Designer Deployment Guide

Deploy Designer (versions v9010005 and above)

4/3/2026

Contents

Learn how to deploy Designer as a service in a Kubernetes cluster (for **DesDepMnfst v9010005** and above).

1. About this document

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.

Information on the following topics is provided:

- Overview of Designer and DAS
- Configuration details
- Deployment process
- Enabling optional features
- Cleanup
- Known limitations

1.1 Intended audience

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

1.2 Before you begin

1. A Kubernetes cluster must be deployed. Refer to the Kubernetes documentation site for installation instructions.
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 an on-premise deployment of Designer and DAS as a service in a K8s cluster.

2. Product overview

The following sections provide a brief overview of Designer and DAS.

2.1 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.

2.2 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.
- Interface to External APIs.

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

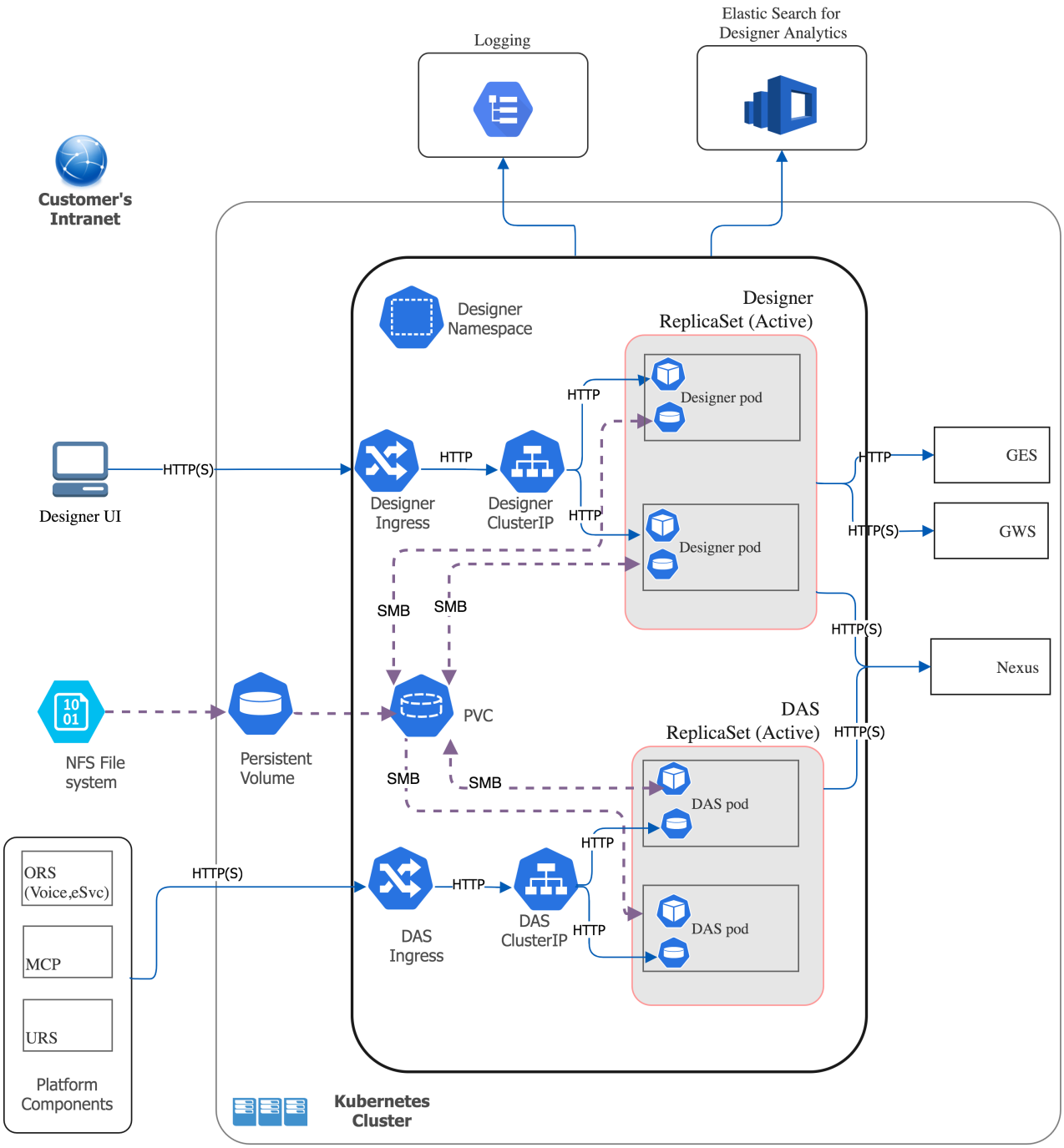
NGINX module.

- 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.

2.3 Deployment architecture

The below architecture diagram illustrates a sample premise deployment of Designer and DAS:



2.4 High Availability (HA) and Scalability

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 explains how to

configure HA and auto-scaling.

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

3. Prerequisites

Before deploying Designer, ensure the following resources are deployed, configured, and accessible:

3.1 Mandatory prerequisites

- Kubernetes 1.12+
- Helm 3.0
- Docker Registry
  - Setup a local docker registry to store Designer and DAS docker images.
- 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.
- 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 manifest package includes a sample YAML file to create an NFS server. Use this only for a demo/lab setup purpose.
- Genesys Web Services (GWS) 9.x
  - Configure GWS to work with a compatible version of Configuration Server.

- Other Genesys Components
  - ORS ORS 8.1.400.x
  - Nexus 9.x
  - URS 8.1.400.x

3.2 Optional prerequisites

- Elasticsearch 7.8.0
  - Elasticsearch is used for Designer Analytics and audit trail.
- Redis 3.2.x
  - Redis is used for resource index caching and multi-user collaboration locks on Designer resources.

4. Deployment configuration settings (Helm values)

This section provides 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.

