Tungsten Operator (for MySQL) 8.0 Manual

Continuent Ltd

Tungsten Operator (for MySQL) 8.0 Manual

Continuent Ltd

Abstract

This manual documents Tungsten Operator (for MySQL).

This manual includes information for 8.0, up to and including 8.0.0.

Build date: 2025-11-18 (ece93a25)

Up to date builds of this document: Tungsten Operator (for MySQL) 8.0 Manual (Online), Tungsten Operator (for MySQL) 8.0 Manual (PDF)

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction to Tungsten Operator

Tungsten Operator™ helps to manage Tungsten MySQL clusters on a Kubernetes container platform. The operator follows the operator pattern, implementing the following features:

- Installation, configuration, management and deletion of Tungsten MySQL clusters
- Installation and configuration of Tungsten Connectors (Proxy)
- Backing up MySQL data to external object storage
- Scheduling of backups
- Initializing MySQL data from a configmap, external volume, HTTPS source or TungstenBackup
- kubectl plugin to help with managing instances of Tungsten clusters
- Resizing volumes and custom `PersistentVolumeClaim` templates
- Management of Tungsten Dataservices and configuration of Tungsten Manager via a generated `tungsten.ini` file

To get started, be sure to have checked that you have all the various prerequisites in place. These are outline here: Chapter 2, *Prerequisites*

You can then continue with installation and configuration by following the documentation in Chapter 3, *Getting Started with Tungsten Operator*

Chapter 2. Prerequisites

Before installaing and configuring Tungsten Operator a number of prerequisites need to be in place. These are outlined in the following sections.

> **Note**
>
> The documentation does not provide steps for installing, configuring or managing Kubernetes clusters, or the installation of Docker. It is assumed that you have extensive knowledge of this, and in situations of uncertainty, the appropriate provider documentation should be referenced.

> **Note**
>
> The operation and management of Tungsten Clusters are not covered in the chapters of this documentation. For all of the necessary operations, you should review the main Tungsten Clustering Documentation.

2.1. Docker Registry

You will need access to a Docker Container Registry to hold the images and act as a repository (For example in GCP, an Artifact Registry).

There are no specific requirments for this, only that it is accessible from your own environment.

2.2. Kubernetes

A pre-configured Kubernetes cluster will be needed. This documentation assumes that you have all needed knowledge and understanding for launching, managing and configuring the Kubernetes cluster before starting - steps for installation and configuration of the Kubernetes cluster are not provided here.

For full documentation for Kubernetes, please click here

However, there are a number of minimum requirements for Tungsten Operator, which should be followed, these are listed below:

- The minimum Kubernetes version must be at least v1.25 (Note: Versions prior to v1.30 are EOL)
- A minimum of 3 nodes are required, and ideally these should be distributed across different regions/zones.
- Each pod that will act as a Tungsten Cluster node will require sufficient memory allocated based on the following minimum requirements:
  - Minimum 1Gb for the Tungsten Manager
  - Minimum 2Gb for the Tungsten Replicator
  - Minimum 1Gb for the Tungsten Connector
  - Memory allocation for MySQL based on your own MySQL database requirements.
- cert-manager : Tungsten Operator has been tested and certified with versions higher than or equal to 1.16. Older versions may work, but have not been tested or certified.

2.3. Staging Host

A staging/admin host will be required from where you will be able to manage the deployment of Tungsten Operator and access nodes.

This section outlines the minimum requirements and configuration needed for this staging host, as follows:

- Connectivity from the staging host to the cloud hosting the Kubernetes Cluster and the Docker Registry.
- Docker - No specific version requirements, only that it is compatible with the version of Kubernetes in use.
- kubectl - No specific version requirements, only that it is in sync with the same version of Kubernetes in use.
- A configured directory for storing downloaded Tungsten images. Further examples in this documentation use `/opt/continuent/software` as this location and aligns with the recommended default for all Tungsten software staging locations.
- A non-root OS user, we recommend configuring an OS user called `tungsten`
- OPTIONAL: kind should be installed if you wish to use the staging host to deploy small local clusters for QA and Testing.

- OPTIONAL: cli tools such as those provided by AWS or GCP. These are not used during installation, however can be useful for testing connectivity

Chapter 3. Getting Started with Tungsten Operator

This section covers the basics for installation and configuration of a Tungsten Cluster using Tungsten Operator.

Before continuing with this section, you should ensure that all prerequisites are in place. See Chapter 2, *Prerequisites* for more information

3.1. Downloading the Provided Images

You should download the Tungsten Operator package file and the Docker image files from the Continuent Software Download portal, using your credentials. If you do not have access, contact Continuent Support.

All of the images are available for either Intel (amd64) or ARM (arm64) architecture and the filenames indicate this accordingly. The examples below use the Intel architecture files.

