Billing Data Service Private Edition Guide

5/20/2022

Table of Contents

Contents

Find links to all the topics in this guide.

Related documentation:

-

Billing Data Service (BDS) is a service available with the Genesys Multicloud CX private edition offering.

Overview

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

- About BDS
- Architecture
- High availability and disaster recovery

Configure and deploy

Find out how to configure and deploy BDS.

- Before you begin
- Configure BDS
- Provision BDS
- Deploy BDS
- Upgrade, rollback, or uninstall BDS

Operations

Learn how to monitor BDS with metrics and logging.

- Observability
- BDS metrics and alerts

About BDS

Contents

Learn about Billing Data Service (BDS) and how it works in Genesys Multicloud CX private edition.

Related documentation:

-

Billing Data Service

BDS provides the data required for the Genesys financial organization to bill customers for subscription services.

The main functions of BDS are:

- Daily extraction of historical reporting and configuration data needed for billing.
- Aggregation of that data into daily and monthly summary files.
- Automated transmission of daily and monthly files to Genesys back-office for invoicing.
- Local storage of encrypted billing files as backup, or for other purposes.
- Creates local reports based on the various agent seat metric calculations.

Supported Kubernetes platforms

Billing Data Service is supported on the following cloud platforms:

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

See the Billing Data Service release notes for information about when support was introduced.

Architecture

Learn about Billing Data Service (BDS) architecture.

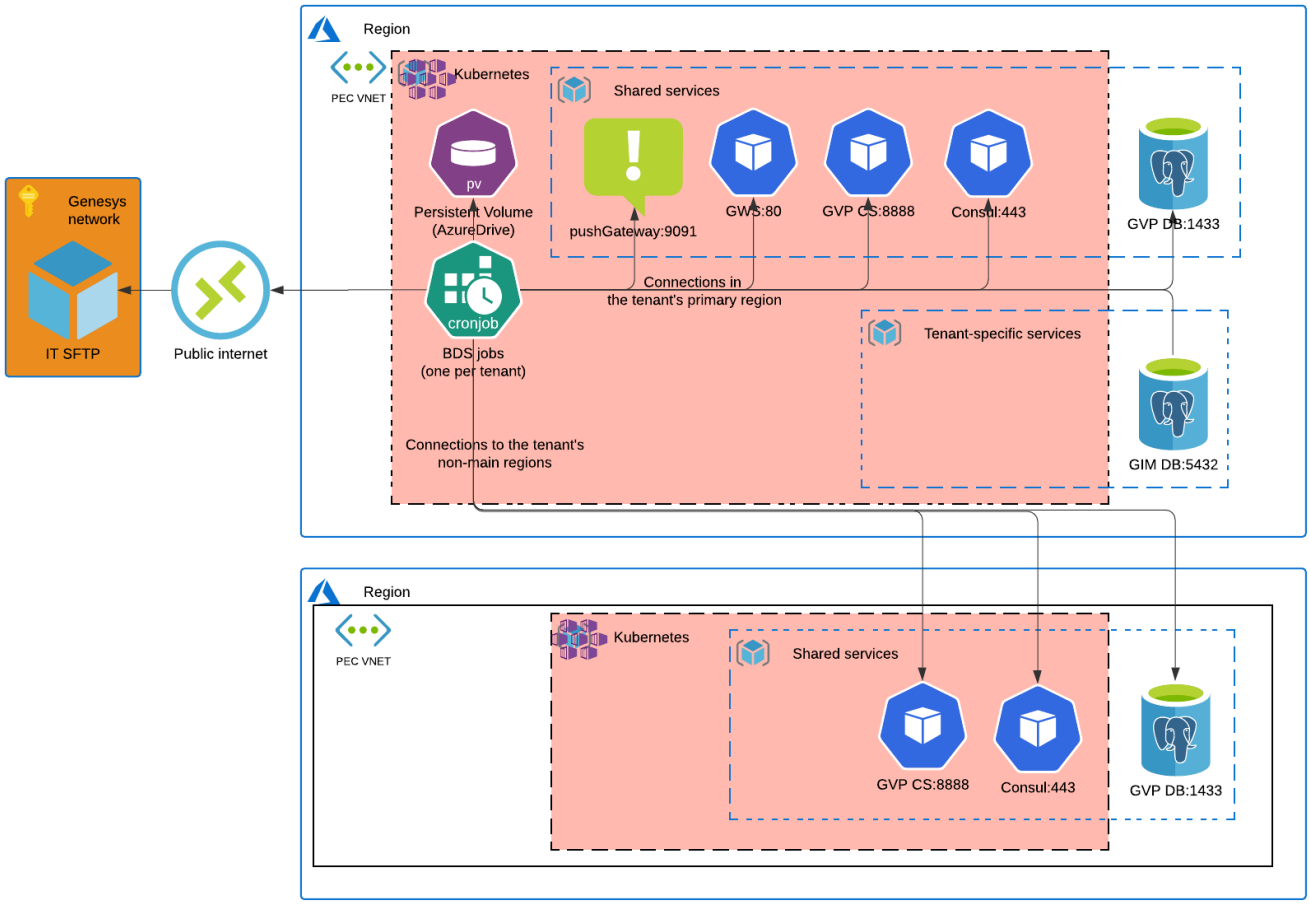
Related documentation:

-

For more information about the overall architecture of Genesys Private Edition Cloud, see: Architecture.

BDS operates as a CronJob, which runs once a day, consisting of a sequence of steps that collect various data from sources -- such as Genesys Info Mart, GVP RS, and Configuration -- and transforms that data into usage statistics that track your consumption of Genesys services, which is used for billing purposes. Files are securely transferred to Genesys back-office using SFTP.

Autoscaling is not supported.


Region
PEC VNET
Kubernetes
Shared services
pv
Persistent Volume
(AzureDrive)
pushGateway:9091
GWS:80
GVP CS:8888
Consul:443
GVP DB:1433
Genesys
network
IT SFTP
Public internet
cronjob
BDS jobs
(one per tenant)
Connections in
the tenant's primary region
Tenant-specific services
GIM DB:5432
Connections to the tenant's
non-main regions
Region
PEC VNET
Kubernetes
Shared services
GVP CS:8888
Consul:443
GVP DB:1433

High availability and disaster recovery

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

Related documentation:

-

BDS is a Cronjob that runs on a per-tenant basis, so High Availability (HA) is not applicable.

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

Before you begin

Contents

Find out what to do before deploying Billing Data Service (BDS).

Related documentation:

-

Limitations and assumptions

- In a private edition environment, Billing Data Service (BDS) currently supports deployments **only** on the specific Kubernetes platforms cited in About BDS. Deploying BDS in environments that are **not** officially tested and certified by Genesys is not supported. If your organization requires a special business case to deploy BDS in an uncertified or unsupported environment (for example, mixed-mode), contact your Genesys Account Representative or Customer Care.

Download the Helm charts

For general information about downloading containers, see: Downloading your Genesys Multicloud CX containers.

