Verify the rollback

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

- Ensure that the image version in the -values.yaml file reflects the version that you rolled back to.

Blue/Green

Blue/Green: Upgrade Designer

1. Identify the current production color by checking the Designer ingress rules:
   `kubectl describe ingress designer-ingress`

   Green is the production color in the below example as the production host name points to the green service.

   kubectl describe ingress designer-ingress

   ```
   Host                        Path  Backends
   ----                        ----  --------
   designer.example.com        /   designer-green:http (10.244.0.23:8888)
   designer.green.example.com  /   designer-green:http (10.244.0.23:8888)
   designer.blue.example.com   /   designer-blue:http (10.244.0.45:8888)
   ```

2. Deploy the Designer service on to the non-production color (in this example, blue is the non-production color and assuming the service name is designer-blue):

```
helm upgrade --install --namespace designer designer-blue -f designer-values.yaml
designer-100.0.112+1401.tgz --set designer.deployment.strategy=blue-green --set
designer.image.tag=100.0.111.05.5 --set designer.deployment.color=blue
```

Use the non-production host name to access the non-production color. For example, `designer.blue.example.com`). You can use this URL for testing.

NodePort Service

The `designer-green` release creates a service called `designer-green` and the `designer-blue` release creates a service called `designer-blue`. If you are using NodePort services, ensure that the value of `designer.service.nodePort` is not the same for both the releases. In other words, you should assign dedicated node ports for the releases. The default value for `designer.service.nodePort` is **30180**. If this was applied to `designer-green`, use a different value for `designer-blue`, for example, **30181**. Use the below helm command to achieve this:

```
helm upgrade --install --namespace designer designer-blue -f designer-values.yaml
designer-100.0.112+1401.tgz --set designer.deployment.strategy=blue-green --set
designer.image.tag=100.0.111.05.5 --set designer.deployment.color=blue --set
designer.service.nodePort=30181
```

Cutover

Once testing is completed on the non-production color, move traffic to the new version by updating the Ingress rules:

- Update the Designer Ingress with the new deployment color by running the following command (in this case, blue is the new deployment color, that is, the non-production color):

```
helm upgrade --install --namespace designer designer-ingress -f designer-values.yaml
designer-100.0.112+1401.tgz --set designer.deployment.strategy=blue-green-ingress --
set designer.deployment.color=blue
```

Verify the upgrade

- Verify the ingress rules by running the following command:
  `kubectl describe ingress designer-ingress`
  The production host name must point to the new color service, that is, `blue`.

Blue/Green: Rollback Designer

To roll back the upgrade, modify the ingress rules to point back to the old deployment pods (green, in this example) by performing a cutover again.

- Perform a cutover using the following command:

```
helm upgrade --install --namespace designer designer-ingress -f designer-values.yaml
designer-100.0.112+1401.tgz --set designer.deployment.strategy=blue-green-ingress --
set designer.deployment.color=green
```

Verify the rollback

- Verify the rollback in the same way that you verified the upgrade (see Blue-Green: Verify the upgrade). The type label must have the active color's label, that is, `color=green`.

Blue/Green: Upgrade DAS

1. Identify the current production color by checking the `designer-das` service selector labels:

   ```
   kubectl describe service designer-das
   ```

   Green is the production color in the below example as the selector label is `color=green`.

   kubectl describe service designer-das

   ```
   Selector:                 color=green
   ```

2. Deploy the DAS service on to the non-production color (in this example, blue is the non-production color and assuming the service name is `designer-das-blue`):

   ```
   helm upgrade --install --namespace designer designer-das-blue -f das-values.yaml
   designer-das-100.0.106+1401.tgz --set das.deployment.strategy=blue-green --set
   das.image.tag=9.0.111.05.5 --set das.deployment.color=blue
   ```

Use the non-production service name to access the non-production color.

