Reporting Suite

Reporting

Kubernetes - Minimum Requirements, Installation and Management Guide

Version 5.6.0

1/20/2026

Enghouse Interactive

Contents

About this Guide

Audience

This Guide is intended for the Administrator of Reporting. The Administrator must have a sufficient access level and needs to possess intermediate computer using skills in order to be able to use this document.

New in this release

This document is new. Please review it in its entirety.

Guide conventions

This Guide uses the following text formats and notation conventions.

Text format

Bold text indicates a button, field, link, option name, or similar function requiring an action.

Italicized text indicates new terms, directory paths, or references to external documents.

`Text in this font` indicates code.

Notes and cautions

Icons used throughout this Guide identify additional details or special conditions.

> **Note**
> Provides additional information or describes special circumstances.

> **Caution**
> Warns of user actions that may cause system failure or irreversible conditions.

> **Stop**
> Describes actions that you should only perform under the supervision of Enghouse Interactive Customer Support.

Legal disclaimer

This document is governed by the terms of the software license agreement and applicable contract (including addendums) entered into with Enghouse.

Contact information

To submit comments or questions about the content in this Guide, please open a case in Support.

Installation - Kubernetes

Reporting is a web application that allows multiple Enghouse products to provide reporting and data visualization features to customers. Starting from version 4.0, Reporting can be installed in cloud environments and used in multi-tenant mode. In this installation configuration, it is recommended to provide automatic monitoring and alerting of the system health check.

The following topics cover the topics relevant for the installation and management of Reporting in a Kubernetes environment.

Product requirements

Required OS

To run Reporting V4.X or newer, you can use a Ubuntu 20.04 or Ubuntu 20.04 as the OS.

Required software

- Kubernetes 1.25+
- Helm 3.13.1+
- Default storage manager (for example, Longhorn)
- NGINX-Ingress

Required servers

Reporting V4.X or newer can be installed in multiple configurations, from small single node installations to multi-node cluster installations. The appropriate number of VMs and their sizing have to be defined based on the expected system load. The following table can be used as a guideline for sizing the VMs:

		CC Agents (10 agents => 1 BI viewer)							
		200				**500**			
Installation type	**Server type**	**Num**	**RAM**	**CPU**	**SSD DISK**	**Num**	**RAM**	**CPU**	**SSD DISK**
Single Server	Reporting	1	8	8	250	1	16	10	450
Cluster	K8s node	3	8	8	250	1	16	10	450

		CC Agents (10 agents => 1 BI viewer)							
		1000				**2000**			
Installation type	**Server type**	**Num**	**RAM**	**CPU**	**SSD DISK**	**Num**	**RAM**	**CPU**	**SSD DISK**
Single Server	Reporting	1	32	16	850	n.d.	n.d.	n.d.	n.d.
Cluster	K8s node	1	32	16	850	3	64	32	1650

Internet Access

Internet access is needed to install Reporting requirements (docker, docker-compose).

Kubernetes Helm chart deployment

> **Note**
> The following information refers to Reporting Version 4.X and newer.

The Enghouse Reporting Helm chart bundles all the Kubernetes manifests, default settings, and helper scripts needed to deploy the Reporting Suite as a single, versioned package, allowing repeatable installations on any OCI-compatible Helm registry and Kubernetes cluster.

Installation requirements

Before installing the chart, verify that your cluster meets the following prerequisites:

- Kubernetes 1.32+
- Helm 3.13.1+
- Default storage manager (for example, Longhorn)
- Ingress controller
  - NGINX-Ingress controller for self-managed deployment. If missing, follow official instructions to install it according to the specific Kubernertes distribution you use, such as:
    - NGINX Ingress Controller Installation
    - Ingress-NGINX Controller Instrallation Guide
    - Google Cloud Platform Deployment Guide
  - ALB ingress controller for EKS deployment
- Authentication to the helm registry

  ```
  helm registry login harbor.utils.reiteklab.eu.rd.eilab.biz
  ```

  > **Note**
  > If needed, add the insecure flag `--insecure`.