To learn what Helm chart version you must download for your release, see Helm charts and containers for Billing Data Service

The BDS Helm chart defines a CronJob:

BDS Helm chart package [bds-cronjob-.tgz]: Scripts in this container run Extraction, Transformation, and Load process according to the main configuration specified as Config Map.

BDS queries data primarily from Genesys Info Mart, GVP Reporting Server, and the Framework Configuration Management Environment, and can transmit the data to the SFTP server.

Third-party prerequisites

Before deploying BDS, configure the following 3rd party components:

Third-party services

Name	Version	Purpose	Notes
Consul	1.9.5 - 1.9.x	Service discovery, service mesh, and key/value store.	Provision Consul to provide `gvp_config_dbid` and `ivr_app_dbid` parameters with Tenant

Name	Version	Purpose	Notes

			ID and default IVR application ID values from the GVP Configuration Server database.
			Where,
			• **gvp_config_dbid** is the DB ID for a particular tenant.
			• **ivr_app_dbid** is the DB ID for the default IVR application for that tenant.
			Example request:
			`curl -H "Authorization: Bearer " https:///v1/kv/tenants//gvp`
			`{ "gvp_config_dbid": "161", "ivr_app_dbid" : "217" }`

Command Line Interface		The command line interface tools to log in and work with the Kubernetes clusters.	

Storage requirements

To deploy a BDS container on a node (or a worker computer), you need the following storage elements:

- dual-core vCPU
- 8 GB RAM
- 40 GB PVC*
- Database - BDS does not require any database but extracts the data from the GIM database, that is, Postgres. Hence, BDS supports any Postgres version that GIM currently supports.

* BDS creates the persistent volume claim (PVC) file storage to save the extracted and transformed files with current ETL time stamps.

- The file share must be persistent, and must be mounted inside the pod to **/mnt/fileshare** folder. **Note:** Mount name cannot be changed, once created.

- The information and files necessary to start BDS in between launches is stored on PVC.
- Data in the file share must be backed up with a retention period of 90 days, and data protection.

Network requirements

Not applicable

Browser requirements

Not applicable

Genesys dependencies

BDS interoperates with the following Genesys software applications to receive billing data:

- Genesys Voice Platform Reporting Server (GVP RS) 9.0 or later
- Genesys Info Mart 8.5.015.19 or later

Depending on the metrics that you need, you must ensure that the corresponding data sources are available in your environment for BDS to generate meaningful reports.

To build requests for main data sources, BDS obtains additional information from the following sources:

- GVP Config Server
- Genesys GWS

Configure BDS

Contents

Learn how to configure Billing Data Service (BDS).

Related documentation:

-

Override Helm chart values

You can override values in the Helm charts to configure Private Edition. For more information about overriding Helm chart values, see the "suite-level" documentation about how to override Helm chart values: Overriding Helm chart values

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

This page provides an overview of configuration; detailed steps appear on the Provision BDS and Deploy BDS pages.

The following table lists values you can override for BDS:

Parameter	Description	Default value	Valid values	Notes
tenantName	Name of Tenant	""	lower case string	
podLabels.app	Label	bus	string	
podLabels.name	Labels	bds	string	
nameSpace	Namespace where BDS is deployed	bds	string	Leave blank if not used
bdsApp.deployment.job.schedule	Cronjob schedule	"0 3,15 * * *"	Cronjob schedule format	
bdsApp.image.registry	Image registry name	{}	string	
bdsApp.image.repository	Image repository name	cloudbilling/	string	
bdsApp.image.tag	Image tag	""	string	
bdsApp.image.pullPolicy	Image pull Policy	IfNotPresent	Policy	
bdsApp.image.pullSecrets.name	Secrets for docker registry	""	string	Leave blank if not used
bdsApp.container.env.pgwUrl	Prometheus Push Gateway URL. Set to empty string if Push Gateway is not available.	"http://prometheus-pushgateway.monitoring.svc.cluster.local:9091/metrics"	URL	

Parameter	Description	Default value	Valid values	Notes
bdsApp.container.env.BDS_Value	BDS deployment type	MULTICLOUD	string	change to MULTICLOUD_PE
bdsApp.volumes.pvc.claim	PVC used to mount file share	{}	string	
bdsApp.config.name	Name of the config map	bds-config	string	
bdsApp.gvars.name	Name of the config map	bds-config	string	
bdsApp.secrets.gvp.secretName	GVP shared secret name	shared-secret-gvp	string or null	Leave blank if not used
bdsApp.secrets.gvp.volumes.ProviderClassName	Populate if CSI is used	""		Leave blank if not used
bdsApp.secrets.gvp.volumes.driver	Populate if CSI is used	""		Leave blank if not used
bdsApp.secrets.gim.secretName	GIM shared secret name	shared-gim-db-t	string or null	Leave blank if not used
bdsApp.secrets.gim.vaultPartOfSecretName	Tenant ID is used as part of secret name	""	string	
bdsApp.secrets.gim.volumes.ProviderClassName	Populate if CSI is used	""	string or null	Leave blank if not used
bdsApp.secrets.gim.volumes.driver	Populate if CSI is used	""	string or null	Leave blank if not used
bdsApp.secrets.consul.secretName	Consul shared secret name	shared-consul-consul-bds-token	string or null	Leave blank if not used
bdsApp.secrets.consul.volumes.ProviderClassName	Populate if CSI is used	""	string or null	Leave blank if not used
bdsApp.secrets.consul.volumes.driver	Populate if CSI is used	""	string or null	Leave blank if not used
bdsApp.secrets.gws.secretName	GWS shared secret name	""	string or null	Leave blank if not used
bdsApp.secrets.gws.mount.name	GWS secret mount name	""	string or null	Mandatory if GWS shared secret used
bdsApp.secrets.manualSecretNames	Manual secret names	""	string	Mandatory
bdsApp.resources.limits.cpu	Maximum CPU count	2	integer	
bdsApp.resources.limits.memory	Maximum Memory volume	4Gi		
bdsApp.resources.requests.cpu	Guaranteed amount of CPU	0.25	percent / 100	
bdsApp.resources.requests.memory	Guaranteed amount of memory	1Gi		
bdsApp.priorityClassName		""		Leave blank if not used
bdsApp.nodeSelector		""		Leave blank if not

Parameter	Description	Default value	Valid values	Notes
				used
bdsApp.monitoring.enabled	Turn on/off monitoring	"false"	"false" / "true"	
bdsApp.secrets.gim.mounts.name	Name of the volume mount for GIM secrets, if stored as volume.	shared-secret-gim	string	Must be set to "" or null if manual secrets are used.
bdsApp.secrets.gvp.mounts.name	Name of the volume mount for GVP secrets, if stored as volume.	shared-secret-gvp	string	Must be set to "" or null if manual secrets are used.
bdsApp.secrets.consul.mounts.name	Name of the volume mount for Consul secrets, if stored as volume.	shared-secret-consul	string	Must be set to "" or null if manual secrets are used.

NOTE: Do not override values other than the parameters mentioned in the preceding table; doing so could cause deployment to fail.