NodePort Service

The `designer-das-green` release creates a service called `designer-das-green` and the `designer-das-blue` release creates a service called `designer-das-blue`. If you are using NodePort services, ensure that the value of `designer.service.nodePort` is not the same for both the releases. In other words, you should assign dedicated node ports for the releases. The default value for `designer.service.nodePort` is **30280**. If this was applied to `designer-das-green`, use a different value for `designer-das-blue`, for example, **30281**. Use the below helm command to achieve this:

```
helm upgrade --install --namespace designer designer-das designer-das-100.0.106+xxx.tgz -f
designer-das-values.yaml --set das.deployment.strategy=blue-green-service --set
das.deployment.color=green --set das.service.nodePort=30281
```

Cutover

Once testing is completed on the non-production color, move traffic to the new version by updating the `designer-das` service.

- Update the `designer-das` service with the new deployment color (in this example, blue is the new deployment color, that is, non-production color)

  ```
  helm upgrade --install --namespace designer designer-das-service -f designer-das-
  values.yaml designer-das-100.0.106+1401.tgz --set das.deployment.strategy=blue-green-
  service --set das.deployment.color=blue
  ```

Verify the upgrade

- Verify the service by executing the `kubectl describe service designer-das` command.
  The type label must have the active color's label, that is, `color=blue`.

Blue/Green: Rollback DAS

To roll back the upgrade, perform a cutover again to point the service back to the old deployment (green).

- Perform a cutover using the following command:

```
helm upgrade --install --namespace designer designer-das-service -f designer-das-
values.yaml designer-das-100.0.106+1401.tgz --set das.deployment.strategy=blue-green-
service --set das.deployment.color=green
```

Verify the rollback

- Verify the rollback in the same way that you verified the upgrade (see Blue-Green: Verify the upgrade).
  The type label must have the active color's label, `color=green`.

Canary

Canary is optional and is only used along with Blue-Green. It is recommended in production. Canary pods are generally used to test new versions of images with live traffic. You will not use Canary pods when you are installing the Designer and DAS services for the first time. You will only use Canary pods for testing the new versions when upgrading the services after initial deployment.

1. Identify the current production color by checking the `designer-das` service selector labels (`kubectl describe service designer-das`). Green is the production color in the below example as the selector label is `color=green`.

   kubectl describe service designer-das

   ```
   Selector:                 color=green
   ```

2. To deploy canary pods, the `das.deployment.strategy` value must be set to `canary` in the **designer-das-values.yaml** file or using the `-- set` flag as shown in the command below:
   `helm upgrade --install --namespace designer designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=green`
   The values.yaml overrides passed as an argument to the above Helm upgrade command:
   `das.deployment.strategy=canary` - This denotes that the Helm install will create canary pods.
   `das.deployment.color=green` - This denotes that the current production (active) color is green.

> **Important**
>
> To make sure Canary pods receive live traffic, they have to be exposed to the designer-das service by setting das.deployment.color=, which is obtained from step 1.

3. Once canary pods are up and running, ensure that the designer-das service points to the canary pods using the `kubectl describe svc designer-das` command.

```
Endpoints: 10.206.0.101:8081,10.206.0.162:8081,10.206.0.90:8081
```

The IP address present in the Endpoints must match the IP address of the canary pod. The canary pod's IP address is obtained using the `kubectl describe pod` command.

```
IP: 10.206.0.90
IPs:
  IP: 10.206.0.90
```

Cleaning up

After completing canary testing, the canary pods must be cleaned up. The `das.deployment.replicaCount` must be made zero and the release is upgraded. It can be changed in the **designer-das-values.yaml** file or through the `--set` flag as follows:

- `helm upgrade --install --namespace designer designer-das-canary -f das-values.yaml designer-das-100.0.106+xxxx.tgz --set das.deployment.strategy=canary --set das.image.tag=9.0.1xx.xx.xx --set das.deployment.color=blue --set das.deployment.replicaCount=0`

Post-upgrade procedures

Upgrading the Designer workspace

Workspace resources must be upgraded after cutover. Perform the following steps to upgrade the system resources in the Designer workspace:

1. Log in to one of the Designer pods using the `kubectl exec -it` bash command.
2. Execute the following migration command (this creates new directories/new files introduced in the new version):
   `node ./bin/cli.js workspace-upgrade -m -t`
3. Execute the workspace resource upgrade command (this upgrades system resources, such as system service PHP files, internal audio files and callback resources):
   `node ./bin/cli.js workspace-upgrade -t`
   In the above command, `contact_center_id` , is the Contact Center ID created in GWS for this tenant (workspace resources are located under the Contact Center ID folder (*/workspaces//workspace*)).