4.1 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
`deployment.replicaCount`	Number of services to be created.	Mandatory	2
`deployment.maxReplicas`	Maximum number of replicas created. It is recommended to configure this setting if auto-scaling is used.	Optional	10
`deployment.strategy`	The strategy to select which type of resources to deploy. Valid values are: `default, service, volume, ingress`. • **volume** - for blue/ green upgrade, this is to create a Persistent Volume Claim (PVC) for the first time.	Mandatory	service

	• **ingress** - for the blue/green upgrade, this is to create an ingress for the first time and update the ingress during service cutover. • **service** - for upgrading the blue/green Designer service. • **default** - for performing a rolling upgrade		
`deployment.green`	This is to deploy/upgrade the Designer service in a blue-green upgrade strategy. Valid values are: `blue, green`.	Optional	green
`desImage.repository`	Docker repository for the Designer image.	Mandatory	pureengage-docker-staging.jfrog.io/designer/designer
`desImage.tag`	Designer image version.	Mandatory	9.0.109.08.20
`desImage.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
`volumes.workspaceMountPath`	The path where the workspace volume is to be mounted inside the Designer container.	Mandatory	/designer/workspace (Changing this value is not recommended.)
`volumes.workspaceClaim`	Persistent volume claim name for the workspace.	Mandatory	designer-managed-disk
`volumes.workspaceClaimSize`	Size of the persistent volume claim for the workspace.	Mandatory	5Gi

	The persistent volume must be equal to or greater than this size.		
`volumes.workspaceStorageClass`	storageClassName provided in the persistent volume that is created for the Designer workspace (example, *nfs*).	Mandatory	manual
`volumes.logMountPath`	The path where the Designer logs volume is to be mounted inside the Designer container.	Mandatory	/designer/logs
`volumes.logClaim`	Persistent volume claim name for logs.	Mandatory	designer-logs
`volumes.logClaimSize`	Size of the persistent volume claim for the Designer logs. The persistent volume must be equal to or greater than this size.	Mandatory	5Gi
`volumes.logStorageClass`	storageClassName provided in the persistent volume that is created for the Designer logs (example, nfs).	Mandatory	manual
`healthApi.path`	Designer Health Check API path.	Mandatory	/health (Changing this value is not recommended.)
`healthApi.containerPort`	Container running port.	Mandatory	8888 (Changing this value is not recommended.)
`healthApi.startupDelay`	Health check will be started after a delay as specified in this setting.	Mandatory	20
`healthApi.checkInterval`	The interval between each health check request.	Mandatory	5
`healthApi.failureCount`	Number of health check failures to be considered before marking the container as instable or restart.	Mandatory	5
`designerEnv.enabled`	This enables providing environment variables as an input to Designer pods.	Mandatory	true (Changing this value is not recommended.)

	It uses `ConfigMap` to store the environment variables.		
`designerEnv.envs.DES_PORT`	Designer port for container (`port` in flowsettings.json).	Mandatory	8888
`designerEnv.envs.DES_APPLICATIONHOST`	DAS hostname (`applicationHost` in flowsettings.json).	Mandatory	das
`designerEnv.envs.DES_APPLICATIONPORT`	DAS port (`applicationPort` in flowsettings.json).	Mandatory	80
`designerEnv.envs.DES_DEPLOY_URL`	This is normally not changed. It is the relative path to the workspace on DAS. The default value `/workspaces` should be always be used (`deployURL` in flowsettings.json).	Mandatory	/workspaces
`designerEnv.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).	Mandatory	true (Changing this value is not recommended.)
`designerEnv.envs.DES_HTCC_SERVER`	GWS server host (`htccserver` in flowsettings.json), for example, `gws.genhtcc.com`.	Mandatory	gws-usw1-int.genhtcc.com
`designerEnv.envs.DES_HTCC_PORT`	GWS server port (`htccport` in flowsettings.json), for example, 80.	Mandatory	80
`designerEnv.envs.DES_ENABLE_ANALYTICS`	To enable or disable Designer Analytics (`enableAnalytics` in flowsettings.json).	Optional	false
`designerEnv.envs.DES_ES_URL`	Elasticsearch URL (for example, http://es-service:9200), `esUrl` in flowsettings.json.	Optional	http://es-spot.usw1.genhtcc.com
`designerEnv.envs.DES_ES_SERVER`	Elasticsearch Server HostName (for example, es-service), `esServer` in flowsettings.json.	Optional	es-spot.usw1.genhtcc.com
`designerEnv.envs.DES_ES_PORT`	Elasticsearch port (for example, 9200), `esPort`	Optional	80

	in flowsettings.json.		
`designerEnv.envs.DES_FILE_LOGGING_ENABLED`	Enable file logging. If not enabled, Designer will output only verbose logs.	Mandatory	false
`designerSecrets.enabled`	This enables providing the GWS client ID / secret as an input to Designer pods. It uses Kubernetes Secrets to store the GWS client credentials.		true
`designerSecrets.GWS_Client_id`	GWS Client ID, create a new GWS client if it doesn't exist, steps are explained in the platform settings section.	Mandatory	designer-secret
`designerSecrets.GWS_Client_secret`	GWS client secret	Mandatory	ZXh0ZXJuYWxfYXBpX2NsaWVudA== (This value is valid only for lab deployments.)
`service.type`	Service type (either `ClusterIP` or `NodePort` or `LoadBalancer`).	Mandatory	ClusterIP
`service.port`	Designer service port to be exposed in the cluster.	Mandatory	8888
`service.targetPort`	Designer application port running inside the container.	Mandatory	8888
`service.nodePort`	Port to be exposed in case `service.type=NodePort`.	Optional	Sample value : 30180
`ingress.enabled`	Enable/Disable ingress. Ingress should be enabled for all cases except a lab/demo setup.	Mandatory	true
`ingress.paths`	Ingress path	Mandatory	[/]
`ingress.hosts`	Hostnames to be configured in ingress for the Designer service.	Mandatory	ssdev1.genhtcc.com
`ingress.tls`	TLS config for ingress.	Optional	[]
`resources.limits.cpu`	Maximum amount of CPU processing power that K8s allocates for the container.	Mandatory	600m
`resources.limits.memory`	Maximum amount of	Mandatory	1Gi