Tungsten Operator Package

You will need to first download the latest Tungsten Operator package and copy and unpack the file into the `/opt/continuent/software` directory on your staging host.

```
shell> cd /opt/continuent/software
shell> tar zxvf tungsten-operator-8.0.0-27.tar.gz
```

Docker Images

The Tungsten Docker image file contains 3 individual tar files, one for each component of Tungsten Clustering. Download the docker images file and unpack it within the `/opt/continuent/software` directory.

```
shell> cd /opt/continuent/software
shell> tar zxvf tungsten-clustering-docker-images-8.0.0-27-amd64.tar.gz
```

Load files into Docker Registry

Now that all the files are downloaded, we need to load them into your local Docker daemon and push to the registry.

The examples below use the Intel (amd64) architecture files and commands applicable to loading into an AWS ECR Docker registry. Syntax may differ for other environments and should be adjusted accordingly.

```
shell> export ARCH=amd64
shell> export RELEASE=8.0.0-27
shell> export REGION=us-east-1
shell> export REGISTRY=111222333444.dkr.ecr.us-east-1.amazonaws.com

shell> cd /opt/continuent/software/tungsten-operator-${RELEASE}
shell> docker load -i images/tungsten-operator-_${RELEASE}-${ARCH}.tar

shell> cd /opt/continuent/software/tungsten-docker-images-${RELEASE}
shell> docker load -i tungsten-connector-_${RELEASE}-${ARCH}.tar
shell> docker load -i tungsten-manager-${RELEASE}-${ARCH}.tar
shell> docker load -i tungsten-replicator-${RELEASE}-${ARCH}.tar
```

Retrieve an authentication token and authenthicate the Docker client to the ECR registry:

```
shell> aws ecr get-login-password --region ${REGION} | docker login --username AWS --password-stdin ${REGISTRY}
```

Tag the images:

```
shell> docker tag tungsten-operator:${RELEASE} ${REGISTRY}/tungsten-operator:${RELEASE}
shell> docker tag tungsten-connector:${RELEASE} ${REGISTRY}/tungsten-connector:${RELEASE}
shell> docker tag tungsten-manager:${RELEASE} ${REGISTRY}/tungsten-manager:${RELEASE}
shell> docker tag tungsten-replicator:${RELEASE} ${REGISTRY}/tungsten-replicator:${RELEASE}
```

Finally, push the images to the Docker Registry:

```
shell> docker push ${REGISTRY}/tungsten-operator:${RELEASE}
shell> docker push ${REGISTRY}/tungsten-connector:${RELEASE}
shell> docker push ${REGISTRY}/tungsten-manager:${RELEASE}
shell> docker push ${REGISTRY}/tungsten-replicator:${RELEASE}
```

3.2. Installation

Once all prerequisites are in place and the images have been loaded into your Docker Registry you can now install Tungsten Operator and Tungsten Cluster.

The first step is to install Tungsten Operator, which can be done either via the install.sh script or manually using helm. Once the Operator is installed, you then proceed to installing Tungsten Cluster. The following list outlines the order of the steps to be performed, linking to the relevant sections for ease:

- Step 1 : Chapter 2, *Prerequisites*
- Step 2 : Section 3.1, "Downloading the Provided Images"
- Step 3 : Section 3.2.1, "Installing Tungsten Operator"
- Step 4 : Section 3.2.2, "Installing Tungsten Cluster"

3.2.1. Installing Tungsten Operator

A very simple install.sh script is provided within the Operator Package; alternatively you can install issuing the helm commands manually. Both paths are outlined below.

3.2.1.1. Installing using install script

The install.sh is provided within the Operator package and can be used to easily launch a default 3-node cluster, with 2 connectors.

Usage:

Table 3.1. install.sh Options

Option	Description
`--help`	Show the help
`-r`, `--repository`	Specify the URL to the Docker Registry
`--skip-confirm`	Skip confirmation messages before proceeding
`--tag`, `-t`	Specify the Tungsten Operator version

If no options are specified, then the script will prompt for the registry URL and operator version, then continue to ensure prerequisites are in place such as cert-manager and kubectl.

The script will then go on to issue the helm commands to launch the cluster.

The example below shows the output of running the install.sh script with no options supplied:

```
shell> cd /opt/continuent/software/tungsten-operator-8.0.0-27
shell> ./install.sh

Please input the container registry URL where Tungsten operator image is stored (without a tag) [tungsten-operator]:
<Enter your registry URL here>

Please enter Tungsten operator version tag [8.0.0-27]:

Current Kubernetes context is set to: your-kubernetes-context
Press enter to continue or any other key to abort
Ensuring Cert-Manager is installed...
Installing Tungsten Operator Helm chart...
NAME: tungsten-operator
LAST DEPLOYED: Tue Feb 25 12:49:14 2025
NAMESPACE: tungsten-operator
STATUS: deployed
REVISION: 1
TEST SUITE: None
Waiting for operator to become ready...
deployment.apps/tungsten-operator condition met
Operator is ready!

Installation completed!

To create an example cluster, run:
kubectl apply -f examples/mysql-passwords.yaml && kubectl apply -f examples/tungsten_v1alpha1_tungstenmysqlcluster.yaml
```