Elasticsearch maintenance recommendations

To help you better manage your indexes and snapshots, and to prevent too many indexes from creating an overflow of shards, Genesys recommends that you set up a scheduled execution of Elasticsearch Curator with the following two actions:

- Delete indexes older than the given threshold according to the index name and mask.
  - `sdr-*` (3 months)
  - `audit-*` (12 months)
- Make a snapshot of each index:
  - `sdr-*` (yesterday and older)
  - `audit-*`
  - `kibana-int-*`

Uninstall

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

Execute the following command to uninstall :

```
helm uninstall  -n
```

Observability in Designer

Contents

- 1 Monitoring
  - 1.1 Enable monitoring
  - 1.2 Configure metrics
  - 1.3 What do Designer metrics monitor?
- 2 Alerting
  - 2.1 Configure alerts
- 3 Logging

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

Related documentation:

-
-
-

RSS:

- For private edition

Monitoring

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

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

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

- Designer Application Server metrics
- Designer metrics

See also System metrics.

Desginer and DAS generate application related metrics at the `/metric` API in the standard Prometheus client format.

> **Important**
> In addition to the metrics listed in the DES and DAS metrics topics, you can also obtain infrastructure related metrics by installing standard Prometheus clients in the Kubernetes cluster.

Enable monitoring

To enable monitoring you must configure the various Prometheus related options. For more details on these various options, refer to the *Designer deployment settings* and *DAS deployment settings* sections in the Configure Designer topic.

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
Designer Application Server	ServiceMonitor	8081	See selector details on the Designer Application Server metrics and alerts page	10 seconds
Designer	ServiceMonitor	8888	See selector details on the Designer metrics and alerts page	10 seconds

Configure metrics

The metrics that are exposed by the DES and DAS 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 DES and DAS services expose. You can also query Prometheus directly or via a dashboard to see all the metrics available from the DES and DAS services.

What do Designer metrics monitor?

The exposed DES and DAS metrics help you monitor a number of data points that are important in a production environment. For more details on the individual metrics, refer to the Metrics pages.

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 Designer alerts, see:

- Designer Application Server alerts
- Designer alerts

Configure alerts

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

Enable alerts in Designer

To enable alerts in Designer, use either of the following methods:

Method 1: Enable Prometheus alerts in the values.yaml file.

```
designer:
    prometheus:
        alerts:
            enabled: true # this will be false by default.
```

Method 2: Find out the active deployment color and execute the below command in the corresponding deployment:

```
helm upgrade --install designer-blue -f designer-values.yaml designer-9.0.xx.tgz --
set designer.deployment.strategy=blue-green --set
designer.prometheus.alerts.enabled=true
```

Disable alerts in Designer

To disable or delete alerts, use either of the following methods:

Method 1: Disable Prometheus alerts in the values.yaml file.

```
designer:
    prometheus:
        alerts:
            enabled: false # this will be false default.
```

Method 2: Pass the below parameter along with the Helm upgrade command.

```
helm upgrade --install designer-blue -f designer-values.yaml designer-9.0.xx.tgz --
set designer.deployment.strategy=blue-green --set
designer.prometheus.alerts.enabled=false
```

Enable alerts in DAS

To enable alerts, use either of the following methods:

Method 1: Enable Prometheus alerts in the values.yaml file.

```
das:
    prometheus:
```

```
        alerts:
            enabled: true # this will be false default.
```