- Authentication to the Eptica registry to pull images within Kkubernetes

1. Log in to the Docker registry

   ```
   sudo docker login registry.eptica.com/enghousebi
   ```

2. Copy the Docker config to a temporary location with read permission for all users

   ```
   sudo cp /root/.docker/config.json /tmp
   sudo chmod a+r /tmp/config.json
   ```

3. Create a Kubernetes secret from Docker config

   ```
   kubectl -n <namespace> create secret generic <secret> \
   ```

```
--from-file=.dockerconfigjson=/tmp/config.json \
--type=kubernetes.io/dockerconfigjson
```

4. Clean up the temporary file

```
sudo rm /tmp/config.json
```

> **Note**
> Set the secret name in `global.registryCredentialSecret` inside your *values.yaml* file. The default value used by Reporting Suite products is `regcred`.

Installation

To install the chart with the `BI Reporting` release name using a dedicated namespace (recommended):

1. Download the `values.yaml` file.

```
helm show values oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/reporting
--version <version> > values.yaml
```

2. Customize the values.yaml file

```
vi values.yaml
```

3. If you want to use a new namespace, you can create it using kubectl client:

```
kubectl create namespace <namespace>
```

4. Run the installer

```
helm -n <namespace> install <my-reporting>
oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/reporting --version
<version> -f values.yaml
```

> **Note**
> If the certificate is not installed, you can use `--insecure-skip-tls-verify` in the previous helm commands to avoid certificate check errors.

Configuration

All runtime behavior of the Reporting Suite is controlled by the `values.yaml` file, which provides a full set of configurable parameters so operators can easily adjust the deployment to meet their security, performance, and multi-tenant needs. The tables below summarize each configurable parameter, its purpose, and the default value supplied by the chart.

Parameter	Description	Default
`global.logToFile`	Enable or disable storing logs in rolling files in addition to the	`false`

Parameter	Description	Default
	default console logs	
`global.registryCredentialSecret`	Name of the secret to be used to download Reporting Suite images from the private registry	`regcred`
`global.registryUrl`	Base URL of the Docker registry that hosts Enghouse images	`registry.eptica.com/enghousebi`
`global.contentSecurityPolicy`	Value for the `Content-Security-Policy` header	`frame-ancestors 'self'`
`global.dependencyProxyPrefix`	URL or identifier for accessing cached or mirrored dependencies instead of the original upstream source	
`logrotate.maxAge`	Number of days to retain the daily rotated Fluentd logs before deletion	`30`
`cacheCleanup.cronSchedule`	Cron expression that schedules the cache-cleanup task	`0 0 * * *`
`cacheCleanup.maxAgeHours`	Maximum age (hours) for unused data in BI Reporting cache	`48`
`cacheCleanup.serviceAccount`	Service account configuration for cacheCleanup (see ServiceAccount elements)	

Parameter	Description	Default
`cacheCleanup.resources`	A resources configuration for tenantSync (see Resources elements)	`0.2 cpu` and `256 Mi`
`miscellaneous.nodeTlsRejectUnauthorized`	Flag to reject internal unsafe communication to web service	`1`
`miscellaneous.dangerouslyDisableHostCheck`	Flag to disable host check	`false`
`sqlConnection.minProtocol`	minProtocol setting to apply when connecting reports to SQL Server	`TLSv1.2`
`sqlConnection.cipherString`	cipherString setting to apply when connecting reports to SQL Server	`DEFAULT@SECLEVEL=2`
`reportingServices.ingress.fqdn`	Full Qualified Domain Name assigned used to access Enghouse Reporting Suite via ingress (consider using wildcards to properly manage multitenant settings)	
`reportingServices.ingress.ingressClass`	IngressClass name used to create reporting services ingress (`alb` or `nginx`)	`nginx`
`reportingServices.ingress.TLSSecret`	Only for `nginx` ingress ; The secret with a TLS certificate and key for the HTTPS	`reporting-cert`

Parameter	Description	Default
	server; if missing, the certificate setting is skipped	
`reportingServices.ingress.annotations`	Provider-specific ingress annotations	`{}`
`api.replicas`	Number of API pod replicas	`1`
`api.servicePort`	Internal port used by the documentation API service	`8080`
`api.serviceAccount`	API service account configuration (see ServiceAccount elements)	
`api.resources`	API resources configuration (see Resources elements)	`0.2 cpu` and `512 Mi`
`reporting.replicas`	Number of Reporting pod replicas	`1`
`reporting.servicePort`	Internal port for the documentation Reporting service	`8080`
`reporting.useDesignerDbCache`	Flag to enable DB cache for the designer (required for multi-replica deployments)	`false`
`reporting.useViewerDbCache`	Flag to enable DB cache for viewer (required for multi-replica deployments)	`false`
`reporting.serviceAccount`	A service account configuration for Reporting (see	`{}`