3.2.1.2. Installing using helm command

If you do not wish to use the install.sh script, you can manually execute the helm commands to do perform the installation. These steps are shown below as examples:

```
shell> cd /opt/continuent/software/tungsten-operator-8.0.0-27
shell> helm install tungsten-operator charts/tungsten-operator-0.1.0.tgz
```

If you need to customize the registry of the image, this can be done by:

```
shell> helm install tungsten-operator --set image.repository=${REGISTRY}/tungsten-operator charts/tungsten-operator-0.1.0.tgz
```

It is recommended to install the operator in a separate namespace, for example:

```
shell> helm install tungsten-operator \
  --set image.repository=${REGISTRY}/tungsten-operator \
  --namespace=tungsten-operator \
  --create-namespace \
  charts/tungsten-operator-0.1.0.tgz
```

For a comprehensive list of configurable Helm values, you can extract the Helm chart and refer to the `values.yaml` file contained within it.

3.2.2. Installing Tungsten Cluster

Installation of the cluster is done via the kubectl command, passing in `yaml` files that pre-define the cluster configuration.

The following sections outline the sample files provided and how to customise them to suit your requirements.

3.2.2.1. Working with the YAML files

The first step is to build the `yaml` files, and to make this simple we provide a number of example files that can be used and adjusted to suit your requirements.

The files are located within the `examples` directory of the extracted operator package, for example:

```
shell> cd /opt/continuent/software/tungsten-operator-8.0.0-27/examples
```

The following table lists the available template files and what each one can be used for:

Table 3.2. Example YAML files provided

Option	Description
mysql-init-script.yaml	A ConfigMap used to initialize the database. This file can contain SQL statements that will be executed during installation.
mysql-passwords.yaml	Used to specify user account passwords, if not used then paswords will be auto-generated.
tungsten_v1alpha1_backup.yaml	Used as an example to create a backup. This is an operational task, not installation.
tungsten_v1alpha1_backupconfiguration.yaml	Examples to use to configure the backup endpoints - S3, Folder etc
tungsten_v1alpha1_mysqlcluster.yaml	Example deployment for a 3-node cluster.
tungsten_v1alpha1_mysqlcluster_caa.yaml	Example deployment for a 6-node Composite Active/Active cluster.
tungsten_v1alpha1_mysqlcluster_caa_with_init.yaml	Example deployment for a 6-node Composite Active/Active cluster that also calls the mysql-init-script.yaml
tungsten_v1alpha1_mysqlcluster_cap.yaml	Example deployment for a 6-node Composite Active/Passive cluster.
tungsten_v1alpha1_mysqlcluster_cap_with_affinity.yaml	Example deployment for a 6-node Composite Active/Passive cluster that set Kubernetes node affinity.
tungsten_v1alpha1_mysqlcluster_cap_with_init.yaml	Example deployment for a 6-node Composite Active/Passive cluster that also calls the mysql-init-script.yaml
tungsten_v1alpha1_mysqlcluster_from_backup.yaml	Example deployment for a 3-node cluster using a specified backup to provision from first.
tungsten_v1alpha1_mysqlcluster_with_init.yaml	Example deployment for a 3-node cluster that also calls the mysql-init-script.yaml
tungsten_v1alpha1_mysqlcluster_with_backup.yaml	Example deployment for a 3-node cluster that will also schedule backups at a specific schedule
tungsten_v1alpha1_opsrequest_reset_node.yaml	Used as an example to restore a backup. This is an operational task, not installation.
tungsten_v1alpha1_opsrequest_reset_replicator.yaml	Used as an example to reset the replicator. This is an operational task, not installation.

Important

Example YAML files are provided to deploy Composite Active/Passive and Composite Active/Active clusters, however at this time it is not possible to deploy these topologies across regions.

Depending on the use-case, you may choose to employ one or more of the sample YAML files, additionally you may wish to create one single YAML file that combines more than one of the above samples together. Whilst this is supported, it is not recommended as this may result in large files that may become dificult to manage.

An example of some of the use-cases that can be achieved using combine YAML files is shown below. This is not an exhaustive list of all possibilities, but are provided as examples of the kind of combinations you can leverage.