	memory K8s allocates for the container.		
`resources.requests.cpu`	Guaranteed CPU allocation for the container.	Mandatory	500m
`resources.requests.memory`	Guaranteed memory allocation for the container.	Mandatory	512Mi
`securityContext.runAsUser`	Controls which user ID the containers are run with. This can be configured to run Designer as a non-root user. Currently, only a **Genesys** user is supported by the Designer base image. 500 is the ID of the Genesys user and it cannot be modified. 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	500
`securityContext.runAsGroup`	Controls which primary group ID the containers are run with. This can be configured to run Designer as a non-root user. Currently, only a **Genesys** user group (GID - 500) is supported by the Designer base image.	Optional	500

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

`affinity`	The K8s standard node affinity and anti-affinity configurations can be added here. Refer to this K8s document for sample values.	Optional	{}
`tolerations`	Tolerations works with taints to ensure that	Optional	[]

	pods are not scheduled onto inappropriate nodes. Refer to this K8s document for sample values.		
`hpa.enabled`	Enables K8s Horizontal Pod Autoscaler (HPA). It automatically scales the number of pods based on average CPU utilization and average memory utilization. More information about HPA is available here.	Optional	false
`hpa.targetCPUPercent`	The K8s HPA controller will scale up/down pods based on the target CPU utilization percentage specified. It scales up/down pods between the range `deployment.replicaCount` to `deployment.maxReplicas`.	Optional	70
`hpa.targetMemoryPercent`	The K8s HPA controller will scale up/down pods based on the target memory utilization percentage specified. It scales up/down pods between the range `deployment.replicaCount` to `deployment.maxReplicas`.	Optional	70
`annotations`	Enables Kubernetes Annotations. Refer to this document for more information on K8s Annotations. The Designer UI requires Session Stickiness if the replica count is more than 1. Configure session stickiness based on the ingress controller type. Ingress configuration like session stickiness can be configured here.	Optional	{}
`labels`	Any custom labels can be configured. It is a key and value, for example, key:value.	Optional	tenant: shared

4.2 DAS deployment settings

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

Parameter	Description	Mandatory?	Default Value
`deployment.replicaCount`	Number of services to be created.	Mandatory	2
`deployment.maxReplicas`	Maximum number of replicas created. It is recommended to configure this setting if auto-scaling is used.	Optional	10
`deployment.strategy`	The strategy to select which type of resources to deploy. Valid values are : `default, service, volume, ingress`. • **ingress** - for the blue/green upgrade, this is to create an ingress for the first time and update the ingress during service cutover. • **service** - for upgrading the blue/green DAS service. • **default** - for performing a rolling upgrade.	Mandatory	service
`deployment.green`	This is to deploy/upgrade the DAS service in a blue-green upgrade strategy. Valid values are: `blue, green`.	Optional	green
`dasImage.repository`	Docker repository for the DAS image.	Mandatory	pureengage-docker-staging.jfrog.io/designer/das
`dasImage.tag`	DAS image version.	Mandatory	9.0.111.05.5
`dasVolumes.workapceMountPath`	DAS workspace path inside the container.	Mandatory	/das/www/workspaces
`dasVolumes.workspaceClaim`	Persistent volume claim name for the workspace (must be the same as Designer's claim name).	Mandatory	designer-managed-disk
`dasVolumes.logMountPath`	DAS log path inside the	Mandatory	/das/log

	container.		
`dasVolumes.logClaim`	Persistent volume claim name for logs (must be the same as Designer's claim name).	Mandatory	designer-logs
`dasHealthApi.path`	DAS Health Check API path.	Mandatory	/health
`dasHealthApi.containerPort`	Container running port.	Mandatory	8081
`dasHealthApi.startupDelay`	Health check will be started after a delay as specified in this setting.	Mandatory	10
`dasHealthApi.checkInterval`	The interval between each health check request.	Mandatory	5
`dasHealthApi.failureCount`	Number of health check failures to consider before marking the container as instable or restart.	Mandatory	5
`dasService.type`	Service port (either `ClusterIP` or `NodePort` or `LoadBalancer`).	Mandatory	ClusterIP
`dasService.port`	DAS service to be exposed in the cluster.	Mandatory	8081
`dasService.targetPort`	DAS application port running inside the container.	Mandatory	8081
`dasService.nodePort`	Port to be exposed in case `service.type=NodePort`.	Optional	Sample value : 30280
`dasEnv.enabled`	This enables providing environment variables as an input to DAS pods. It uses `ConfigMap` to store the environment variables.	Mandatory	true
`dasEnv.envs.DAS_FILE_LOGGING_ENABLED`	Enable file logging. DAS supports only stdout logging, this must always be false.	Mandatory	false
`dasEnv.envs.DAS_LOG_LEVEL`	Enables log levels. Valid values are: `FATAL, ERROR, WARN, INFO, DEBUG, TRACE`.	Optional	DEBUG
`dasEnv.envs.DAS_STDOUT_LOGGING_ENABLE`	Enables standard output console logging.	Mandatory	true
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_ENABLED`	To enable or disable Designer Analytics. This config is required for DAS to initialize ES	Optional	false

	templates.		
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_HOST`	Elasticsearch Server HostName with http:// prefix (for example, http://es-service)	Optional	http://designer-es-client-service
`dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_PORT`	Elasticsearch port (for example, 80)	Optional	9200
`dasresources.limits.cpu`	Maximum amount of CPU processing power that K8s allocates for the container.	Mandatory	600m
`dasresources.limits.memory`	Maximum amount of memory K8s allocates for the container.	Mandatory	1Gi
`dasresources.requests.cpu`	Guaranteed CPU allocation for container.	Mandatory	400m
`dasresources.requests.memory`	Guaranteed Memory allocation for the container.	Mandatory	512Mi
`securityContext.runAsUser`	Controls which user ID the containers are run with. This can be configured to run DAS as a non-root user. Currently, only a **Genesys** user is supported by the DAS base image. 500 is the ID of the Genesys user and it cannot be modified.	Optional	500
`securityContext.runAsGroup`	Controls which primary group ID the containers are run with. This can be configured to run DAS as a non-root user. Currently, only a **Genesys** user group (GID - 500) is supported by the DAS base image.	Optional	500