Parameter	Description	Default
	ServiceAccount elements)	
`reporting.resources`	A resources configuration for Reporting (see Resources elements)	`0.5 cpu` and `1 Gi`
`web.replicas`	Number of web pod replicas	`1`
`web.servicePort`	Internal port for the documentation Web service	`8080`
`web.ssoTimeout`	Timeout (ms) for calls to the SSO identity server (applicable when SSO through OIDC is enabled)	`3500`
`web.ssoMaxRetries`	Max retry attempts for SSO calls in case of response timeout (applicable when SSO through OIDC is enabled)	`5`
`web.ssoWaitBeforeRetryMs`	Wait period (ms) between SSO retries (applicable when SSO through OIDC is enabled)	`1000`
`web.customTranslationsConfigMap`	Name of the configmap that stores custom translations file to be used in Reporting Suite web module (leave empty to use defaults)	
`web.serviceAccount`	Web service account	

Parameter	Description	Default
	configuration (see ServiceAccount elements)	
`web.resources`	Web resources configuration (see Resources elements)	`0.5 cpu` and `1 Gi`
`subscriptions.replicas`	Number of Subscription pod replicas	`1`
`subscriptions.servicePort`	Internal port for the Subscriptions service	`8080`
`subscriptions.serviceAccount`	Subscriptions service account configuration (see ServiceAccount elements)	
`subscriptions.resources`	Subscriptions resources configuration (see Resources elements)	`0.2 cpu` and `128 Mi`
`doc.enabled`	Enables deployment of internal documentation	`true`
`doc.replicas`	Number of Documentation pod replicas	`1`
`doc.servicePort`	Internal port for the Documentation service	`8080`
`doc.serviceAccount`	Documentation service account configuration (see ServiceAccount elements)	
`doc.resources`	Documentation	`0.2 cpu` and `128 Mi`

Parameter	Description	Default
	resources configuration (see Resources elements)	
`stdReportsImportTool.serviceAccount`	Service account configuration for stdReportsImportTool (see ServiceAccount elements)	
`stdReportsImportTool.resources`	Resources configuration for stdReportsImportTool (see Resources elements)	`0.2 cpu` and `128 Mi`
`translationsImportTool.serviceAccount`	Service account configuration for translationsImportTool (see ServiceAccount elements)	
`translationsImportTool.resources`	Resources configuration for translationsImportTool (see Resources elements)	`0.2 cpu` and `128 Mi`
`rabbitmq.replicas`	Number of RabbitMQ pods	`1`
`rabbitmq.adminPassword`	Password for the `admin` user to access the RabbitMQ management console	
`rabbitmq.ingress.enabled`	Enables or disables RabbitMQ via ingress	`true`
`rabbitmq.ingress.fqdn`	FQDN used to reach RabbitMQ	`rabbitmq.reporting.k8s`

Parameter	Description	Default
	when ingress is enabled	
`rabbitmq.ingress.ingressClass`	IngressClass name used to create RabbitMQ ingress (`alb` or `nginx`)	`nginx`
`rabbitmq.ingress.TLSSecret`	Only for `nginx` ingress ; The secret with a TLS certificate and key for the HTTPS server; if missing, the certificate setting is skipped	`rabbitmq-cert`
`rabbitmq.ingress.annotations`	Provider-specific ingress annotations	`{}`
`persistence.storageType`	Storage type used for persistent volumes	`longhorn`
`persistence.logsDiskSize`	Size of the persistent volume for application logs (ignored when `logToFile` is disabled)	`100Mi`
`persistence.appDbDiskSize`	Size of the persistent volume for application database	`2000Mi`
`persistence.rabbitmqDiskSize`	Size of the persistent volume for the RabbitMQ instance for internal messaging (0=no persistence)	`0`
`db.enabled`	Enables internal installation of the PostgreSQL database used for	`false`