- To install a cluster, with auto-generated passwords:
  - One of `tungsten_v1alpha1_mysqlcluster.yaml`, `tungsten_v1alpha1_mysqlcluster_cap.yaml` or `tungsten_v1alpha1_mysqlcluster_caa.yaml`
- To install a cluster, defining your own custom passwords:
  - `mysql-passwords.yaml`
  - One of `tungsten_v1alpha1_mysqlcluster.yaml`, `tungsten_v1alpha1_mysqlcluster_cap.yaml` or `tungsten_v1alpha1_mysqlcluster_caa.yaml`
- To install a cluster, defining your own custom passwords, and manually take your own backups:
  - `mysql-passwords.yaml`
  - `tungsten_v1alpha1_backupconfiguration.yaml`
  - One of `tungsten_v1alpha1_mysqlcluster.yaml`, `tungsten_v1alpha1_mysqlcluster_cap.yaml` or `tungsten_v1alpha1_mysqlcluster_caa.yaml`
- To install a cluster, defining your own custom passwords, and configure automated backups:
  - `mysql-passwords.yaml`
  - `tungsten_v1alpha1_backupconfiguration.yaml`
  - `tungsten_v1alpha1_backup.yaml`
  - One of `tungsten_v1alpha1_tmysqlcluster.yaml`, `tungsten_v1alpha1_mysqlcluster_cap.yaml` or `tungsten_v1alpha1_mysqlcluster_caa.yaml`
- To install a cluster, defining your own custom passwords, and provisioning from an existing backup:
  - `mysql-passwords.yaml`
  - `tungsten_v1alpha1_backupconfiguration.yaml`
  - `tungsten_v1alpha1_mysqlcluster_from_backup.yaml`

3.2.2.2. Configuring Custom Passwords

The operator, by default, generates the MySQL root password, along with the passwords for the replicator and application user during the creation of the cluster.

You have the option to provide custom passwords using a pre-existing Kubernetes secret during cluster creation, using the following example as a guide.

You can use the provided `mysql-passwords.yaml` file as a template. Either edit directly or make a copy to a directory of your choice. You can then edit it, changing the passwords to suit.

An example is shown below:

```
apiVersion: v1
kind: Secret
metadata:
  name: tungsten-passwords-sample-secret
type: Opaque
stringData:
  mysqlRootPassword: rootsecret
  applicationPassword: appsecret
  replicatorPassword: replicatorsecret
  restApiAdminPassword: apisecret
```

In the above example, `apiVersion`, `kind` and `type` should be left untouched. The passwords within the `stringData` block can be changed to suit, along with the `name` in the `metadata` block.

This name can be used if you have multiple clusters and different password yaml files with different passwords for each cluster. You could then change the name to help identify which cluster the passwords are associated with. The value you set here should then be used within the `customPasswordSecret` property in the cluster yaml file.

For more details on the cluster yaml files, see Section 3.2.2.3, “Configuring Custom Cluster YAML templates”

To then apply this configuration, issue:

```
shell> kubectl apply -f mysql-passwords.yaml
```

> **Warning**
>
> Changing the passwords after cluster creation is not supported or tested. If passwords need to be changed, they can be first changed in the secret, and manually changed in the MySQL database.

3.2.2.3. Configuring Custom Cluster YAML templates

To configure the cluster, it is recommended to first select the example YAML file that matches the topology you wish the deploy, and then customize the content to match your needs.

A full list of all the sample files available is outlined in Section 3.2.2.1, “Working with the YAML files”

You can either edit the sample file directly or copy it and rename it to a different location. A number of options can be edited, and these are all explained below. Additionally, you can create your own if you wish but it is strongly recommended to take a sample file as the template to ensure you do not miss out any key entries.

Each YAML file has a number of key/value blocks depending on the complexity of the chosen deployment topology. The following is a complete example of the `tungsten_v1alpha1_mysqlcluster.yaml`, the sections that follow explain each individual component of the file and how it can be changed to suit your needs.

```
apiVersion: tungsten.continuent.com/v1alpha1
kind: MySQLCluster
metadata:
  labels:
    app.kubernetes.io/name: mysqlcluster
    app.kubernetes.io/instance: sample
    app.kubernetes.io/part-of: tungsten-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: tungsten-operator
  name: sample
spec:
  configuration:
    customPasswordSecret: tungsten-passwords-sample-secret
    enablePodDisruptionBudget: true
  monitoring:
    enabled: false
  dataServices:
  - name: alpha
    topology: clustered
    size: 3
  connector:
    replicas: 2
    podTemplate:
      spec:
        containers:
        - name: connector
          resources:
            requests:
              cpu: 1
              memory: 512Mi
            limits:
              memory: 512Mi
    # Uncomment to expose via external load balancer
    #serviceTemplate:
    #  spec:
    #    externalTrafficPolicy: Local
    #    type: LoadBalancer
  podTemplate:
    spec:
      containers:
      - name: mysql
        resources:
          requests:
            cpu: 1
            memory: 2Gi
          limits:
            memory: 2Gi
      - name: manager
        resources:
          requests:
            cpu: 100m
            memory: 512Mi
          limits:
            memory: 512Mi
```

```
      - name: replicator
        resources:
          requests:
            cpu: 1
            memory: 2Gi
          limits:
            memory: 2Gi
  volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 40Gi
  - metadata:
      name: thl
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 20Gi
  - metadata:
      name: state
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 100Mi
```