Method 2: Pass the below parameter along with the Helm upgrade command.

```
helm upgrade --install designer-das-blue -f designer-values.yaml designer-
das-9.0.xx.tgz --set das.deployment.strategy=blue-green --set
das.prometheus.alerts.enabled=true
```

Disable alerts in DAS

To disable or delete alerts, use either of the following methods:

Method 1: Disable Prometheus alerts in the values.yaml file.

```
das:
    prometheus:
        alerts:
            enabled: false # this will be false default.
```

Method 2: Pass the below parameter along with the Helm upgrade command.

```
helm upgrade --install designer-das-blue -f designer-values.yaml designer-
das-9.0.xx.tgz --set das.deployment.strategy=blue-green --set
das.prometheus.alerts.enabled=false
```

Update alert parameters

The following alert parameters can be updated:

- Alert Threshold (ALERT_PARAMETER_NAME: threshold)
- Alert Interval (ALERT_PARAMETER_NAME: interval)
- Alert Severity (ALERT_PARAMETER_NAME: AlertPriority)

Perform the following steps to update the above alerts:

1. Refer to the list of alerts and identify the name of the alert you want to update or modify.
2. Update the alert by adding a parameter in the below format in the values.yaml file:

   ```
   designer:
        prometheus:
            alerts:
                :
                    :
                    :
   ```

For example, consider the CPU utilization alert. The alert name is CPUUtilization with a default threshold of 75, severity set to CRITICAL and interval set to 180s. To modify its threshold to 80, severity to HIGH, and interval to 120 seconds, you will have to make the following changes in the values.yaml file:

```
designer:
```

```
prometheus:
    alerts:
        CPUUtilization:
            threshold: 80
            interval: 120
            AlertPriority: HIGH
```

> **Important**
>
> Though the ability to create custom alerts and some dashboards are packaged with the Designer Helm charts, these are not documented and are not supported as of now.

Logging

Refer to the Logging topic for information on configuring logging for the DES and DAS services.

DES metrics and alerts

Contents

- 1 Metrics
- 2 Alerts

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

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

DES	ServiceMonitor	8888	`selector:`
			`  matchLabels:`
			`  {{- include "designer.labels" . | nindent 6 }}`
			
			Labels to identify which service to communicate with depend on the release name.
			
			Path: `/metrics`
				10 seconds

See details about:

- DES metrics
- DES alerts

Metrics

Given below are some of the metrics exposed by the DES service:

> **Important**
>
> Designer exposes many Genesys-defined as well as system metrics. You can query Prometheus directly to see all the available metrics. The metrics documented on this page are likely to be particularly useful. Genesys does not commit to maintain other currently available Designer metrics not documented on this page.

Metric and description	Metric details	Indicator of
des_csp_violations_total Number of CSP violations.	**Unit:** **Type:** Counter **Label:** **Sample value:** 0	

Alerts

The following alerts are defined for DES.

Alert	Severity	Description	Based on	Threshold
CPUUtilization (Alarm: Pod CPU Usage)	CRITICAL	Triggered when a pod's CPU utilization is beyond the threshold.		75% Default interval: 180s
MemoryUtilization (Alarm: Pod Memory Usage)	CRITICAL	Triggered when a pod's memory utilization is beyond the threshold.		75% Default interval: 180s
containerRestartAlert (Alarm: Pod Restarts Count)	CRITICAL	Triggered when a pod's restart count is beyond the threshold.		5 Default interval: 180s
containerReadyAlert (Alarm: Pod Ready Count)	CRITICAL	Triggered when a pod's ready count is less than the threshold (1).		1 Default interval: 60s
AbsentAlert (Alarm: Deployment availability)	CRITICAL	Triggered when Designer pod metrics are unavailable.		1 Default interval: 60s
WorkspaceUtilization (Alarm: Azure Fileshare PVC Usage)	HIGH	Triggered when file share usage is greater than the threshold.		80% Default interval: 180s
Health (Alarm: Health Status)	CRITICAL	Triggered when Designer health status is 0.		0 Default interval: 60s
WorkspaceHealth (Alarm: Workspace Health Status)	CRITICAL	Triggered when Designer is not able to communicate with the workspace.		0 Default interval: 60s
ESHealth (Alarm: Elasticsearch Health Status)	CRITICAL	Triggered when Designer/DAS is not able to reach the Elasticsearch		0 Default interval: 60s