Parameter	Description	Default
	the Reporting Suite database	
`db.storage`	Kubernetes storage requests for PostgreSQL data directory	`1Gi`
`db.replicas.instance`	Number of PostgreSQL primary pods	`1`
`db.replicas.proxy`	Number of pgBouncer pods for connection pooling	`1`
`db.backups.retentionFull`	Retention count / days for full backups	`14`
`db.backups.retentionFullType`	Retention type for full backups: `count` for number of backups or `time` for number of days	`time`
`db.backups.schedules.full`	Scheduled time to make a full backup specified in the crontab format	`0 0 * * 6`
`db.backups.schedules.differential`	Scheduled time to make a differential backup specified in the crontab format	
`db.backups.volume.storage`	Kubernetes storage requests for the pgBackRest storage	
`db.backups.storageSecretName`	Secret containing credential for S2, GCS or Azure storage	
`db.backups.s3.bucket`	Amazon S3 bucket name used for	

Parameter	Description	Default
	backups	
`db.backups.s3.endpoint`	Endpoint URL of the S3-compatible storage to be used for backups (not needed for the original Amazon S3 cloud)	
`db.backups.s3.region`	AWS region to use for Amazon and all S3-compatible storage	
`db.backups.gcs.bucket`	Google Cloud Storage bucket name used for backups	
`db.backups.azure.container`	Name of the Azure Blob Storage container for backups	
`db.monitoring.enabled`	Enables or disables monitoring Percona Distribution for PostgreSQL cluster with PMM	`false`
`mqoperator.msgTopologyOperator.service.ports.webhook`	Internal port for RabbitMQ-messaging-topology-operator-webhook	`8443`
Note The following settings are needed only if `db.enabled=false`.		
`externaldb.host`	Host name or IP of the external database used to store the Reporting Suite database	`postgres-host`
`externaldb.port`	Port of the external database used to store the Reporting	`5432`

Parameter	Description	Default
	Suite database	
`externaldb.dbname`	Database name of the external database to store the Reporting Suite database	`ersdb`
`externaldb.user`	Username of the external database to store the Reporting Suite database	`ersdb`
`externaldb.password`	Password of the external database to store the Reporting Suite database	`ersdb`
`externaldb.pgsslmode`	PostgreSQL connection SSL mode	`no-verify`

serviceAccount elements

You can optionally create and configure a dedicated service account for each deployment; the table below lists the available service account parameters and their defaults.

Parameter	Description	Default
`serviceAccount.create`	Create a dedicated service account for the component	`false`
`serviceAccount.annotations`	Provider-specific annotations for the service account	`{}`

resources elements

You can specify the minimum resource requirements and limits for each deployment; the table below lists the available service account parameters and their defaults.

Parameter	Description	Default
`resources.limits.cpu`	Maximum CPU a pod can use	deployment dependent
`resources.limits.memory`	Maximum memory a pod can use	deployment dependent
`resources.requests.cpu`	Minimum CPU required to create a pod	deployment dependent
`resources.requests.memory`	Minimum memory required to create a pod	deployment dependent

By default, the chart sets identical request and limit values in **values.yaml** to prevent pod migration across nodes. You can lower the initial resource allocation if you need to conserve capacity, or raise the limits if the defaults prove insufficient for your workload.

Kubernetes elements

> **Note**
> The following information refers to Reporting Version 4.X and newer.

This section describes the components that the EnghouseReporting Helm chart deploys.

> **Note**
> All element names receive a prefix derived from the Helm release name and the component they belong to. For example, installing the chart with `helm -n test install ers` adds the prefix `ers-` to every Kubernetes object.
>
> Objects belonging to the reporting product receive the prefix `ers-reporting-`, while RabbitMQ-related objects receive `ers-rabbitmq-`.

Subsequent sections use the base name (for example, `reporting-services`) for brevity. In a live deployment the full name of the ingress object would be `ers-reporting-reporting-services`.

Dependencies

- If the `db.enabled` parameter is set to `true` (default is `false`), the helm chart installs `pg-operator` from Percona Helm Charts. It is recommended to use your own external PostgreSQL database whenever possible.
- The rabbitmq-cluster-operator is installed to deploy RabbitMQ, which is used to manage scheduling.

Configmaps and secrets

EnghouseReporting Suite stores configuration data (for example, system cache settings and DB connection strings) in ConfigMaps and Secrets.