Configure Kubernetes

Configs Layout

Tenant configuration is stored in ConfigMap. Contains BDS configurations files:

```
data:
  config-.json: {}
  gvars.py: {}
```

Layout of Secrets:

Shared Secrets:

Create secrets manually using the instructions in Create Secrets.

Genesys Info Mart — Example of GIM configuration section:

```
"gimdb": {
    "db_type": "postgre",
    "driver_name": "PostgreSQL",
    "server": "BDS_CFG_GLOBALS_GIM_DB_HOST_PLACEHOLDER",
    "port": 5432,
    "database": "BDS_CFG_GLOBALS_GIM_DB_NAME_PLACEHOLDER",
    "username": "BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER",
    "password": "BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER"
}
```

Install the following PLACEHOLDERs with values as secrets:

`BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER`

`BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER`

`BDS_CFG_GLOBALS_GIM_DB_HOST_PLACEHOLDER`

– add to secrets or replace with value in tenant config file

`BDS_CFG_GLOBALS_GIM_DB_NAME_PLACEHOLDER`

– add to secrets or replace with value in tenant config file

GVP — Example of GVP configuration section :

```
"gvp": {
    "gvp_primary_rs_name": "GVP",
    "db_type": "sql_server",
    "driver_name": "FreeTDS",
    "server": "BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_HOST_PLACEHOLDER",
    "port": 1433,
    "database": "BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_NAME_PLACEHOLDER",
    "username": "BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER",
    "password": "BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER"
}
```

Install the following PLACEHOLDERs with values as secrets:

`BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER`

`BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER`

`BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_HOST_PLACEHOLDER`

– add to secrets or replace with value in tenant config file

`BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_NAME_PLACEHOLDER`

– add to secrets or replace with value in tenant config file

Consul — Example of Consul configuration section :

```
"consul": {
    "token": "BDS_CFG_CONSUL_TOKEN_PLACEHOLDER",
    "url_api": "BDS_CFG_CONSUL_URL_API_PLACEHOLDER"
}
```

> **Important**
>
> Beginning with 100.0.003.0009 release, the **url_api** parameter must include the api version and key value endpoint.

> For example: https://consul.genesys.svc.cluster.local:8501/v1/kv

Install the following PLACEHOLDERs with values as secrets:

```
BDS_CFG_CONSUL_TOKEN_PLACEHOLDER
```

Manual secrets:

Secrets created manually. Values pulled from shared key-vault or added manually.

Assigned to POD as environment variables.

GWS

Client ID and Secret to access AUTH service.

Values assigned manually.

```
BDS_CFG_BDS_DEV_GWS_CLIENTID_PLACEHOLDER
BDS_CFG_BDS_DEV_GWS_CLIENT_SECRET_PLACEHOLDER
```

Configuration example:

```
"gws": {
   "host": "BDS_CFG_GWS_HOST_PLACEHOLDER",
    "auth_host": "BDS_CFG_GWS_AUTH_HOST_PLACEHOLDER",
    "grant_type": "client_credentials",
    "client_id": "BDS_CFG_GLOBALS_GWS_USR_PLACEHOLDER",
    "client_secret": "BDS_CFG_GLOBALS_GWS_PSW_PLACEHOLDER"
}
```

SFTP

BDS automatically uploads the resulting output files at the end of job runs, to the SFTP server. You can configure the SFTP server details in the `loader_sftp` section.

An example SFTP configuration section is as follows. In the example, the values for `hostname` and `hostkey` corresponds to the Genesys SFTP server. You can use the same values to configure the Genesys SFTP server for uploading BDS files.

```
"loader_sftp": {
        "hostname": "BDS_CFG_SFTP_HOST_PLACEHOLDER",
        "hostkey": "ssh-rsa SHA256:gT7Aa37+yTnd6mwv6Nl01E44u2o2TYxlL/iPgA2T2wc",
        "path": "BDS_CFG_SFTP_PATH_PLACEHOLDER",
        "username": "BDS_CFG_LEGACY_GLOBALS_SFTP_USR_PLACEHOLDER",
        "password": "BDS_CFG_LEGACY_GLOBALS_SFTP_PSW_PLACEHOLDER"
}
```

GVP

Manually obtained GVP secrets to get BD DB access from secondary region (used if secondary region exists).

```
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_USR_PLACEHOLDER
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_PSW_PLACEHOLDER
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_HOST_PLACEHOLDER
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_NAME_PLACEHOLDER
```

Configure security

Example of SFTP configuration section:

```
"loader_sftp": {
        "hostname": "BDS_CFG_SFTP_HOST_PLACEHOLDER",
        "hostkey": "ssh-rsa SHA256:gT7Aa37+yTnd6mwv6Nl01E44u2o2TYxlL/iPgA2T2wc",
        "path": "BDS_CFG_SFTP_PATH_PLACEHOLDER",
        "username": "BDS_CFG_LEGACY_GLOBALS_SFTP_USR_PLACEHOLDER",
        "password": "BDS_CFG_LEGACY_GLOBALS_SFTP_PSW_PLACEHOLDER"
}
```

Pod security policy:

By default, BDS defines a user/group for running the process in the POD, as follows:

```
securityContext:
    # Containers should run as genesys user and cannot use elevated permissions
   runAsNonRoot: true
   runAsUser: 500
   runAsGroup: 500
   fsGroup: 500
```

If you want to use arbitrary UIDs in your OpenShift deployment, you must override the **securityContext** settings as shown in the following code, so that you do not define any specific IDs.

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

Provision BDS

Contents

- Administrator

Learn how to provision Billing Data Service (BDS).

Related documentation:

-

Tenant Provisioning

This page describes the following provisioning procedures:

- Create configmap
- Create secrets
- Create StorageClass, Persistent Volume, and Persistent Volume Claims
- Create watermarks

Create configmap

1. Download bds-config archive:

   ```
   pureengage-helm-production-local.jfrog.io/bds-config-100.0.003+0029-pe
   ```

   The archive contains two tenants examples in the **result** folder:

   - multiregion configuration file: `config-t100-100.json`
   - single-region configuration file: `config-t101-101.json`

   This procedure demonstrates a multi-region scenario.

2. Extract archive:

   ```
   developer@docker:/media/sf_shared/OpenShift$ tar xvzf bds-config-100.0.003+0029-pe.tgz
   bds-config/Chart.yaml
   bds-config/values.yaml
   bds-config/templates/bds-configmap.yml
   bds-config/result/config-t100-100.json
   bds-config/result/config-t101-101.json
   bds-config/result/gvars.py
   bds-config/result/main.json
   ```

3. Update **bds-config/result/config-t100-100.json**:
   1. In the "globals" section:

1. Add "gws" section:

```
"gws": {
            "host": "BDS_CFG_GWS_HOST_PLACEHOLDER",
            "auth_host": "BDS_CFG_GWS_AUTH_HOST_PLACEHOLDER",
            "grant_type": "client_credentials",
            "client_id": "BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER",
            "client_secret":
 "BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER"
      }
```

2. Add/modify "SourceId": "MultiCloudPE". If the `modeValue = MULTICLOUD_PE`, the `SourceId` value is by default set as `MultiCloudPE` in the `globals` section.