3.2.2.3.1. apiVersion and kind

At the top of every file, there will be `apiVersion` and `kind`. These values should be left as they are, and look like the following:

```
apiVersion: tungsten.continuent.com/v1alpha1
kind: MySQLCluster
```

3.2.2.3.2. metadata

The metadata section must also always be included with all values left as per the template, with the exception of `app.kubernetes.io/instance` and `name`. These values can be changed but must be set to the same value for each. An example of this section is as follows:

```
metadata:
  labels:
    app.kubernetes.io/name: mysqlcluster
    app.kubernetes.io/instance: sample
    app.kubernetes.io/part-of: tungsten-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: tungsten-operator
  name: sample
```

3.2.2.3.3. spec: configuration

Within this section, you specify the password configuration to associate with this cluster if you are setting custom passwords. The value for `customPasswordSecret` should be the same as the vlaue of `name` within the `mysql-passwords.yaml`, or the file you are using for the password secrets. An example of this section is as follows:

```
spec:
  configuration:
    customPasswordSecret: tungsten-passwords-sample-secret
    enablePodDisruptionBudget: true
```

3.2.2.3.4. spec: monitoring

Monitoring can be enabled by specifying true, or disabled by specifying false.

Setting this value to `true` will enable the various Prometheus exporters that will provide metrics that can be used with, for example, Graphana or other monitoring tools.

```
monitoring:
    enabled: false
```

3.2.2.3.5. spec: dataServices

The `dataServices` section allows you to confgure the desired topology and the cluster data service names. There are 3 different layouts for this section depending on the topology required, for example a standard cluster, a Composite Active/Passive cluster or a Composite Active/Active cluster. They all follow the same basic pattern.

The `name` is used for the corresponding dataservice. This must only be alphanumeric characters along with - or _ and is not case-sensitive. For Composite Active/Passive and Composite Active/Active topologies, each cluster must be given a different name.

`topology` is used to specify the topology of the cluster. The accepted values are as follows:

- `clustered`: This should be used to define a standard cluster. It should also be used to define the individual clusters within a composite cluster. See examples below.
- `composite-active-passive`: This should be used to define the top level cluster within a Composite Active/Passive cluster. `clustered` should then be used to define the clusters within.
- `composite-active-active`: This should be used to define the top level cluster within a Composite Active/Passive cluster. `clustered` should then be used to define the clusters within.

`sources` is used in composite clusters only. Within the top level composite cluster it defines the names of the clusters within. For a Composite Active/Passive custer it should also be defined within the passive cluster to define the source cluster.

`sizes` is used to specify the number of nodes within the cluster. This must be an odd number per cluster to conform to quorum.

Examples:

A standard 3-node cluster called `alpha`:

```
dataServices:
  - name: alpha
    topology: clustered
    size: 3
```

A Composite Active/Passive cluster with a composite cluster called `global` comprising of an active cluster of 3 nodes called `east` and a passive cluster of 3 nodes called `west`:

```
dataServices:
  - name: global
    topology: composite-active-passive
    sources:
    - east
    - west
  - name: east
    topology: clustered
    size: 3
  - name: west
    topology: clustered
    sources:
    - east
    size: 3
```

A Composite Active/Active cluster with a composite cluster called `global` comprising fo 2 x 3 node clusters called `north` and `south`:

```
dataServices:
  - name: global
    topology: composite-active-active
    sources:
    - north
    - south
  - name: north
    topology: clustered
    size: 3
  - name: south
    topology: clustered
    size: 3
```

3.2.2.3.6. spec: connector

The `connector` section allows you to define the number of connectors to launch using the `replicas` option. For Composite Active/Passive and Composite Active/Active topologies this will number is per cluster, so in the above examples where we define 2 clusters, setting `replicas` to 2 would result in a total of 4 connectors.

Additionally, within the `containers` block you can specify the number of CPU's and memory allocation per connector. The default values in the template are generally sufficient for the majority of workloads and shouldn't require changing.

An example of the `connector` block is as follows:

```
connector:
    replicas: 2
    podTemplate:
      spec:
        containers:
```

```
      - name: connector
        resources:
          requests:
            cpu: 1
            memory: 512Mi
          limits:
            memory: 512Mi
```

3.2.2.3.7. spec: podTemplate

The `podTemplate` section allows for configuration of the MySQL, Tungsten Manager and Tungsten Replicator containers, allowing you to specify appropriate Memory and CPU properties. The `name` values shouldn't be changed. The MySQL limits should be adjusted based on your expected workload in line with your MySQL usage. The Tungsten Manager and Tungsten Replicator default values are generally sufficient for the majority of workloads and shouldn't require changing.