Alert	Severity	Description	Based on	Threshold
		server.		
GWSHealth (Alarm: GWS Health Status)	CRITICAL	Triggered when Designer/DAS is not able to reach the GWS server.		0 Default interval: 60s

DAS metrics and alerts

Contents

- 1 Metrics
- 2 Alerts

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

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval
DAS	ServiceMonitor	8081	`selector: matchLabels: {{- include "das.serviceSelectorLabels" . | nindent 6 }}` Labels to identify which service to communicate with depend on an unique label applicable to DAS. **Path**: `/metrics`	10 seconds

See details about:

- DAS metrics
- DAS alerts

Metrics

Given below are some of the metrics exposed by the DAS service:

> **Important**
> DAS exposes many Genesys-defined as well as system metrics. You can query Prometheus directly to see all the available metrics. The metrics documented on this page are likely to be particularly useful. Genesys does not commit to maintain other currently available DAS metrics not documented on this page.

Metric and description	Metric details	Indicator of
sdr_requests_received Number of requests received since DAS is running (provided for each CCID).	**Unit:** **Type:** Counter **Label:** **Sample value:** 1998352	
sdr_requests_rejected	**Unit:**	

Metric and description	Metric details	Indicator of
Number requests rejected since DAS is running (provided for each CCID).	**Type:** Counter **Label:** **Sample value:**	
data_tables_requests_failures Number of failed data table requests since DAS is running (provided for each CCID).	**Unit:** **Type:** Counter **Label:** **Sample value:** 80	
data_tables_request_duration Data table requests latency in seconds, since DAS is running (provided for each CCID).	**Unit:** seconds **Type:** Histogram **Label:** **Sample value:** 189	
business_hours_requests_failures Number of failed business hours requests since DAS is running.	**Unit:** **Type:** Counter **Label:** **Sample value:**	
business_hours_request_duration Business hours requests latency in seconds, since DAS is running (provided for each CCID).	**Unit:** seconds **Type:** Histogram **Label:** **Sample value:** 26	
special_days_requests_failures Number of failed special days requests since DAS is running.	**Unit:** **Type:** Counter **Label:** **Sample value:**	
special_days_request_duration Special days requests latency in seconds, since DAS is running (provided for each CCID).	**Unit:** seconds **Type:** Histogram **Label:** **Sample value:** 34	
external_requests_failures Number of failed external requests since DAS is running.	**Unit:** **Type:** Counter **Label:** **Sample value:**	
external_requests_timedout Number of timed out external requests since DAS is running.	**Unit:** **Type:** Counter **Label:** **Sample value:**	
external_requests_duration External requests latency in seconds, since DAS is running.	**Unit:** seconds **Type:** Histogram **Label:** **Sample value:**	
das_http_request_duration_seconds	**Unit:** seconds	

Metric and description	Metric details	Indicator of
HTTP request latency in seconds (provided for each request type and CCID).	**Type:** Histogram **Label:** **Sample value:** 40	
das_http_requests_total Number of HTTP requests (provided for each request type and CCID).	**Unit:** **Type:** Counter **Label:** **Sample value:** 40	
nginx_metric_errors_total Number of nginx-lua-prometheus errors.	**Unit:** **Type:** Counter **Label:** **Sample value:** 2	

Alerts

The following alerts are defined for DAS.