2. In **tenants//**, populate **contact_center_id** with correct value. You can get this value by running a command specified in Get contact center ID from GWS.
   For example: **"contact_center_id": "e01f1720-c1ec-11eb-a4b5-53da51249f17"**

3. Configure your Genesys Info Mart endpoint by replacing corresponding placeholders. This step is optional, but if you do not complete it, then you must populate placeholders' values in bds-manual-secrets (see Create secrets)
   Example:

```
"gimdb": {
    "db_type": "postgre",
    "driver_name": "PostgreSQL",
    "server": "BDS_CFG_GLOBALS_GIM_DB_HOST_PLACEHOLDER",
    "port": 5432,
    "database": "BDS_CFG_GLOBALS_GIM_DB_NAME_PLACEHOLDER",
    "username": "BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER",
    "password": "BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER"
}
```

   Where,

   - **db_type** supports `sql_server`, `postgre`, and `oracle` databases
   - **driver_name** must be `FreeTDS`, `PostgreSQL`, and `OracleODBC-12.1` correspondingly for **sql_server, postgre**, and **oracle** databases.
   - **server** supports both FQDN and IP values.

4. Configure your GVP endpoint in each region of your tenant by replacing corresponding placeholders. This step is optional, but if you do not complete it, then you must populate placeholders' values in bds-manual-secrets (see Create secrets).
   Example:

```
"gvp": {
    "gvp_primary_rs_name": "GVP",
    "db_type": "sql_server",
    "driver_name": "FreeTDS",
    "server": "BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_HOST_PLACEHOLDER",
    "port": 1433,
    "database": "BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_NAME_PLACEHOLDER",
    "username": "BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_USR_PLACEHOLDER",
    "password": "BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_PSW_PLACEHOLDER"
}
```

5. Add Consul endpoints in each region of your tenant (For example: **/tenants//regions/us/ash/**):

```
"consul": {
                              "url_api": "BDS_CFG_CONSUL_TOKEN_PLACEHOLDER",
                              "token": "BDS_CFG_CONSUL_GLOBALS_TOKEN_PLACEHOLDER"
```

```
}
```

6. Configure the **production_date** according to your business needs. Genesys recommends to set the production date as one day prior to the date in which you want to go-live.
   For example, if you want to go-live on January 25, 2022, then you must set the production date as at least one day prior to January 25, that is, January 24, 2022 or earlier dates.

   "production_date": "2022-01-24"

4. Execute the following command to create the configmap file:

```
helm install --debug bds-config bds-config  --version=100.0.003+0029-pe -n bds
```

5. Manually create extract and transform folders on shared storage which is mounted by PVC.

Create secrets

This section describes how to create manual secrets.

> **Important**
> BDS supports multi-region deployment. If you are configuring using the multi-regional configuration file, you must define the multi-region specific placeholders in the configuration file and secret key-value pairs in the secret file.

OpenShift

1. Create file with secrets (change all * to your real variables):

```
cat  ./bds-manual-secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: bds-manual-secrets
  namespace: bds
type: Opaque
data:
  BDS_CFG_GLOBALS_GVP_DB_USER_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_USER_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_GVP_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_GVP_USER_PLACEHOLDER:
  BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER:
  BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER:
  BDS_CFG_CONSUL_GLOBALS_TOKEN_PLACEHOLDER:
  BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER:
  BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER:
  BDS_CFG_CONSUL_TOKEN_PLACEHOLDER:
  BDS_CFG_BDS_DEV_GWS_CLIENTID_PLACEHOLDER:
```

```
  BDS_CFG_BDS_DEV_GWS_CLIENT_SECRET_PLACEHOLDER:
stringData:
  BDS_CFG_GLOBALS_GVP_DB_USER_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_USER_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_GVP_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_GVP_USER_PLACEHOLDER: **
  BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER: **
  BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER: **
  BDS_CFG_CONSUL_GLOBALS_TOKEN_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER: **
  BDS_CFG_CONSUL_TOKEN_PLACEHOLDER: **
  BDS_CFG_BDS_DEV_GWS_CLIENTID_PLACEHOLDER: **
  BDS_CFG_BDS_DEV_GWS_CLIENT_SECRET_PLACEHOLDER: **
EOF
```

> **Important**
>
> In the secrets manifest file, specify the base64 encoded secrets values in the **data** section and the plain text values in the **stringData** section, which will be converted into base64 string after the file is created. The secrets values are validated either from the data or stringData sections or both. However, in Kubernetes, you cannot leave any values as blank in the secret manifest file.

2. Install this secret to OpenShift by executing the following command:

```
developer@docker:~$ oc create -f bds-manual-secrets.yaml
secret/bds-manual-secrets created
```

3. Since manual secrets are used, you must override **bdsApp.secrets.gim.mounts.name**, **bdsApp.secrets.gvp.mounts.name**, **bdsApp.secrets.consul.mounts.name** to **" "**. For more details on these parameters, refer Configure BDS.

Kubernetes

1. Create file with secrets (change all * to your real variables).

```
cat  ./bds-manual-secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: bds-manual-secrets
  namespace: bds
type: Opaque
data:
  BDS_CFG_GLOBALS_GVP_DB_USER_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_USER_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_GVP_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_CME_GVP_USER_PLACEHOLDER:
  BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER:
  BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER:
  BDS_CFG_CONSUL_GLOBALS_TOKEN_PLACEHOLDER:
  BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER:
```

```
  BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER:
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER:
  BDS_CFG_CONSUL_TOKEN_PLACEHOLDER:
  BDS_CFG_BDS_DEV_GWS_CLIENTID_PLACEHOLDER:
  BDS_CFG_BDS_DEV_GWS_CLIENT_SECRET_PLACEHOLDER:
stringData:
  BDS_CFG_GLOBALS_GVP_DB_USER_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_USER_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_GVP_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_GVP_USER_PLACEHOLDER: **
  BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER: **
  BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER: **
  BDS_CFG_CONSUL_GLOBALS_TOKEN_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_USR_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PL_WESTUS2_PSW_PLACEHOLDER: **
  BDS_CFG_CONSUL_TOKEN_PLACEHOLDER: **
  BDS_CFG_BDS_DEV_GWS_CLIENTID_PLACEHOLDER: **
  BDS_CFG_BDS_DEV_GWS_CLIENT_SECRET_PLACEHOLDER: **
EOF
```

> ### Important
>
> In the secrets manifest file, specify the base64 encoded secrets values in the **data** section and the plain text values in the **stringData** section, which will be converted into base64 string after the file is created. The secrets values are validated either from the data or stringData sections or both. However, in Kubernetes, you cannot leave any values as blank in the secret manifest file.

2. Install this secret to the Kubernetes cluster by executing the following command:

```
kubectl create -f bds-manual-secrets.yaml
secret/bds-manual-secrets created
```

3. Since manual secrets are used, you must override `bdsApp.secrets.gim.mounts.name`, `bdsApp.secrets.gvp.mounts.name`, and `bdsApp.secrets.consul.mounts.name` to `" "`. For more details on these parameters, refer Configure BDS.