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

`affinity`	The K8s standard node affinity and anti-affinity configurations can be added here. Refer to	Optional	{}

	this K8s document for sample values.		
`tolerations`	Tolerations works with taints to ensure that pods are not scheduled onto inappropriate nodes. Refer to this K8s document for sample values.	Optional	[]
`hpa.enabled`	Enables K8s Horizontal Pod Autoscaler (HPA). It automatically scales the number of pods based on average CPU utilization and average memory utilization. More information about HPA is available here.	Optional	false
`hpa.targetCPUPercent`	The K8s HPA controller will scale up/down pods based on the target CPU utilization percentage specified. It scales up/down pods between the range `deployment.replicaCount` to `deployment.maxReplicas`.	Optional	75
`hpa.targetMemoryPercent`	The K8s HPA controller will scale up/down pods based on the target memory utilization percentage specified. It scales up/down pods between the range `deployment.replicaCount` to `deployment.maxReplicas`.	Optional	70
`annotations`	Enables Kubernetes Annotations. Refer to this document for more information on K8s Annotations.	Optional	{}
`labels`	Any custom labels can be configured. It is a key and value, for example, key:value.	Optional	tenant: shared

5. Post deployment Designer settings

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

5.1 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. The *Flowsettings.json update* section (*8.2.2 Designer deployment process*) describes the steps to update the **flowsettings.json** file.

5.2 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.

5.3 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 CME or GAX.

5.4 Configuration settings reference table

> **Tip**
> As the following table extends beyond the margin of the page, use the horizontal scroll bar at the bottom of your browser window to view the complete table.

Category: Analytics							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
enableAnalytics (optional)	Yes	Yes			This flag enables or disables	true	false

					the analytics feature.		
esUrl (optional)	Yes	Yes			Elasticsearch URL	http://es-spot.usw1.genhtcc.com:80	
esServer (optional)	Yes	Yes			Elasticsearch Server HostName (for example, es-service)	es-spot.usw1.genhtcc.com	
esPort (optional)	Yes	Yes			Elasticsearch port	80	
ReportingURL (optional)			Yes	reporting	URL of Elasticsearch where Designer applications will report data.	http://es-spot.usw1.genhtcc.com:80	
esMaxQueryDuration (optional)	Yes	Yes			The maximum time range (in days) to query in Designer Analytics. Each day's data is stored in a separate index in Elasticsearch.	90	90
sdrMaxObjCount (optional)	Yes	Yes			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 the *milestones* or *activities*	20	

					visited in runtime are expected to be captured in an SDR.		
SdrTraceLevel (optional)	Yes	Yes			It controls the level of SDR detail that is recorded by the `blocks` array for each application. Currently, the valid values 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.	300	300

					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).		
Category: Audit							
Setting Name	lowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
enableESAuditLogs (optional)	Yes	Yes			Enable or Disable audit logs captured in Elasticsearch.	false	false
enableFSAuditLogs (optional)	Yes	Yes			Enable or Disable audit logs captured in the file system under the **logs** directory or in standard output.	true	true
maxAppSizeCompare (optional)	Yes	Yes			The maximum size of data object for	1000000	1000000

					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.		
enableReadAuditLogs (optional)	Yes	Yes			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.	false	false
Category: Authorization							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
disableRBAC (optional)	Yes	Yes			Controls if Designer reads and enforces permissions associated with the logged in user's roles.	false	false
rbacSection (optional)	Yes	Yes			In a Role object, the name of the section within the Annex	CfgGenesysAdministrator	CfgGenesysAdministrator

					where the privileges are stored.		
disablePBAC (optional)	Yes	Yes			Controls if Designer allows partitioning of the Designer workspace and restricts a user's access to Designer objects in the user's partitions.	false	false
Category: Collaboration							
Setting Name	flowsettings.json	tenantsettings.cfg	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
locking (optional)	Yes				The type of locking used, for an editing session of applications, modules, or data tables. Valid values : `file, redis, none`	file	file
Category: DAS							
Setting Name	flowsettings.json	tenantsettings.cfg	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
applicationHost (mandatory)	Yes				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.	das.usw1.genhost.com	localhost

applicationPort	Yes				The corresponding port to be used with applicationHost.	80	80
deployURL	Yes				This is normally not changed. It is the relative path to the workspace on DAS.	/workspace	/workspace
Category: Digital							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
rootsSRL (optional)	Yes	Yes			If specified, this is used to filter which Root Categories to display when selecting Standard Responses.	A REGular EXpression (REGEX).	
maxFlowEntryCount (optional)	Yes		Yes	flowsettings	Specify how many times the same application can process a specific digital interaction.	20	20
Category: External APIs							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
httpProxy (optional)	Yes	Yes	Yes	flowsettings	Specify the proxy used for external requests and nexus API calls (if `enable_proxy` is `true`).	http://vpcproxy-000-int.geo.genprim.co	
redundantHttpProxy (optional)	Yes	Yes	Yes	flowsettings	Specify the	http://vpcproxy-001-int.geo.genprim.co	

					backup proxy used for external requests and nexus API calls (if `enable_proxy` is `true`), when httpProxy is down.		
Category: Features							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value

features	Yes	Yes			This is an object. See the **5.5 Features** section for a list of supported features.		{
							nexus: true,
							enableBulkAudioImport: true
							}