Alert	Severity	Description	Based on	Threshold
CPUUtilization (Alarm: Pod CPU Usage)	CRITICAL	Triggered when a pod's CPU utilization is beyond the threshold.		75% Default interval: 180s
MemoryUtilization (Alarm: Pod Memory Usage)	CRITICAL	Triggered when a pod's memory utilization is beyond the threshold.		75% Default interval: 180s
containerRestartAlert (Alarm: Pod Restarts Count)	CRITICAL	Triggered when a pod's restart count is beyond the threshold.		5 Default interval: 180s
containerReadyAlert (Alarm: Pod Ready Count)	CRITICAL	Triggered when a pod's ready count is less than the threshold (1).		1 Default interval: 60s
AbsentAlert (Alarm: Deployment availability)	CRITICAL	Triggered when DAS pod metrics are unavailable.		1 Default interval: 60s
WorkspaceUtilization	HIGH	Triggered when file		80%

Alert	Severity	Description	Based on	Threshold
(Alarm: Azure Fileshare PVC Usage)		share usage is greater than the threshold.		Default interval: 180s
Health (Alarm: Health Status)	CRITICAL	Triggered when DAS health status is 0.		0 Default interval: 60s
WorkspaceHealth (Alarm: Workspace Health Status)	CRITICAL	Triggered when DAS is not able to communicate with the workspace.		0 Default interval: 60s
PHPHealth (Alarm: PHP Health Status)	CRITICAL	Triggered when Designer/DAS experiences a PHP Health check failure.		0 Default interval: 60s
ProxyHealth (Alarm: Proxy Health Status)	CRITICAL	Triggered when Designer/DAS experiences a Proxy Health check failure.		0 Default interval: 60s
HTTP5XXCount (Alarm: Application 5XX Error)	HIGH	Triggered when DAS exceeds the allowed 5xx error count threshold specified here.		10 Default interval: 180s
HTTP4XXCount (Alarm: Application 4XX Error)	HIGH	Triggered when DAS exceeds the 4xx error count threshold specified here.		100 Default interval: 180s
PhpLatency (Alarm: DAS PHP Latency Alert)	HIGH	Triggered when the average time taken by a PHP request is greater than the threshold (in seconds) specified here.		10s Default interval: 180s
HTTPLatency (Alarm: DAS HTTP Latency Alert)	HIGH	Triggered when the average time taken by a HTTP request is greater than the threshold (in seconds) specified here.		10s Default interval: 180s

Logging

Contents

- 1 Log levels
  - 1.1 Designer
  - 1.2 DAS

Learn how to configure log levels for Designer and DAS.

Related documentation:

-
-
-

RSS:

- For private edition

Designer and DAS support console output (stdout) logging. Genesys recommends configuring console output logging to minimize the host IOPs and PVCs consumption by using log volumes. Console output logs can be extracted using log collectors like *fluentbit/fluentd* and *Elasticsearch*.

Ensure the below setttings are configured in the respective **values.yaml** overrides for console logging:

1. Designer
   `designerEnv.envs.DES_FILE_LOGGING_ENABLED = false`
2. DAS
   `dasEnv.envs.DAS_FILE_LOGGING_ENABLED = falsedasEnv.envs.DAS_STDOUT_LOGGING_ENABLE = true`

Log levels

Post deployment, Designer and DAS log levels can be modified as follows:

Designer

1. Configure the `logging` setting in the flowsettings override (**flowsettings.yaml**) - Refer to the table in the Post deployment Designer configuration settings section for option descriptions.
2. Execute the steps in the *Updating the flowsettings file* section in Post deployment procedures for the changes to take effect .

DAS

1. Configure the `dasEnv.envs.DAS_LOG_LEVEL` setting in the Helm **das-values.yaml** file. For setting descriptions, refer to the *DAS deployment settings* section in Deployment configuration settings.
2. Execute the steps in the Upgrade section in the DAS deployment process for the Blue-Green strategy. The same DAS version running in production can be used for the upgrade,
3. Execute the steps in the Cutover section in the DAS deployment process for the Blue-Green strategy.

Designer on GKE

Contents

- 1 Configure a secret to access JFrog
- 2 Create a Designer secret with GWS
  - 2.1 GWS settings for auth
- 3 Checking logs

Learn more about specific settings that you have to configure when deploying Designer on Google Kubernetes Engine (GKE).