Ingress

reporting-services is a Kubernetes Ingress object that controls access to the reporting services. It specifies path-based routing rules to different services for the given hostname (the `fqdn` configuration field). The Ingress also handles HTTPS termination for all services. The `TLSSecret` property is required for the HTTPS setup to function.

reporting-services requires an Ingress controller to be running in the cluster. By default, it is configured to work with the Ingress-Nginx controller and uses annotations specific to that controller. Reporting Suite also supports the ALB Ingress controller for deployments on Amazon EKS.

Ingress configuration can be customized using annotations provided in the *values.yaml* file.

Reporting Suite can deploy an additional Ingress named rabbitmq, which can be enabled or disabled. This Ingress provides external access to the RabbitMQ admin UI. Internally, RabbitMQ handles subscription coordination for Reporting Suite, and in most cases it requires no manual administration.

reporting-api service

The **reporting-api** micro-service exposes internal and external APIs. To deploy this service within Kubernetes, the helm chart creates a **deployment** and a **service**, both named `reporting-api`.

reporting-cachecleanup deployment

The cache clean-up tool is a task within Enghouse Reporting that runs periodically to clean the internal cache of the application. It is not available for external use; its integration with Kubernetes consists solely of a deployment. Because it is not a critical resource for the proper behavior of Reporting Suite, multiple replicas of this tool are unnecessary. Consequently, the values file does not provide a setting to change the initial number of replicas, which defaults to 1.

Migrator

The migrator is a tool that updates the application database to the latest version. It runs as an **init-container** of the `reporting-api` pod, ensuring migration completes before the API starts.

Application database

Based on the `db.enabled` configuration parameter value, the Reporting Suite Helm chart can initialize an internal PostgreSQL instance using the Percona operator.

The recommended approach is to use an external PostgreSQL instance whenever possible, as this integrates Reporting Suite more cleanly with existing database monitoring, backup, and maintenance processes. This recommendation is especially important when deploying to a managed Kubernetes service such as Amazon EKS.

The internal PostgreSQL instance should be reserved for development and testing environments, where high-availability requirements are not a primary concern.

reporting-engine

The **reporting-engine** component is the micro-service within Reporting Suite that manages and renders reports, dashboards, and data models.

Configuration properties `reporting.useDesignerDbCache` and `reporting.useViewerDbCache` can be used to disable database cache for reporting-engine, which is enabled by default. The database cache allows `reporting.service` to run as a stateless service that can be scaled horizontally. If the database cache is

disabled, in-memory cache is used, and reporting-enging cannot be scaled. This setting is useful for reducing resource usage in development and test environments where high availability is not required.

To deploy this component, the Helm chart creates a deployment and a service named `reporting-reporting`.

Web

The **web** component is the web UI provided by Reporting Suite. To deploy this component, the Helm chart creates a deployment and a service named `reporting-web`.

Subscriptions

The **subscriptions** component is an internally used module that manages scheduled execution of subscriptions (reports delivered to users via email or other selected channels). This module is deployed by Reporting Suite helm chart as a deployment and a service named `reporting-subscriptions`.

Documentation

ERS helm chart can expose an internal HTML documentation that can be used when Reporting Suite in embedded in other Enghouse applications. The ERS documentation deployment is enabled by default, but can be disabled if documentation is provided in other ways, reducing the resources used in Kubernetes. To disable it, set the `doc.enabled` parameter to `false` in the `values.yaml` file.

When enabled, documentation is accessible here.

Documentation is deployed in Kubernetes as a service and a deployment named `reporting-doc`.

rabbitmq

RabbitMQ is used by the subscriptions module to enforce a single active pod that acts as the leader for executing subscriptions. All other pods remain in a passive state but can take over the leader role at any time to provide high availability (HA).

When deploying this component, the Helm chart creates:

- an object of type `rabbitmqCluster` named `reporting-rabbitmq`,
- a primary service also named `reporting-rabbitmq`, and
- the additional services and pods prefixed with `rabbitmq-`.

Standard report import tool

This tool allows system administrators to import standard assets (reports, dashboards, data models, and views) within Reporting Suite; the standard assets are available to all tenants configured in Reporting Suite.

When deployed through the Helm chart, this component is created in Kubernetes as a deployment with 0 replicas, named `reporting-std-reports-import-tool`. This configuration prevents the system from consuming resources for a component that is needed only intermittently, on demand by the system administrator.

To use this tool, the administrator must:

1. Increase the number of replica count to 1.
2. Perform the required import activities.
3. Reset the replica count back to 0.