Category: GWS							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
usehtcc	Yes				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.	true	false
htccServer	Yes				GWS Server	gws-usw1-int.genhtcc.com	gws-usw1-int.genhtcc.com
htccport	Yes				GWS Port	80	80
ssoLoginUrl	Yes				URL of GWS authentication UI. Designer redirects to this URL	https://gws-usw1.genhtcc.com	https://gws-usw1.genhtcc.com

					for authentication.		
maxConcurrentHTCCRequest (optional)	Yes				For batch operations to GWS, the max number of concurrent requests that Designer will send to GWS.	5	5
batchOperationResultTTL (optional)	Yes				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.	100000	100000
Category: Help							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
docsMicroserviceURL (optional)	Yes				URL for Designer documentation.		https://docs.genesys.com/Documentation/PSAAS/Public/Administrator/Designer
Category: IVR							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
recordingType (optional)	Yes	Yes			Specify the recording type to be used in Record block. Set as GIR. If the option is missing	GIR	GIR

					or blank, `Full Call Recording` type will be used.		
Category: Logging							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
`logging: { designer: { level: debug }, audit: { level: trace}, auditdebug: { level: debug }, cli: { level: debug } }` (optional)	Yes				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.	`logging: { designer: { level: debug}, audit: { level: trace }, auditdebug: { level: debug}, cli: { level: debug } }`	`logging: { designer: { level: debug }, audit: { level: trace }, auditdebug: { level: debug }, cli: { level: debug } }`
Category: Nexus							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
url (optional)			Yes	nexus	URL of Nexus that typically includes the API version path. For example, https://nexus-server/nexus/api/v3.		http://nex-dev.usw1.genhtcc.com
password (optional)			Yes	nexus	nexus x-api-key		dc4qeiro13nsof569dfn234

					created by Nexus deployment		
enable_proxy (optional)			Yes	nexus	Boolean value to indicate if httpProxy is used to reach Nexus.		false
profile (optional)			Yes	nexus	Enable Contact Identification via Nexus (for example, to enable Last Called Agent routing).		
Category: Process							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
port	Yes				Designer process port in the container. Normally, the default value should be left as is.	8888	3000
Category: Provisioning							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
primarySwitch (optional)	Yes	Yes			Specify the primary switch name if more than one switch is defined for the tenant. Designer fetches and works with route points from this switch.		us-west-1
Category: Routing							

Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
ewtRefreshTimeout (optional)			Yes	flowsettings	Specify the interval (in seconds) at which to refresh the Estimated Waiting Time when routing an interaction.	5	1
Category: Redis							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
`redis: { host: "", port: "", tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }` (optional)					Used by Designer for resource index caching and multi-user collaboration locks on Designer resources. It is a separate object and 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 seconds) of the	`redis: { host: "", port: "", tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }`	`redis: { host: redis.server.genhtcc.com port: 6379, tlsEnabled: true, lockTimeout: 120, listTimeout: 1800 }`

					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	DesignerEnv Section	Description	Sample Value	Default Value
zipFileSizeLimitInMegaBytes (optional)	Yes	Yes			Defines the maximum zipFile size limit (in megabytes) during bulk audio import.	50	No default.
disableCSRF (optional)	Yes	Yes			Disable CSRF attack protection. http://cwe.mitre.org/data/definitions/352.html By default, CSRF attack protection is enabled. It can be disabled by setting this	false	false

					flag to `true`.		
disableSecureCookie (optional)	Yes				Disable the secure cookies header	false	false
Category: Session							
Setting Name	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv Section	Description	Sample Value	Default Value
idleTimeout (optional)	Yes	Yes			Idle timeout, in seconds, before a user session is terminated while editing applications, modules, or data tables.	840	840
lockTimeout (optional)	Yes	Yes			Timeout, in seconds, before a resource lock is released, for an editing session of applications, modules, or data tables.	120	120
lockKeepalive (optional)	Yes	Yes			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.	15	15
Category: Workflow							
Setting	flowsettings.json	tenantsettings.json	DesignerEnv	DesignerEnv	Description	Sample	Default

Name				Section		Value	Value
maxBuilds (optional)	Yes	Yes			Specify the maximum number of builds allowed per application.	20	20
enablePTE (optional)			Yes	flowsettings	Boolean value to indicate if PTE objects are enabled at runtime.	true	false

5.5 Features

The features specified here must be configured under the `features` object in the `flowsettings.json` file.

For example,

```
        features: {
callbackv2: true,
..
..
}
```

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

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	false

					validate invalid grammar files during grammar upload and you can upload only valid grammar files (GRXML or Nuance compiled binary grammar files).	
	`externalAudio`	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	true

					survey feature.	
Milestone	enableImplicitModuleMilestones	Optional	Yes	Yes	Enable reporting each Shared Module call as an internal milestone. If disabled, Shared Module calls will not generate a milestone.	false
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