GKE

1. Create file with secrets (change all * to your real variables):

```
cat  ./bds-manual-secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: bds-manual-secrets
  namespace: bds
type: Opaque
stringData:
  BDS_CFG_GLOBALS_GVP_DB_USER_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_GVP_DB_NAME_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_PSW_PLACEHOLDER: **
  BDS_CFG_GLOBALS_CME_USER_PLACEHOLDER: **
```

```
BDS_CFG_GLOBALS_CME_GVP_PSW_PLACEHOLDER: **
BDS_CFG_GLOBALS_CME_GVP_USER_PLACEHOLDER: **
BDS_CFG_BDS_GLOBALS_GWS_CLIENTID_PLACEHOLDER: **
BDS_CFG_BDS_GLOBALS_GWS_CLIENT_SECRET_PLACEHOLDER: **
BDS_CFG_GLOBALS_GIM_DB_HOST_PLACEHOLDER: **
BDS_CFG_GLOBALS_GIM_DB_NAME_PLACEHOLDER: **
BDS_CFG_GLOBALS_GIM_DB_USR_PLACEHOLDER: **
BDS_CFG_GLOBALS_GIM_DB_PSW_PLACEHOLDER: **
BDS_CFG_CONTACT_CENTER_ID_PLACEHOLDER: **
BDS_CFG_SOURCEID_PLACEHOLDER: **
BDS_CFG_CS_CONSUL_PLACEHOLDER: **
BDS_CFG_GWS_HOST_PLACEHOLDER: **
BDS_CFG_GWS_AUTH_HOST_PLACEHOLDER: **
BDS_CFG_BDS_GWS_CLIENTID_PLACEHOLDER: **
BDS_CFG_BDS_GWS_CLIENT_SECRET_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_HOST_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_NAME_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_USR_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_PL_EASTUS2_PSW_PLACEHOLDER: **
BDS_CFG_CONSUL_TOKEN_PLACEHOLDER: **
BDS_CFG_CONSUL_URL_API_PLACEHOLDER: **
BDS_CFG_GIR_ES_PLACEHOLDER: **
BDS_CFG_CONSUL_URL_API_REGION_PLACEHOLDER: **
BDS_CFG_CONSUL_WESTUS2_TOKEN_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_HOST_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_NAME_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_PSW_PLACEHOLDER: **
BDS_CFG_GLOBALS_GVP_DB_WESTUS2_USR_PLACEHOLDER: **
BDS_CFG_LEGACY_GLOBALS_SFTP_PSW_PLACEHOLDER: **
BDS_CFG_LEGACY_GLOBALS_SFTP_USR_PLACEHOLDER: **
BDS_CFG_SFTP_HOST_PLACEHOLDER: **
BDS_CFG_SFTP_PATH_PLACEHOLDER: **
```

2. Install this secret to your GKE2-2 cluster by executing the following command:

```
kubectl create -f bds-manual-secrets.yaml
```

3. Since manual secrets are used, you must override `bdsApp.secrets.gim.mounts.name`, `bdsApp.secrets.gvp.mounts.name`, `bdsApp.secrets.consul.mounts.name` to `" "`. For more details on these parameters, refer Configure BDS.

Create StorageClass, Persistent Volume, and Persistent Volume Claim

OpenShift

1. Create Storage Class (only if your configuration requires it):
    1. Create file **bds-storageclass.yaml** by referring the following sample code:

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name:
parameters:
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner:
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

2. Apply this yaml:

```
oc create -f bds-storageclass.yaml
```

2. Create Persistent Volume (only if your configuration requires it):
   1. Create file **bds-pv.yaml** by referring the following sample code:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name:
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  persistentVolumeReclaimPolicy: Retain
  storageClassName:
  volumeMode: Filesystem
```

   2. Apply this yaml:

```
oc create -f bds-pv.yaml
```

3. Create Persistent Volume Claim:
   1. Create file **bds-pvc.yaml** by referring the following sample code:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name:
  namespace:
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName:
  volumeMode: Filesystem
```

   2. Apply this yaml:

```
oc create -f bds-pvc.yaml
```

Kubernetes

1. Create Persistent Volume (only if your configuration requires it).
   1. Create file **pv.yaml** by referring the following sample code:

```
cat  ./pv.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: bds-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/genesys/data"
EOF
```

2. Apply this yaml:

```
kubectl create –f pv.yaml
```

2. Create Persistent Volume Claim.
   1. Create file **pvc.yaml** by referring the following sample code:

```
cat  ./pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: bds-pvc
  namespace: bds
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
EOF
```

   2. Apply this yaml:

```
kubectl create –f pvc.yaml
```

GKE

1. Create StorageClass and Persistent Volume, (only if your configuration requires it). If they already exists, then skip to the next step.
2. Create Persistent Volume Claim.
3. Create file **bds-pvc.yaml** by referring the following sample code:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: bds-pvc
namespace: bds
spec:
accessModes:
- ReadWriteMany
resources:
requests:
```

```
        storage: 5Gi
        storageClassName: nfs-client
        volumeMode: Filesystem
```

4. Apply **bds-pvc.yaml**, by executing the following command:.

```
kubectl apply -f  bds-pvc.yaml -n bds
```

Create watermarks on persistent volume

The extract, transform, and load processes in BDS relies on watermark dates. Typically, the watermark date is either the agreed go-live date, or a date that is one day earlier than the date when data exists in all relevant data sources.

Watermark dates are automatically created for BDS persistence volume. You can also manually update the watermark in scenarios where you want to reprocess or catch up the data.

> ### Important
> Before creating a watermark file for BDS, a persistence volume must be already provisioned.

To **automatically** create watermarks, run BDS. Running BDS creates the watermark file and updates all the watermark related fields such as `extract_watermark, transform_watermark,` and `load_watermark` with the production date. Note that the production date is configured as part of Tenant provisioning.

> ### Important
> Versions prior to 100.0.002.0020 require watermarks file to be created manually. If you are upgrading from previous versions to 100.0.002.0020 or higher, and if the watermark file doesn't exist, then running BDS automatically creates the watermark file and updates the watermark fields with the production date. For BDS to process the data on the go-live date, make sure you set the production_date as at least one day prior to the go-live date.

To **manually** create watermarks, perform the following steps:

1. Create the pod yaml by referring the following sample code.

```
cat  ./pv-static.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-pv
spec:
```

```
  volumes:
    - name: pv-test
      persistentVolumeClaim:
        claimName: bds-pvc
  containers:
    - name: test
      image: ubuntu
      command: [ "/bin/sh" ]
      args: [ "-c", "while true; do echo hello; sleep 10;done" ]
      volumeMounts:
        - mountPath: "/tmp"
          name: pv-test
EOF
```

2. Execute the following commands to verify the status of the pod created, create the **watermarks** folder, and create the tenant json file.