The command to change the number of replicas is:

```
kubectl -n <namespace> scale deployment <instance-name>-reporting-std-reports-import-tool --replicas=1
```

Translations import tool

This tool allow system administrators to import translations files into Reporting Suite data warehouses. The translations can be the standard or custom and are imported per DWH instance. An instance can be a single, shared instance for all tenants, or more tenant-specific instances, possibly shared across multiple tenants, depending on your specific deployment.

When deployed via the Helm chart, this component is instantiated in Kubernetes as a deployment with 0 replicas, named `reporting-translations-import-tool`. This configuration prevents the system from consuming resources for a component that is needed only intermittently, on demand by the system administrator.

To use this tool, the administrator must:

1. Increase the number of replica count to 1.
2. Perform the required import activities.
3. Reset the replica count back to 0.

The command to change the number of replicas is:

```
kubectl -n <namespace> scale deployment <instance-name>-reporting-translations-import-tool --replicas=1
```

Backup and restore

> **Note**
> The following information refers to Reporting Version 4.X and newer.

Enghouse Reporting data and configurations are stored in an internal PostgreSQL database. In order to have a backup of all the application data, it is possible to manually run a backup (e.g. before a scheduled system upgrade) or schedule a periodic backup.

Since the Enghouse Reporting database is managed through Postgres Percona Operator, any backup/resore options provided by the operator is available in our deployment as well. This section includes the main and most common procedures to backup or restore the application database.

The application database contains the following data:

- Users (except SSO users)
- Groups
- Reports
  - Definition
  - Sharing
  - Schedules
  - Preferred filters
  - Bookmarks
- Data models
  - Definition
  - Sharing
- Configurations
  - SSO
  - Tenant

> **Note**
> Enghouse Reporting backup **does not** include the DWH or the call center data.

When you use the internal Keycloak instance, you can back up and restore its database, including all configured realms, clients, roles, and other settings. The sections below explain how to perform manual and scheduled backups, as well as how to restore an existing backup.

Configure backup storage

To run backup and restore activities properly, configure storage through the Helm chart by editing the `db.backups` section.

Backup repositories

The Operator uses the open-source pgBackRest utility to backup and restore.

When the Operator creates a new PostgreSQL cluster, it also creates a dedicated *pgBackRest repository* to facilitate the usage of all pgBackRest features.

Backup types

You can make the following types of backups:

- `full`: A complete backup of the entire PostgreSQL cluster.
- `differential`: Backs up only the files that have changed since the last full backup.
- `incremental`: Default. Backs up files that have changed since the last full or differential backup.

Backup storage

You can store PostgreSQL backups outside the Kubernetes cluster using one of the following options:

- Cloud storage
  - Amazon S3 (or any S3-compatible service)
  - Google Cloud Storage
  - Azure Blob Storage
- Persistent Volume attached to the pgBackRest pod

All storage settings are defined in the `db.backups` section of `values.yaml` file.

S3-compatible storage

To back up to an S3-compatible store you must gather the following information:

- **Bucket name** - Name of the S3 bucket
- **Endpoint** - URL used to reach the bucket
- **Region** - Geographic location of the bucket
- **Credentials** - Access key and secret key (stored encoded in a Kubernetes Secret along with other sensitive information)

Configuration steps

1. Encode the S3 credentials. In this example, we use AWS S3 key and S3 key secret:

```
$ cat <<EOF | base64 --wrap=0
[global]
repo1-s3-key=<YOUR_AWS_S3_KEY>
repo1-s3-key-secret=<YOUR_AWS_S3_KEY_SECRET>
```

```
repo1-storage-verify-tls=y

EOF
```

`repo1-storage-verify-tls` enables TLS verification for pgBackRest when set to **y** (or omitted). Use **n** to turn verification off.