6. Logging

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 = false
   dasEnv.envs.DAS_STDOUT_LOGGING_ENABLE = true
   ```

6.1 Log levels

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

6.1.1 Designer

1. Configure the `logging` setting in the flowsettings override (**flowsettings.yaml**) - Refer to section *5.4 Configuration settings reference table* for option descriptions.
2. Execute the steps in the *Flowsettings.json update* section (*8.2.2 Designer deployment process*) for the changes to take effect .

6.1.2 DAS

1. Configure the `dasEnv.envs.DAS_LOG_LEVEL` setting in the Helm **das-values.yaml** file. Refer to section *4.2 DAS deployment settings* for setting descriptions.
2. Execute the steps in the *Upgrade non production color* section (*8.2.3 DAS deployment process*). The same DAS version running in production can be used for the upgrade,
3. Execute the steps in the *Cutover* section (*8.2.3 DAS deployment process*).

7. Platform / Configuration Server and GWS settings

This section explains the Configuration Server objects and settings required for Designer.

7.1 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.

- **DesignerDeveloper** - 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.
- **DesignerBusinessUser** - These users cannot create objects but they can manage them (for example, upload audio, change data tables, and view analytics).
- **DesignerAnalytics** - These users only have access to Designer Analytics.
- **DesignerAdmin** - These users can set up and manage partitions associated with users and Designer objects.
- **DesignerOperations** - 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 Manifest** 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 CME folders - Scripts and Transactions.

7.2 Create the DesignerEnv transaction list

Designer requires a transaction list for configuration purposes as described in other sections of this document. To set this up:

1. Create a transaction list called **DesignerEnv**.
2. Import the file **configuration/DesignerEnv.conf**, located in the Designer Deployment Manifest package.
3. Edit any values according to the descriptions provided in the **Designer settings** section.
4. Save the list.
5. Ensure Designer users have at least read access to the **DesignerEnv** transaction list.

7.3 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	
URS	'http' port, protocol = 'http'		Required only for Route Agent block to work.

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

7.4 GWS Configuration

7.4.1 Create Contact Center

Create a contact center in GWS if it is not already created. Refer to the GWS documentation for more information on this.

7.4.2 Create GWS Client

Create new GWS client credentials if they are not already created . Refer to the GWS documentation for more information on this.

8. Deployment

This section describes the deployment process for Designer and DAS.

8.1 Preparation

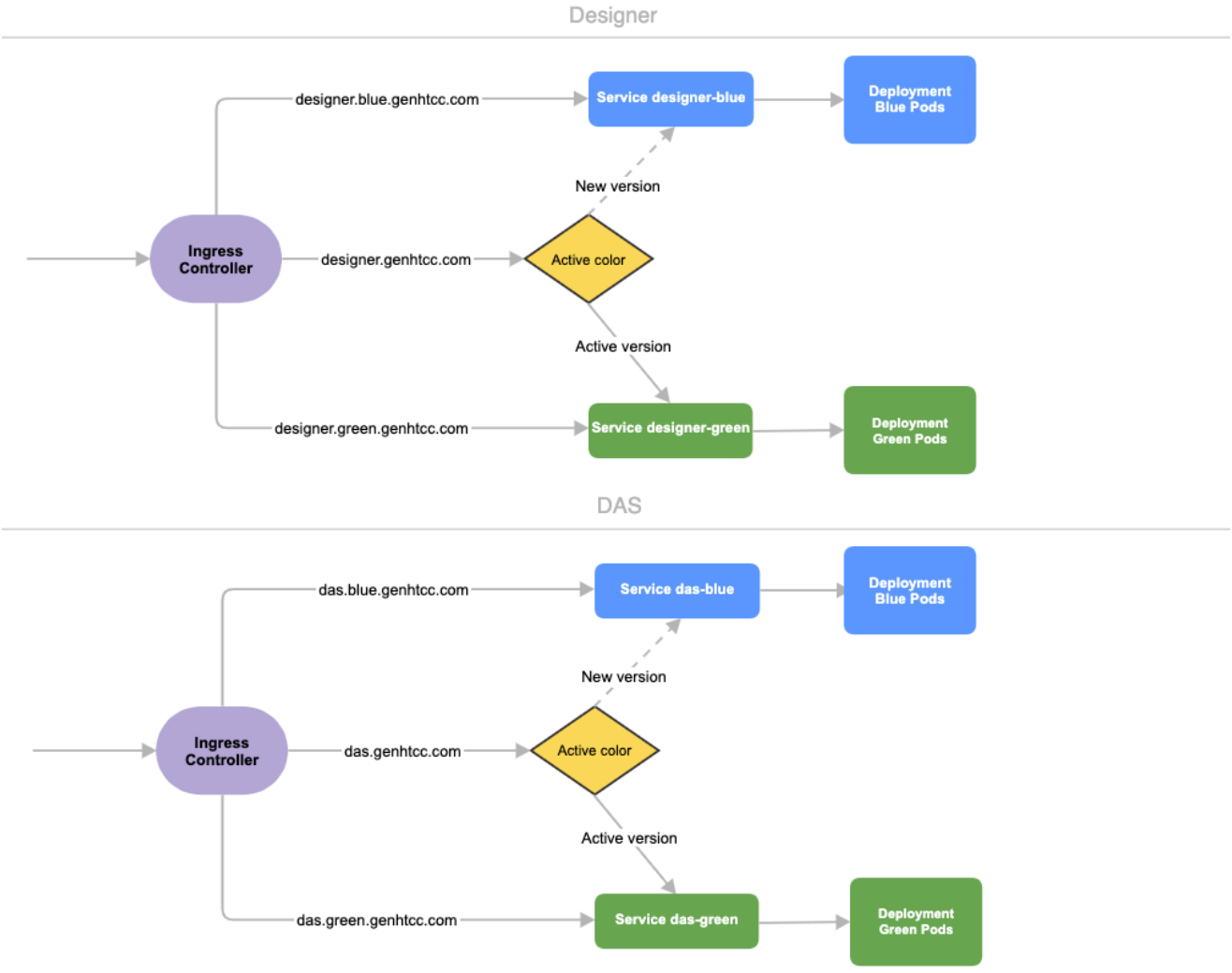
Before you deploy Designer and DAS using Helm charts, complete the following preparation steps:

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 manifest package and extract to the current working directory.
7. Configure Designer and DAS value overrides (designer-values.yaml and das-values.yaml) - please ensure the mandatory settings are configured. If the blue-green deployment process is used, Ingress settings are explained in the following section.

8.2 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 a Helm package and after the testing is completed, the traffic is moved to the newly created deployment, which becomes the ACTIVE environment, and the old environment becomes INACTIVE. This way, a fast rollback is possible by just changing route if a new issue is found with live traffic. The old inactive deployment can be removed once the new active deployment becomes stable.

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



8.2.1 Preparation for Blue-Green deployment

Before you deploy Designer and DAS using the Blue-Green deployment strategy, complete the following preparation steps:

1. Configure the Ingress host names for Designer. Create 3 host names as shown below.
   The Blue service host name must contain the string *blue*, for example, `designer.blue.genhtcc.com` or `designer-blue.genhtcc.com`. The Green service host name must contain the string *green*, for example, `designer.green.genhtcc.com` or `designer-green.genhtcc.com`. The Blue/Green services can be accessed separately with the Blue/Green host names as shown in this example:
   `designer.genhtcc.com` (production host URL used for external access).
   `designer.blue.genhtcc.com` (URL for Blue service testing).
   `designer.green.genhtcc.com` (URL for Green service testing).
2. Configure the host names in the **designer-values.yaml** file under ingress. Annotations and paths can be modified based on the requirement.
   For example,
   ```
   ingress:
   enabled: true
   annotations: {}
   paths: [/]
   hosts:
   - designer.genhtcc.com
   - designer.blue.genhtcc.com
   - designer.green.genhtcc.com
   ```
3. Configure the Ingress host names for DAS. Create 3 host names as shown below.
   The Blue service host name must contain the string *blue*, for example, `das.blue.genhtcc.com` or `das-blue.genhtcc.com`. The Green service host name must contain the string *green*, for example, `das.green.genhtcc.com` or `das-green.genhtcc.com`. he Blue/Green services can be accessed separately with the Blue/Green host names as shown in this example:
   `das.genhtcc.com` (the production host URL used for external access).
   `das.blue.genhtcc.com` (URL for Blue service testing).
   `das.green.genhtcc.com` (URL for Blue service testing).
4. Configure the host names in the **das-values.yaml** file under ingress. Annotations and paths can be modified based on the requirement.
   For example,
   ```
   ingress:
   enabled: true
   annotations: {}
   paths: [/]
   hosts:
   - das.genhtcc.com
   - das.blue.genhtcc.com
   - das.green.genhtcc.com
   ```

8.2.2 Designer deployment process

Initial deployment

The resources's ingress and persistent volume claims (PVC) must be created initially before deploying the Designer service as these resources are shared between the Blue/Green services and must be created at the very beginning of the deployment. They will not be needed for subsequent upgrades. The required values are passed using the SET command as shown below or by modifying 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 designer-volume -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=volume`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=volume` - indicates that this helm install will create persistent volume claim.
2. Create ingress rules for the Designer service (assuming the ingress service name is `designer-ingress`):
   `helm upgrade --install designer-ingress -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=ingress --set-string deployment.color=green`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=ingress` - indicates that this helm install will create ingress rules for the Designer service.
   `deployment.color=green` - indicates that the current production instance (active) color is Green.
3. Deploy the Designer service to the color selected in step 2. In this case, Green is selected and assuming the service name is `designer-green`:
   `helm upgrade --install designer-green -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=service --set desImage.tag=9.0.1xx.xx.xx --set-string deployment.color=green`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=service` - indicates that the Designer service will be installed.
   `desImage.tag=9.0.1xx.xx.xx` - indicates the Designer version to be installed, for example, 9.0.116.07.10.
   `deployment.color=green` - indicates that the Green color service will be installed.

Upgrade non-production color

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.genhtcc.com        /     designer-green:http (10.244.0.23:8888)
designer.green.genhtcc.com  /     designer-green:http (10.244.0.23:8888)
designer.blue.genhtcc.com   /     designer-blue:http (10.244.0.45:8888)
```