Related documentation:

-
-
-

RSS:

- For private edition

> **Important**
> Configure and deploy Designer as described in the topics under the **Configure and deploy** section. Only additional information that is specific to deploying Designer on GKE is provided here.

Configure a secret to access JFrog

If you haven't done so already, create a secret for accessing the JFrog registry (for example, `jfrog-stage-credentials`):

```
kubectl create secret docker-registry jfrog-stage-credentials \
--docker-server=pureengage-docker-staging.jfrog.io \
--docker-username= \
--docker-password= \
--docker-email=
```

Now map the secret to the default service account:

```
kubectl secrets link default jfrog-stage-credentials --for=pull
```

Create a Designer secret with GWS

To create a Designer secret with GWS, update the following values to your Environment in the GWS service:

- : Set to the internal API of GWS
- `contactCenterIds`: Set to your tenant ID

- `redirectURIs`: Set to the URL(s) to be used for Designer
- : Set to the domain address for the environment

And, optionally, update the following:

- `-u opsAdmin`: `opsPass` (this is the default delivered for tenants)
- `client_secret`: Set to any value (used in secret)
- `name`: Set to anything
- `client_id`: Set to your client ID (used in secret)

GWS settings for auth

In the Designer flowsettings override file, update the following options with these values:

- `htccserver`: `gws-service-proxy.gws.svc.cluster.local`
- `gwsenvurl`: `http://gauth-environment.gauth.svc.cluster.local:80`
- `gwsauthurl`: `http://gauth-auth.gauth.svc.cluster.local:8`
- `ssoLoginUrl`: `https://gauth.apps.`

Checking logs

After deploying Designer, you check the logs using the following commands:

Designer

```
kubectl get pods
```

```
kubectl logs
```

DAS

```
kubectl get pods
```

```
kubectl logs
```

Designer on AKS

Contents

- 1 Configure a secret to access JFrog
- 2 Create a Designer secret with GWS
  - 2.1 GWS settings for auth
- 3 Checking logs

Learn more about specific settings that you have to configure when deploying Designer on Azure Kubernetes Service (AKS).

Related documentation:

-
-
-

RSS:

- For private edition

> **Important**
> Configure and deploy Designer as described in the topics under the **Configure and deploy** section. Only additional information that is specific to deploying Designer on AKS is provided here.

Configure a secret to access JFrog

If you haven't done so already, create a secret for accessing the JFrog registry (for example, `jfrog-stage-credentials`):

```
kubectl create secret docker-registry jfrog-stage-credentials \
--docker-server=pureengage-docker-staging.jfrog.io \
--docker-username= \
--docker-password= \
--docker-email=
```

Now map the secret to the default service account:

```
kubectl secrets link default jfrog-stage-credentials --for=pull
```

Create a Designer secret with GWS

To create a Designer secret with GWS, update the following values to your Environment in the GWS service:

- : Set to the internal API of GWS
- `contactCenterIds`: Set to your tenant ID

- `redirectURIs`: Set to the URL(s) to be used for Designer
- : Set to the domain address for the environment

And, optionally, update the following:

- `-u opsAdmin`: `opsPass` (this is the default delivered for tenants)
- `client_secret`: Set to any value (used in secret)
- `name`: Set to anything
- `client_id`: Set to your client ID (used in secret)

GWS settings for auth

In the Designer flowsettings override file, update the following options with these values:

- `htccserver`: `gws-service-proxy.gws.svc.cluster.local`
- `gwsenvurl`: `http://gauth-environment.gauth.svc.cluster.local:80`
- `gwsauthurl`: `http://gauth-auth.gauth.svc.cluster.local:8`
- `ssoLoginUrl`: `https://gauth.apps.`

Checking logs

After deploying Designer, you check the logs using the following commands:

Designer

```
kubectl get pods
```

```
kubectl logs
```

DAS

```
kubectl get pods
```

```
kubectl logs
```