An example of this block is as follows:

```
podTemplate:
    spec:
      containers:
      - name: mysql
        resources:
          requests:
            cpu: 1
            memory: 2Gi
          limits:
            memory: 2Gi
      - name: manager
        resources:
          requests:
            cpu: 100m
            memory: 512Mi
          limits:
            memory: 512Mi
      - name: replicator
        resources:
          requests:
            cpu: 1
            memory: 2Gi
          limits:
            memory: 2Gi
```

3.2.2.3.8. spec: volumeClaimTemplates

The `volumeClaimTemplates` section allows you to adjust the storage allocated. The only values that should be changed here are the `storage` values, adjusting to suit the expected usage for the MySQL databases and THL files. For THL allocation you should calculate this storage based on roughly 1.5-2 times the size of the binary logs generated within the retention period set [Default 10 days]

An example of this block is as follows:

```
volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 40Gi
  - metadata:
      name: thl
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 20Gi
  - metadata:
      name: state
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 100Mi
```

3.2.2.4. Configuring Custom tungsten.ini properties

If you are familiar with Tungsten Clustering, you will know that there are many properties that can be set via the `/etc/tungsten/tungsten.ini` file which will influence the behavior of the cluster. You have the option to include these custom properties by including the name/value key pairs within the `yaml` file.

You will need to include a section with the keyword `customTungstenIniProperties` under which you add the appropriate name and value. The location of this property will influence where within the `/etc/tungsten/tungsten.ini` the property is included.

For it to be part of the `[defaults]` and apply to all clusters, you should add this within `configuration`. For the values to apply only to a specific cluster, you should include this within the relevant cluster section under `dataServices` section.

The example below shows values included in both of these sections:

```
...
spec:
  configuration:
    customPasswordSecret: custom-tungsten-passwords-sample
    customTungstenIniProperties:
      - name: connector-bridge-mode
        value: "true"
      - name: repl-buffer-size
        value: "10"
  dataServices:
  - name: composite
    topology: composite-active-passive
    sources:
    - east
    - west
  - name: east
    topology: clustered
    size: 3
    customTungstenIniProperties:
      - name: connector-disconnect-timeout
        value: "10"
      - name: property=manager.prometheus.exporter.enabled
        value: "true"
...
```

For a full list of valid properties, their behavior and options, consult the Tungsten Clustering™ Doc pages

3.2.2.5. Installing the cluster using kubectl

When you have prepared the `yaml` files to suit your needs, you install the cluster using the kubectl apply command. The following example sets custom passwords using the `mysql-passwords.yaml` file and then creates a 3-node cluster using the `tungsten_v1alpha1_mysqlcluster.yaml` file:

```
shell> kubectl apply -f examples/mysql-passwords.yaml
shell> kubectl apply -f examples/tungsten_v1alpha1_mysqlcluster.yaml
```

3.3. Setting up a local test cluster using kind

Prior to deploying into production, you can setup a small QA cluster using kind

kind is a tool that allows you to run a local Kubernetes cluster within a Docker container. Before proceeding, ensure that Kind is installed on your system. For installation instructions, please refer to the Kind Quickstart Guide.

The following steps can be followed to create the setup:

If you are using a host different to the staging host, ensure the prerequisities are in place, including installation of kind, and then download and unpack the images to `/opt/continuent/software`

```
shell> cd /opt/continuent/software
shell> tar zxvf tungsten-operator-8.0.0-27.tar.gz
shell> cd tungsten-operator-8.0.0-27
```

Then, using kind, create the cluster and configure cert-manager:

```
shell> kind create cluster
shell> kind load image-archive images/tungsten-operator_8.0.0_amd64.tar
shell> kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.17/cert-manager.yaml
```

> **Note**
>
> It is recommended to use the latest version of cert-manager. At time of publishing, this is currenty v1.17

Wait a moment for cert_manager to be installed first, then install the operator:

```
shell> kubectl apply -f deploy.yaml
```

To cleanup:

```
shell> kind delete cluster
```

Chapter 4. Using the kubectl plugin

The operator package includes pre-compiled binaries for a kubectl plugin, which simplifies the management of Tungsten Clusters.

The kubectl plugin is installed automatically if you make use of the install.sh script. The steps below are only required if you choose to install the operator manually.

To install the plugin, copy the binary into an executable PATH, for example:

```
shell> cd /opt/continuent/software/tungsten-operator-8.0.0-27
shell> sudo cp kubectl-tungsten/kubectl-tungsten-$(uname -o | tr '[:upper:]' '[:lower:]')-$(uname -m) /usr/local/bin/kubectl-tungsten
```

After installation, the plugin is available by executing either kubectl tungsten or kubectl-tungsten, for example:

```
shell> kubectl tungsten cluster list
```

The following commands are available:

Table 4.1. kubectl tungsten Commands

Option	Description
backup	Manage Backups
cluster	Manage Clusters
completion	Generate the auto-completion script for the specified shell
expert	Expert Commands
help	Show help for any command, eg kubectl tungsten help`

To access more infomation about each command, you can use the help, for example:

```
shell> kubectl tungsten cluster help
```

4.1. kubectl tungsten backup Command

Table 4.2. kubectl tungsten backup Commands

Option	Description
create	Creates new backup of a cluster to external object storage
list	Lists available backups

4.1.1. kubectl tungsten backup create Command

Usage: `kubectl tungsten backup create BACKUP_NAME [flags]`

Creates a new TungstenBackup to initialize backup for supplied cluster name. Backup process is picked up by the operator and starts immediately.