```
developer@docker:~/OpenShift$ kubectl apply -f pv-static.yaml -n bds
pod/static-pv created
developer@docker:~/OpenShift$ kubectl get pod static-pv -n bds
NAME        READY   STATUS    RESTARTS   AGE
static-pv   1/1     Running   0          68s
developer@docker:~/OpenShift$ kubectl exec  -n bds -it static-pv bash
1000670000@static-pv:/$ cd /tmp
1000670000@static-pv:/$ mkdir watermarks
1000670000@static-pv:/tmp$ cd watermarks/
1000670000@static-pv:/tmp/watermarks$  echo -e "{ \n\t\"extract_watermark\": \"YYYY-
MM-DD\",\n\t\"transform_watermark\": \"YYYY-MM-DD\",\n\t\"load_watermark\": \"YYYY-
MM-DD\"\n}" > .json
```

Where,

- **extract_watermark** - The last successful extraction date in YYYY-MM-DD format, should be greater than or equal to the production date.
- **transform_watermark** - The last successful transformation date in YYYY-MM-DD format, should be less than or equal to the extract watermark, and greater than the production date.
- **load_watermark** - The last successful load date in YYYY-MM-DD format, should be less than or equal to the transform watermark, and greater than the production date.
- - The name of the tenant as it is configured in the Config.json file for BDS. A watermark file will be created in the **json** format for the tenant with the name specified here, for example, **MultiRegionExampleTenant.json**.

3. Execute the following command to verify the watermark values created in the tenant's json file. You will see a result similar to the following example. In this example, **MultiRegionExampleTenant.json** is the tenant file that contains the extract, transform, and load watermark values for the tenant.

```
cd /tmp/watermarks/
MultiRegionExampleTenant.json
{
"extract_watermark": "2021-06-27",
"transform_watermark": "2021-06-27",
"load_watermark": "2021-06-27"
}
```

4. Exit and remove pod.

Deploy BDS

Contents

Learn how to deploy Billing Data Service (BDS).

Related documentation:

-

Prepare your environment

OpenShift

1. In the OpenShift UI, on the menu where your user name appears, select **Copy Login Command**.
   A token and command for login appears.
2. Log in to cluster with OC CLI:

   ```
   oc login --token=$token --server=https://api.vce-c0.eps.genesys.com:6443
   ```
3. Check if the cluster is running:

   ```
   oc get clusterversion
   ```
4. If the project doesn't already exist, create new project:

   ```
   oc new-project bds
   ```
5. Create Secret for accessing the JFrog registry and map it to the genesys account:

   ```
   oc create secret docker-registry mycred --docker-server=pureengage-docker-
    staging.jfrog.io --docker-username= --docker-password= --docker-email= -n bds
    oc secrets link genesys mycred --for=pull -n bds
   ```
6. Create StorageClass, Persistent Volume, and Persistent Volume Claim by referring to Create StorageClass, Persistent Volume, and Persistent Volume Claim > OpenShift Storage in Provision BDS.

Kubernetes

1. Log in to the Kubernetes cluster.
2. Check if the cluster is running:

   ```
   kubectl cluster-info
   ```
3. If the namespace for BDS doesn't already exist, create a new namespace:

   ```
   kubectl create namespace bds
   ```
4. Create Secret for accessing the Docker private registry if needed. Use the following command:

   ```
   kubectl create secret generic docker-cred \
   ```

```
--from-file=.dockerconfigjson="/home/${USER}/.docker/config.json" \
--type=kubernetes.io/dockerconfigjson --namespace=bds
```

If the Docker registry requires authentication, name of the Docker secret must be passed to deployment as `bdsApp.image.imagePullSecrets` parameter.

5. If you are deploying in a production environment,
   - Create StorageClass, Persistent Volume, and Persistent Volume Claim by referring to Create StorageClass, Persistent Volume, and Persistent Volume Claim > OpenShift Storage in Provision BDS. For the Kubernetes production environment, you can use the yaml manifests similar to OpenShift storage manifests.

> **Important**
>
> In production environments, the actual storage class and storage method could be different, for example, network storage or cloud storage. Hence, modify the sample code with appropriate storage values that is suitable for the environment you deploy.

6. If you are deploying in a lab or testing environment,
   1. Create a folder on every cluster worker node that schedules BDS cronjob. Use the following command:

```
sudo mkdir –p /genesys/data
sudo chmod 777 –R /genesys/data
```

   2. Create yaml manifests for Persistent Volume and Persistent Volume Claim. Use the sample codes from Create StorageClass, Persistent Volume, and Persistent Volume Claim > Kubernetes Storage in Provision BDS.
7. Override the `bdsApp.volumes.pvc.claim` chart parameter with the PVC name created in the previous step, for example, bds-pvc.

Google Kubernetes Engine (GKE)

1. Log in to the GKE cluster.
2. If the namespace for BDS doesn't already exist, create a new namespace:

```
kubectl create namespace bds
```

3. Create Secret for accessing the JFrog registry, if needed. Use the following command:

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

4. Create Persistent Volume Claim by referring to Create StorageClass, Persistent Volume, and Persistent Volume Claim > GKE in Provision BDS.

> - **Important**
>
> In production environments, the actual storage class and storage method could be different, for

> example, network storage or cloud storage. Hence, modify the sample code with appropriate storage values that is suitable for the environment you deploy.

5. Override the `bdsApp.volumes.pvc.claim` chart parameter with the PVC name created in the previous step, for example, bds-pvc.

Deploy

Prerequisites

Ensure that you have the following:

- From JFrog:
  - Helm chart package
  - BDS docker image
- The CLI is installed (Download the CLI), and **oc** copied to the folder **/usr/bin/**.

Deployment steps

OpenShift

1. Download the bds-cronjob helm charts from the JFrog repository (or receive it from premise packages). Helm charts are located here: https://pureengage.jfrog.io/artifactory/helm-stage/

   For example, the bds-cronjob-100.0.003+0029.tgz version is used for explaining the deployment.

   1. Create a folder for current deployment and extract the bds-cronjob-100.0.003+0029.tgz:

      ```
      developer@docker:~/ tar xvzf bds-cronjob-100.0.003+0029.tgz
      bds-cronjob/Chart.yaml
      bds-cronjob/values.yaml
      bds-cronjob/templates/_helpers.tpl
      bds-cronjob/templates/alerts.yml
      bds-cronjob/templates/run-bds.yml
      bds-cronjob/.helmignore
      ```

   2. Create **bds-manual-secrets.yaml** with secrets to access data sources like GIM and few other services like GWS. See OpenShift Secrets for a sample secrets file.
   3. Create **override.yaml** with applicable values in the bds-cronjob folder. The Default value of BDS mode type must be one of the following:
      - For 9.0.00x releases: BDS_OPERATING_MODE = MULTICLOUD
      - For 100.0.00x.xxxx and later releases: BDS_OPERATING_MODE = MULTICLOUD_PE

        For example:

        ```
        tenantName: testtenant
        ```