2. Deploy the Designer service into the non-production color. In the above example, Blue is the non-production color (assuming the service name is `designer-blue`):
   `helm upgrade --install designer-blue -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=service --set desImage.tag=9.0.1xx.xx.xx --set-string deployment.color=blue`
   Note: The overrides passed as an argument in the above helm upgrade:
   `deployment.strategy=service` - indicates that the Designer service will be installed.
   `desImage.tag=9.0.1xx.xx.xx` - indicates the Designer version to be installed, for example, 9.0.116.08.12.
   `deployment.color=blue` - indicates that the Blue color service will be installed.
3. The non-production color can be accessed with the non-production host name (for example - `designer.blue.genhtcc.com`), any testing can be done using this URL.

Cutover

Once testing is completed on the non-production color, traffic can be moved to the new version by updating the ingress rules.

1. Update the Designer Ingress with the new deployment color by running the below command (in this case, Blue is the new deployment color, that is, the non-production color):
   `helm upgrade --install designer-ingress -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=ingress --set-string deployment.color=blue`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=ingress` - indicates that this helm install will create ingress rules for the Designer service.
   `deployment.color=blue` - indicates that the current production (active) color is Blue.
2. Verify the ingress rules by executing the command `kubectl describe ingress designer-ingress`. The production host name should point to the new color service.

Workspace upgrade

Workspace resources must be upgraded after cutover. This will upgrade the system resources in the Designer workspace.

1. Log in to one of the Designer pods with the command: `kubectl exec -it bash`.
2. Execute the migration command (this will create 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 will upgrade system resources, such as system service PHP files, internal audio files, and callback resources):
   `node ./bin/cli.js workspace-upgrade -t`
   `contact_center_id` is the contact center ID created in GWS for this tenant. The workspace resources are located within the contact center ID folder *(/workspaces//workspace)*.

> **Important**
>
> The above steps - upgrade non production color, cutover, and workspace upgrade will also be used for further upgrades.

Flowsettings.json update

Post deployment, the **flowsettings.json** file can be modified via helm install using the below steps:

1. Download the current **flowsettings.json** file from the location: */designer/flowsettings.json*.
2. Modify the necessary settings (refer to section *5.4 Configuration settings reference table*).
3. Create a new YAML file, for example, **flowsettings.yaml**.
4. Copy and paste the above modified **flowsettings.json** content in the new **flowsettings.yaml** file:
   `flowsettings:`
   For example:
   `flowsettings: {`
   `port:8888,`

```
usehtcc:true,
htccserver:gws-int-genhtcc.com,
htccport:80,
....
....
}
```

5. Run the below helm upgrade command on the non-production color service. It can be done as part of Designer upgrade by passing the **flowsettings.yaml** in the extra argument `--values`. In this case, the new Designer version can be used for the upgrade. If it is only a **flowsettings.json** update, the same Designer version will be used.
   `helm upgrade --install designer-blue -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=service --set desImage.tag=9.0.1xx.xx.xx --set-string deployment.color=blue --values flowsettings.yaml`
   The non-active color Designer will have updated settings after the above upgrade.
6. Once testing is completed on the non-production service, perform the cutover steps as mentioned in the Cutover section. Now, the production service will contain the changed settings.