Table 4.3. kubectl tungsten backup create Commands

Option	Description
-c, --cluster	Name of the Tungsten cluster
-t, --timeout	Timeout in minutes (default 300)
-w, --wait	Wait for backup to be completed

4.1.2. kubectl tungsten backup list Command

Usage: `kubectl tungsten backup list [flags]`

Lists available backups.

Table 4.4. kubectl tungsten backup list Commands

Option	Description
-c, --cluster	Name of the Tungsten cluster

4.2. kubectl tungsten cluster Command

Table 4.5. kubectl tungsten cluster Commands

Option	Description
`cctrl`	Get cctrl session in specified cluster
`list`	List clusters.
`proxy`	Proxies primary MySQL instance via connector to local machine
`recover`	Recover cluster or specific datasource
`reset`	Reset specific datasource
`switch`	Switch cluster to most advanced datasource or specific datasource
`trepctl`	Run trepctl in specified cluster

4.2.1. kubectl tungsten cluster cctrl Command

Usage: `kubectl tungsten cluster cctrl CLUSTER_NAME [args] [flags]`

Get cctrl shell session in specified cluster.

Table 4.6. kubectl tungsten cluster cctrl Commands

Option	Description
`-d`, `--dataservice`	Optional dataservice name. If not given, first physical dataservice will be used.
`-i`, `--stdin`	Pass stdin to the manager container (default true)
`-t`, `--tty`	Allocate a TTY to stdin (default true)

4.2.2. kubectl tungsten cluster proxy Command

Usage: `kubectl tungsten cluster proxy CLUSTER_NAME [options] [LOCAL_PORT]`

Proxies MySQL primary via Tungsten Connector.

Table 4.7. kubectl tungsten cluster proxy Commands

Option	Description
`--address`	Addresses to listen on (comma separated). Only accepts IP addresses or localhost as a value. When localhost is supplied, kubectl will try to bind on both 127.0.0.1 and ::1 and will fail if neither of these addresses are available to bind. (default [localhost])
`-d`, `--dataservice`	Optional dataservice name. If not given, first physical dataservice will be used.
`--pod-running-timeout`	The length of time (eg 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running (default 1m0s)

4.2.3. kubectl tungsten cluster recover Command

Usage: `kubectl tungsten cluster recover CLUSTER_NAME [datasource] [flags]`

Attempts to recover Tungsten cluster. When passed with optional parameter [datasource], attempts to recover only that datasource.

Table 4.8. kubectl tungsten cluster recover Commands

Option	Description
`-d`, `--dataservice`	Optional dataservice name. If not given, first physical dataservice will be used.

4.2.4. kubectl tungsten cluster reset Command

Usage: `kubectl tungsten cluster reset CLUSTER_NAME DATASOURCE_NAME [flags]`

Resets specific datasource of a Tungsten cluster.

Table 4.9. kubectl tungsten cluster reset Commands

Option	Description
`--backup`	Optional backup name. If not given, the latest backup will be used.
`-t, --timeout`	Timeout in minutes (default 300)
`-w, --wait`	Wait for reset to be completed

4.2.5. kubectl tungsten cluster switch Command

Usage: `kubectl-tungsten cluster switch CLUSTER_NAME [datasource] [flags]`

Attempts to switch primary instance of Tungsten Cluster. When passed with optional parameter [datasource], attempts to switch to that specific datasource.

Table 4.10. kubectl tungsten cluster switch Commands

Option	Description
`-d, --dataservice`	Optional dataservice name. If not given, first physical dataservice will be used.

4.3. kubectl tungsten completion Command

Table 4.11. kubectl tungsten completion Commands

Option	Description
`bash`	Generate the autocompletion script for bash
`fish`	Generate the autocompletion script for fish
`powershell`	Generate the autocompletion script for powershell
`zsh`	Generate the autocompletion script for zsh

4.4. kubectl tungsten expert Command

Table 4.12. kubectl tungsten expert Commands

Option	Description
`mysql`	Execute MySQL command in specified cluster member

4.5. kubectl tungsten Global Flags

Table 4.13. kubectl tungsten Global Flags

Option	Description
`--add_dir_header`	If true, adds the file directory to the header of the log messages
`--alsologtostderr`	Log to standard error as well as files (no effect when -logtostderr=true)
`--as-group`	Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
`--as-uid`	UID to impersonate for the operation.
`--as`	Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
`--cache-dir string`	Default cache directory (default '$HOME/.kube/cache')