2. Create the Secret configuration file. Save the base64-encoded string from the previous step as the value of s3.conf in a Kubernetes Secret definition, for example:

```
apiVersion: v1

kind: Secret

metadata:

name: bireportingdb-pgbackrest-secrets

type: Opaque

data:

s3.conf: <base64-encoded-configuration-contents>
```

3. Create the Secrets object from this YAML file. Replace the `<namespace>` placeholder with your value:

```
$ kubectl apply -f bireportingdb-pgbackrest-secrets.yaml -n <namespace>
```

4. Update `values.yaml`. Add the secret name under `db.backups.storageSecretName` and supply the remaining S3 parameters under `db.backups.s3`. The secret name must match the one used in step 1.

```
...

db:

backups:

storageSecretName: bireportingdb-pgbackrest-secrets

s3:

bucket: "<YOUR_AWS_S3_BUCKET_NAME>"

endpoint: "<YOUR_AWS_S3_ENDPOINT>"

region: "<YOUR_AWS_S3_REGION>""
```

5. Create or update the cluster. Replace the placeholders with your values:

```
$ helm -n <namespace> <install/upgrade> <reporting-instance>
oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/bireporting --version
<reporting-version> -f values.yaml <--insecure-skip-tls-verify>
```

Google Cloud Storage

To use Google Cloud Storage as an object store for backups, you need the following information:

- **GCS bucket name** - Pass the bucket name to `pgBackRest` via the `gcs.bucket` key in the `backups.pgbackrest.repos` subsection of `deploy/cr.yaml`.
- **Service account key** - The Operator needs a JSON key to access storage.

Configuration steps

1. Create your service account key following the official Google Cloud instructions.

2. Export the key. In the Google Cloud Console:
   a. Navigate to **IAM & Admin** > **Service Accounts**.
   b. Select your service account > **Keys** tab.
   c. Click **Add Key** > **Create new key**, choose **JSON**, and download the resulting `gcs-key.json` file.

3. Create the Kubernetes Secret. The Secret consists of base64-encoded versions of two files: the `gcs-key.json` file with the Google service account key you have just downloaded, and the special gcs.conf configuration file.

- Create the `gcs.conf` configuration file.

```
[global]
repo1-gcs-key=/etc/pgbackrest/conf.d/gcs-key.json
```

- Encode both `gcs-key.json` and `gcs.conf` files.
- Create the Kubernetes Secret configuration file and specify your cluster name and the base64-encoded contents of the files from previous steps.

```
apiVersion: v1
kind: Secret
metadata:
name: bireportingdb-pgbackrest-secrets
type: Opaque
data:
gcs-key.json: <base64-encoded-json-file-contents>
gcs.conf: <base64-encoded-conf-file-contents>
```

4. Create the Secrets object from the Secret configuration file. Replace the `<namespace>` placeholder with your value:

```
$ kubectl apply -f cluster1-pgbackrest-secrets.yaml -n <namespace>
```

5. Update your `values.yaml` configuration. Specify your GCS credentials Secret in `db.backups.storageSecretName` and put GCS bucket name into the `bucket` option in the `db.backups.gcs.bucket` subsection. The repository name must be the same as the name you specified when you created the `gcs.conf` file.

```
...
db:
backups:
storageSecretName: bireportingdb-pgbackrest-secrets
gcs:
bucket: "<YOUR_GCS_BUCKET_NAME>"
```

6. Create or update the cluster. Replace the placeholders with your values:

```
$ helm -n <namespace> <install/upgrade> <reporting-instance>
oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/bireporting --version
<reporting-version> -f values.yaml <--insecure-skip-tls-verify>
```

Azure Blob storage

To use Microsoft Azure Blob Storage for storing backups, you need the following:

- **Azure container name** - The bucket where backups are stored.
- **Azure Storage credentials** -These are stored in an encoded form in the Kubernetes Secret.

Configuration steps

1. Encode the Azure Storage credentials with base64:

```
$ cat <<EOF | base64 --wrap=0
[global]
repo1-azure-account=<AZURE_STORAGE_ACCOUNT_NAME>
repo1-azure-key=<AZURE_STORAGE_ACCOUNT_KEY>
EOF
```

2. Create the Secret configuration file and specify the base64-encoded string from the previous step. Here is an example of the `bireportingdb-pgbackrest-secrets.yaml` Secret file:

```
apiVersion: v1
kind: Secret
metadata:
name: bireportingdb-pgbackrest-secrets
type: Opaque
data:
azure.conf: <base64-encoded-configuration-contents>
```

3. Create the Secrets object from this yaml file. Replace the `<namespace>` placeholder with your value:

```
$ kubectl apply -f bireportingdb-pgbackrest-secrets.yaml -n <namespace>
```