Rollback

- If any blocking issues are noticed in the current production service, traffic can be rolled back to the previous active color by updating the ingress rules:
  `helm upgrade --install designer-ingress -f designer-values.yaml designer-9.0.xx.tgz --set deployment.strategy=ingress --set-string deployment.color=green`
  Rollback of workspace resources is generally not required as the workspace resources shipped with Designer are backward and forward compatible. If required, the workspace can be upgrade from the old version, but it is not necessary. Future new version upgrades must run the workspace upgrade as per the normal process.
  Rollback of applications and shared modules is also not required as these resources are also backward and forward compatible with Designer.

8.2.3 DAS deployment process

Initial deployment

The ingress must be created initially before deploying the DAS service as it is shared between the Blue/Green services and must be created at the very beginning of the deployment. It will not be needed for subsequent upgrades. The required values are passed using the `SET` command as shown below or by modifying the **values.yaml** file.

1. Create ingress rules for the Designer service (assuming the ingress service name is `das-ingress`):
   `helm upgrade --install das-ingress -f das-values.yaml das-9.0.xx.tgz --set deployment.strategy=ingress --set-string deployment.color=green`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=ingress` - indicates that this helm install will create ingress rules for the DAS service.
   `deployment.color=green` - indicates that the current production instance (active) color is Green.
2. Deploy the DAS service to the color selected in step 1. In this case, Green is selected and assuming the service name is `das-green`:
   `helm upgrade --install das-green -f das-values.yaml das-9.0.xx.tgz --set deployment.strategy=service --set dasImage.tag=9.0.1xx.xx.xx --set-string deployment.color=green`
   Note: The overrides passed as an argument to the above helm upgrade command:
   `deployment.strategy=service` - indicates that the DAS service will be installed.

`dasImage.tag=9.0.1xx.xx.xx` - indicates the DAS version to be installed, for example, 9.0.111.04.4.
`deployment.color=green` - indicates that the Green color service will be installed.

Upgrade non-production color

1. Identify the current production color by checking the DAS ingress rules (`kubectl describe ingress das-ingress`). Green is the production color in the below example as the production host name points to the Green service.

kubectl describe ingress das-ingress

```
Host                        Path  Backends
----                        ----  --------
das.genhtcc.com             /     das-green:http (10.244.0.5:8081)
das.green.genhtcc.com       /     das-green:http (10.244.0.5:8081)
das.blue.genhtcc.com        /     das-blue:http (10.244.0.37:8081)
```

2. Deploy the DAS service into the non-production color. In the above example, Blue is the non-production color (assuming the service name is `das-blue`):
`helm upgrade --install das-blue -f das-values.yaml das-9.0.xx.tgz --set deployment.strategy=service --set dasImage.tag=9.0.1xx.xx.xx --set-string deployment.color=blue`
Note: The overrides passed as an argument to the above helm upgrade command:
`deployment.strategy=service` - indicates that the DAS service will be installed.
`dasImage.tag=9.0.1xx.xx.xx` - indicates the DAS version to be installed , for example, 9.0.111.05.5.
`deployment.color=blue` - indicates that the Blue color service will be installed.
3. The non-production color can be accessed with the non-production host name (for example - `das.blue.genhtcc.com`), any testing can be done using this URL.

Cutover

Once testing is completed on the non-production color, traffic can be moved to the new version by updating the ingress rules.

1. Update the DAS Ingress with the new deployment color by running the below command (in this case, Blue is the new deployment color, that is, the non-production color):
`helm upgrade --install das-ingress -f das-values.yaml das-9.0.xx.tgz --set deployment.strategy=ingress --set-string deployment.color=blue`
Note: The overrides passed as an argument to the above helm upgrade command:
`deployment.strategy=ingress` - indicates that this helm install will create ingress rules for the DAS service.
`deployment.color=blue` - indicates that the current production (active) color is Blue.
2. Verify the ingress rules by running the command kubectl describe ingress das-ingress. The production host name should point to the new color service.

> **Important**

> The above steps - upgrade non production color and cutover will also be used for further upgrades.

Rollback

If any blocking issues are noticed in the current production service, traffic can be rolled back to the previous active color by updating the ingress rules:

```
helm upgrade --install das-ingress -f das-values.yaml das-9.0.xx.tgz --set
deployment.strategy=ingress --set-string deployment.color=green
```

8.3 Rolling upgrade

A rolling upgrade is not recommended. Use the Blue/Green upgrade procedure.

8.4 Uninstall

To uninstall a service/volume/ingress rules:

```
helm uninstall
```

9. Enabling optional features

9.1 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.

9.1.1 Designer changes

1. Configure the following settings in flowsettings override (**flowsettings.yaml**) - Refer to section *5.4 Configuration settings reference table* 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 *Flowsettings.json update* section (*8.2.1 Designer deployment process*).

9.1.2 DAS changes

1. Configure the following settings in the helm **das-values.yaml** file. Refer to the *4.2 DAS deployment settings* section for setting descriptions.
   `dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_ENABLED = true`
   `dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_HOST`
   `dasEnv.envs.DAS_SERVICES_ELASTICSEARCH_PORT`
2. Perform the steps in the *Upgrade non production color* section (*8.2.2 DAS deployment process*). The same DAS version running in production can be used for the upgrade.
3. Perform the steps in the *Cutover* section (*8.2.2 DAS deployment process*).

10. Cleanup

10.1 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-*`

11. Limitations

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.