```
serviceAccount:
   name: genesys
bdsApp:
  secrets:
    gim:
      volumes:
        tenantID: 001
      mounts:
        name: ""
    gvp:
      mounts:
        name: ""
    consul:
      mounts:
        name: ""
  container:
    env:
      ## Description of the BDS mode type
      modeValue: MULTICLOUD
      ## URL for Prometheus PushGateway service. Set to empty string if PG is not
available.
      pgValue: "http://prometheus-
pushgateway.monitoring.svc.gke2-useast1.gcpe002.gencpe.com:9091"
  volumes:
    pvc:
      claim: bds-pvc
  image:
    tag: 100.0.003.0029

    ## Should be overriden with own values
    registry: pureengage-docker-staging.jfrog.io/
  securityContext:
    # Containers should run as genesys user and cannot use elevated permissions
    runAsGroup: 500
    runAsUser: 500
```

2. Run the following command to install CronJob:

```
helm upgrade --install --history-max 5   ./helm/bds-cronjob -f override.yaml -n bds
```

Where = bds- -

For Example:

```
developer@docker:~/OpenShift$ helm upgrade --install --history-max 5 bds-dev-testtenant ./bds-
cronjob -f override.yaml -n bds
coalesce.go:196: warning: cannot overwrite table with non table for tag (map[])
coalesce.go:196: warning: cannot overwrite table with non table for tenantID (map[])
Release "bds-dev-testtenant" has been upgraded. Happy Helming!
NAME: bds-dev-testtenant

LAST DEPLOYED: Wed May 12 13:31:53 2021
NAMESPACE: bds
STATUS: deployed

REVISION: 2
TEST SUITE: None
```

Kubernetes

1. Download the bds-cronjob helm charts from the JFrog repository (or receive it from premise packages).

Helm charts are located here: https://pureengage.jfrog.io/artifactory/helm-stage/

For example, the bds-cronjob-100.0.003+0029.tgz version is used for explaining the deployment.

1. Create a folder for current deployment and extract the bds-cronjob-100.0.003+0029.tgz:

```
developer@docker:~/ tar xvzf bds-cronjob-100.0.003+0029.tgz
bds-cronjob/Chart.yaml
bds-cronjob/values.yaml
bds-cronjob/templates/_helpers.tpl
bds-cronjob/templates/alerts.yml
bds-cronjob/templates/run-bds.yml
bds-cronjob/.helmignore
```

2. Create **bds-manual-secrets.yaml** with secrets to access data sources like GIM and few other services like GWS. See Kubernetes Secrets for a sample secrets file.
3. Create **override.yaml** with applicable values in the bds-cronjob folder. The Default value of BDS mode type must be one of the following:
   - For 9.0.x versions: BDS_OPERATING_MODE = MULTICLOUD
   - For 100.0.00x.xxxx versions and later releases: BDS_OPERATING_MODE = MULTICLOUD_PE

   For example:

```
tenantName: testtenant
serviceAccount:
   name: genesys
bdsApp:
  secrets:
    gim:
      volumes:
        tenantID: 001
      mounts:
        name: ""
    gvp:
      mounts:
        name: ""
    consul:
      mounts:
        name: ""
  container:
    env:
      ## Description of the BDS mode type
      modeValue: MULTICLOUD_PE
      ## URL for Prometheus PushGateway service. Set to empty string if PG is not
available.
      pgValue: "http://prometheus-
pushgateway.monitoring.svc.gke2-useast1.gcpe002.gencpe.com:9091"
  volumes:
    pvc:
      claim: bds-pvc
  image:
    tag: 100.0.003.0029

    ## Should be overriden with own values
    registry: pureengage-docker-staging.jfrog.io/
  securityContext:
    # Containers should run as genesys user and cannot use elevated permissions
    runAsGroup: 500
    runAsUser: 500
```

Make sure you modify the service account name that your organization created or default to Kubernetes value (default k8s value).

2. Run the following command to install CronJob:

```
helm upgrade --install --history-max 5   ./helm/bds-cronjob -f override.yaml -n bds
```

Where = bds- -

GKE

1. Download the bds-cronjob helm charts from the JFrog repository (or receive it from premise packages). Helm charts are located here: https://pureengage.jfrog.io/artifactory/helm-stage/

   For example, the bds-cronjob-100.0.003+0029.tgz version is used for explaining the deployment.

2. Create a folder for current deployment and extract the bds-cronjob-100.0.003+0029.tgz:

```
developer@docker:~/ tar xvzf bds-cronjob-100.0.003+0029.tgz
bds-cronjob/Chart.yaml
bds-cronjob/values.yaml
bds-cronjob/templates/_helpers.tpl
bds-cronjob/templates/alerts.yml
bds-cronjob/templates/run-bds.yml
bds-cronjob/.helmignore
```

3. Deploy the Configuration file manually by running the following command:

```
helm install --debug bds-config bds-config  --version=100.0.003+0029-pe -n bds
```

4. Create **bds-manual-secrets.yaml** with secrets to access data sources like GIM and few other services like GWS. See GKE Secrets for a sample secrets file.
5. Create **override.yaml** with applicable values in the bds-cronjob folder. The Default value of BDS mode type must be one of the following:
   - For 9.0.x versions: BDS_OPERATING_MODE = MULTICLOUD
   - For 100.0.00x.xxxx versions and later releases: BDS_OPERATING_MODE = MULTICLOUD_PE

   For example:

```
tenantName: t100-100
serviceAccount:
   name: default
bdsApp:
  secrets:
    gim:
      volumes:
        tenantID: 001
      mounts:
        name: ""
    gvp:
      mounts:
        name: ""
    consul:
      mounts:
        name: ""
  container:
    env:
      ## Description of the BDS mode type
```

```
      modeValue: MULTICLOUD_PE
      ## URL for Prometheus PushGateway service. Set to empty string if PG is not
available.
      pgValue: "http://prometheus-
pushgateway.monitoring.svc.gke2-useast1.gcpe002.gencpe.com:9091"
  volumes:
    pvc:
      claim: bds-pvc
  image:
    tag: "100.0.003.0029"
    ## Should be overriden with own values
    registry: pureengage-docker-staging.jfrog.io/
    pullSecrets:
      name: "pullsecret"
```

Make sure you modify the service account name that your organization created or default to Kubernetes value (default k8s value).

6. Run the following command to install CronJob:

```
helm --install -n bds bds-cronjob https://pureengage.jfrog.io/artifactory/helm-stage/
bds-cronjob-100.0.003+0029.tgz  -f override_values.yaml –username  --password
```

Validate the deployment

To validate the deployment, run the following commands to trigger the CronJob manually, and check pod Events:

```
kubectl -n bds create job --from=cronjob/bds-mytenant bds-run
kubectl -n bds get pod
NAME READY STATUS RESTARTS AGE
bds-run-98771 0/1 Completed 0 100s

kubectl -n bds describe pod bds-run-98771
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  15s   default-scheduler  Successfully assigned bds/bds-run-npqc6 to
minikube
  Normal  Pulling    14s   kubelet            Pulling image "genesys-
local.jfa.genesyslab.com:443/cloudbilling/scripts:latest"
  Normal  Pulled     11s   kubelet            Successfully pulled image "genesys-
local.jfa.genesyslab.com:443/cloudbilling/scripts:latest" in 2.829413381s
  Normal  Created    11s   kubelet            Created container scripts-mytenant
  Normal  Started    10s   kubelet            Started container scripts-mytenant
```

Upgrade, rollback, or uninstall BDS

Contents

Learn how to upgrade, rollback, or uninstall Billing Data Service (BDS).

Related documentation:

-

Upgrade BDS

Before upgrading, ensure that you have a current backup of configuration files for configmaps.

To upgrade to a new release of BDS, deploy the new release. BDS provides a tool to automatically migrate configuration settings to the new release. If the new release introduces new parameters, or has changes to existing parameters, BDS prompts you to populate them.

Migrate configuration

This section describes how to use the BDS migration tool to patch your configuration file in preparation for upgrading BDS.

Prerequisites: Ensure that the **cfg_file** parameter in **gvars.py** contains the correct path to the configuration file to be patched:

```
docker run -t -v $(pwd)/etc:/genesys/etc -i genesys-local.jfa.genesyslab.com:443/cloudbilling/
scripts:latest /bin/bash
```

1. Prepare environment:

   ```
   . /home/genesys/.bash_profile
   export BDS_OPERATING_MODE=MULTICLOUD_PE
   ```

2. Run configuration wizard:

   ```
   python3 brsctl.py migrate

   Starting configuration migration
   config.py:init: Trying to read configuration from /genesys/brs/config.json
   2021-04-28 11:54:50,290 DEBUG MainThread config.py:init: Read configuration from
   /genesys/brs/config.json
   Found configuration 100.0.000.0001, BDS 10.0.000.01. Searching for required
   migration patches...
   Patches to be applied: 0.0.0.0 to 100.0.000.0002, 100.0.000.0002 to 100.0.000.0003
   Proceed? [y/N]: y
   Current configuration backup file '/genesys/brs/config.json.100.0.000.0001.backup'
   and gvars '/genesys/brs/gvars.py.100.0.000.0001.backup' are created
   Applying patch 0.0.0.0 to 100.0.000.0002 ...
           ------> OK
   Applying patch 100.0.000.0002 to 100.0.000.0003 ...
           ------> OK
   Migration process completed successfully. The configuration has been upgraded to
   100.0.000.03
   ```

3. Refresh the configmap as described in Provision BDS: Create configmap.

Rollback BDS

To roll back BDS to an earlier release, run the deployment process for the desired release. To roll back the configuration, use the backup files you created before upgrade.

Uninstall BDS

To uninstall, execute the command **helm -n uninstall** .

Observability in Billing Data Service

Contents

Learn about the logs, metrics, and alerts you should monitor for Billing Data Service.

Related documentation:

-

Monitoring

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

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

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

- Billing Data Service metrics

See also System metrics.

Enable monitoring

Billing Data Service works as a short living cronjob, so it uses Pushgateway for providing metrics. To enable pushing metrics, find the following Helm values as an example:

```
bdsApp:
  container:
     env:
      pgValue: http://prometheus-pushgateway.monitoring.svc.cluster.local:9091
```

Service	CRD or annotations?	Port	Endpoint/ Selector	Metrics update interval
Billing Data Service	n/a	n/a	n/a	n/a

Configure metrics

The metrics that are exposed by Billing Data Service are available by default. No further configuration is required in order to define or expose these metrics. You cannot define your own custom metrics.

Alerting

Private edition services define a number of alerts based on Prometheus metrics thresholds.

> **Important**
> While you can use general third-party functionality to create rules to trigger alerts based on metrics values you specify, private edition does not enable you to create custom alerts, and Genesys does not provide support for custom alerting.

For descriptions of available Billing Data Service alerts, see:

- Billing Data Service alerts

Configure alerts

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

The alerts are defined as **PrometheusRule** objects in a **prometheus-rule.yaml** file in the Helm charts. As described above, Billing Data Service does not support customizing the alerts or defining additional **PrometheusRule** objects to create alerts based on the service-provided metrics.

Logging

BDS writes all logs to the standard output.

Viewing logs

1. Execute the following command to get the pod name:

   ```
   kubectl -n bds get po
   NAME                                READY   STATUS      RESTARTS   AGE
    bds-t2023-2023-1624633200-w8nh5   0/1     Completed   0          34m
   ```

   Where the NAME string is composed of the following elements: bds-t---
2. Execute the following command to check the logs:

   ```
   kubectl logs bds-pod-
   ```

   Where is the pod name.

For example:

```
kubectl logs bds-pod-wn8bx
```

Setting the log level

Optionally, you can set the log level using the BDS_LOG_LEVEL environment variable. Acceptable values are:

```
CRITICAL, 50, ERROR, 40, WARNING, 30, INFO, 20, DEBUG, 10,NOTSET, 0.
```

If you do not set a value, BDS reads the default value from **gvars.py**. If **gvars.py** contains no value, BDS uses the default value (DEBUG).

BDS metrics and alerts

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

Contents

Service	CRD or annotations?	Port	Endpoint/Selector	Metrics update interval
BDS	n/a	n/a	n/a	n/a
	Note: As a serverless component, BDS is run via a Kubernetes CronJob. By default, the job runs every 12 hours (twice a day) and pushes information into the Prometheus Pushgateway.			

See details about:

- BDS metrics
- BDS alerts

Metrics

Billing Data Service (BDS) exposes few metrics through Prometheus Pushgateway for monitoring BDS performance and containers. Note that the service-monitoring metrics are distinct from the metrics BDS provides to monitor contact center activity, which are described in the Billing Data Server User's Guide.

The following system metrics are likely to be particularly useful.

- kube_pod_container_status_restarts_total
- kube_job_status_start_time
- kube_job_status_failed

For information about standard system metrics, see System metrics.

Other than the system metrics, BDS provides the following service specific metrics for monitoring and alerting purposes:

Metric and description	Metric details	Indicator of
bds_pod_processing_start The time in which the BDS cron job has started its process.	**Unit:** Second **Type:** Gauge **Label:** **Sample value:** 1638529175	
bds_pod_processing_end The time in which the BDS cron job has ended its process.	**Unit:** Second **Type:** Gauge **Label:** **Sample value:** 1638538924	
bds_processing_exit_code Exit code indicating whether the cron job has successfully started. Returns 0 if the cron job has started successfully, else returns a numeric value.	**Unit:** **Type:** Gauge **Label:** **Sample value:** 0	

Alerts

Billing Data Service does not define any alerts by default in the Helm charts. The following table indicates the sample alerts configuration that you can create using the supported metrics.

The following alerts are defined for BDS.

Alert	Severity	Description	Based on	Threshold
BDS-ContainerRestartCount		The container in which BDS did not start or no longer responsive.	kube_pod_container_status_restarts_total	>0
BDS-JobStartTime		The cron job that did not start in scheduled time or that was being processed for a long time.	kube_job_status_start_time	
BDS-JobFailStatus		Cron job failed status.	kube_job_status_failed	>0