4. Update your `values.yaml` configuration. Specify the Secret file you have created in the previous step in the `db.backups.storageSecretName` subsection. Put Azure container name in the `db.–backups.azure.container` subsection. This name must match the name you used when you encoded the credentials in step 1.

```
...
db:
backups:
storageSecretName: bireportingdb-pgbackrest-secrets
...
```

```
azure:

container: "<YOUR_AZURE_CONTAINER>"
```

5. Create or update the cluster. Replace the placeholders with your values:

```
$ helm -n <namespace> <install/upgrade> <reporting-instance>
oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/bireporting --version
<reporting-version> -f values.yaml <--insecure-skip-tls-verify>
```

Backup retention

The Operator supports setting pgBackRest retention policies for full and differential backups. When a full backup expires, pgBackRest removes the associated files and write-ahead logs, which also deletes any incremental backups that depend on that full backup.

Set retention via the following `pgBackRest` options:

- `--retention-full` – how many full backups to retain.
- `--retention-full-type` – whether `retentionFull` is interpreted as days or count.

Backup retention type can be expressed as either `count` (keep a specific number of backups) or `time` (keep backups for a given number of days).

You can set both the type and retention policy in the `values.yaml` file:

```
db:

backups:

retentionFull: "14" # keep 14 days of full backups

retentionFullType: time # interpret the value as days
```

Scheduled backups

Backup schedules are defined in the `db.backups.schedules` subsection of the `value.yaml` file. You can supply a `schedules.<backup type>` key equal to an actual schedule that you specify in the cron-style format.

The following example shows the schedule for the repository:

```
db:

backups:

schedules:

full: "0 0 * * 6" # every Saturday at midnight

differential: "0 1 * * 1-6" # weekdays at 01:00
```

After updating the schedule, update the cluster:

```
$ helm -n <namespace> upgrade <instance name>
oci://harbor.utils.reiteklab.eu.rd.eilab.biz/enghousebi/bireporting --version <version>
-f values.yaml <--insecure-skip-tls-verify>
```

On-demand backups

To manually make an on-demand backup, you need a backup configuration file like the following `backup.yaml` example:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGBackup
metadata:
name: backup1
spec:
pgCluster: bireportingdb
repoName: repo1
# options:
# - --type=full
```

In the `backup.yaml` configuration file, specify the cluster name to be used for backups. If needed, you can add any pgBackRest command line options.

Make a backup with the following command:

```
$ kubectl apply -f deploy/backup.yaml
```

To list the backup, run:

```
$ kubectl get pg-backup
```

Restoring a cluster from a saved backup

The Operator supports both full restores and point-in-time recovery (PITR). Although you can restore to a new cluster or in-place to an existing cluster, the examples below focus on restoring to an existing PostgreSQL cluster.

Restore to an existing PostgreSQL cluster

To restore the previously saved backup, use a *backup restore* configuration file like in the following `restore.yaml` example:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGRestore
metadata:
name: restore1
spec:
pgCluster: bireportingdb
repoName: repo1
# options:
# - --type=time
```

```
# - --target="2022-11-30 15:12:11+03"
```

The following keys are the most important ones:

- `pgCluster` - specifies the name of your cluster.
- `repoName` - specifies the name of one of the pgBackRest repositories (Enghouse Reporting currently supports only `repo1`)
- `options` - passes through any pgBackRest command line options.

To start the restoration process, run the following command:

```
$ kubectl apply -f deploy/restore.yaml
```

Restore the cluster with point-in-time recovery

Point-in-time recovery allows users to revert the database back to a state before an unwanted change had occurred.

> **Note**
>
> For this feature to work, the Operator initiates a full backup immediately after the cluster creation to use it as a basis for the point-in-time recovery when needed (this backup is not listed in the output of the `kubectl get pg-backup` command).

You can set up a point-in-time recovery using the normal restore command of pgBackRest with few additional `spec.options` fields in `restore.yaml`:

- set `--type` option to `time`,
- set `--target` to a specific time you would like to restore to. You can use the typical string formatted as `<YYYY-MM-DD HH:MM:DD>`, optionally followed by a timezone offset: `"2021-04-16 15:13:32+00"` (`+00` in the above example means UTC),
- optional `--set` argument allows you to select a particular backup as the starting point for the point-in-time recovery. You can look through the available backups with the `kubectl get pg-backup` command to find out the proper backup name. This option must be specified if the target is one or more backups away from the current moment.

To start the restoration process, run the following command:

```
$ kubectl apply -f restore.yaml
```