Option	Description
`--certificate-authority`	Path to a cert file for the certificate authority
`--client-certificate`	Path to a client certificate file for TLS
`--client-key`	Path to a client key file for TLS
`--context`	The name of the kubeconfig context to use
`--disable-compression`	If true, opt-out of response compression for all requests to the server
`--insecure-skip-tls-verify`	If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
`--cluster`	The name of the kubeconfig cluster to use
`--kubeconfig`	Path to the kubeconfig file to use for CLI requests.
`--user`	The name of the kubeconfig user to use
`--log_backtrace_at`	when logging hits line file:N, emit a stack trace (default :0)
`--log_dir`	If non-empty, write log files in this directory (no effect when -logtostderr=true)
`--log_file`	If non-empty, use this log file (no effect when -logtostderr=true)
`--log_file_max_size`	Defines the maximum size a log file can grow to (no effect when -logtostderr=true). Unit is megabytes. If the value is 0, the maximum file size is unlimited. (default 1800)
`--logtostderr`	Log to standard error instead of files (default true)
`--namespace`, `-n`	If present, the namespace scope for this CLI request
`--one_output`	If true, only write logs to their native severity level (vs also writing to each lower severity level; no effect when -logtostderr=true)
`--request-timeout`	The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests. (default 0)
`-s`, `--server`	The address and port of the Kubernetes API server
`--skip_headers`	If true, avoid header prefixes in the log messages
`--skip_log_headers`	If true, avoid headers when opening log files (no effect when -logtostderr=true)
`--stderrthreshold`	Logs at or above this threshold go to stderr when writing to files and stderr (no effect when --logtostderr=true or --alsologtostderr=true) (default 2)
`--tls-server-name`	Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
`--token`	Bearer token for authentication to the API server
`-v [Level]`, `--v [Level]`	Number for the log level verbosity
`--vmodule moduleSpec`	Comma-separated list of pattern=N settings for file-filtered logging

Chapter 5. Advanced Configuration

5.1. Configuring Backups

The backup functionality in Tungsten Operator is managed through two main resources: `TungstenBackup` and `BackupConfiguration`.

Two sample YAML files are provided that allow you to configure and schedule backups, these are `tungsten_v1alpha1_backupconfiguration.yaml` and `tungsten_v1alpha1_mysqlcluster_with_backup.yaml`.

`TungstenBackup` is a custom resource that represents a single backup operation. It includes details such as the source cluster, plus the status and size of the backup. Once a `TungstenBackup` resource is created, the backup process is initiated.

`BackupConfiguration` is another custom resource that defines the configuration for backups. It includes details such as backup policies and storage destination and backend configuration.

5.1.1. Configuring backup backends

To build the `BackupConfiguration`, the example YAML file `tungsten_v1alpha1_backupconfiguration.yaml` can be used.

`BackupConfiguration` contains an array of `backendConfigurations`. Multiple backends can be configured, but only one will be active for new backups at any given moment. This allows the user to switch backends, while supporting restoration and rotation of `TungstenBackup` that were created on a previous backend.

Currently the only supported backend uploader type is rclone. rclone itself supports all major object storage destinations.

The sample YAML file, shows an example on how to configure an rclone backend using environment variables:

```
apiVersion: tungsten.continuent.com/v1alpha1
kind: BackupConfiguration
metadata:
  labels:
    app.kubernetes.io/name: backupconfiguration
    app.kubernetes.io/instance: backupconfiguration-sample
    app.kubernetes.io/part-of: tungsten-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: tungsten-operator
  name: backupconfiguration-sample
spec:
  backend: sample-rclone-backend
  backendConfigurations:
  - name: sample-rclone-backend
    type: rclone
    rclone:
      type: mysqldump
      destination:
        bucket: "backups"
        directory: "test"
      flags:
        purgeRemoteBackupOnDeletion: true
        overwriteExistingPhysicalResource: false
      env:
        - name: RCLONE_CONFIG_REMOTE_TYPE
          value: s3
        - name: RCLONE_CONFIG_REMOTE_ENDPOINT
          value: "http://minio.default.svc.cluster.local"
        - name: RCLONE_CONFIG_REMOTE_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: backups-user-1
              key: CONSOLE_ACCESS_KEY
        - name: RCLONE_CONFIG_REMOTE_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: backups-user-1
              key: CONSOLE_SECRET_KEY
```

The `type` defines the method of backup, Currently this can be set to either mysqldump or xtrabackup.

The `destination` block determines which backend from `backendConfigurations` will be used for all new backups.

The `flags` field includes several options that effect behaviour of backups:

- `purgeRemoteBackupOnDeletion`: This boolean field determines whether the remote physical backup file should be deleted when the `TungstenBackup` resource is deleted. If set to true, the remote backup will be deleted upon deletion of the `TungstenBackup` resource. As alternative life cycle

rules of the object storage backend can be used to ensure desired backup retention and rotation, and the Tungsten Operator can be configured to only upload backup artifacts.

- `overwriteExistingPhysicalResource`: This boolean field determines whether an existing physical resource should be overwritten during the backup process. If set to true, the backup process will overwrite any existing physical resource with the same name. This should be usually set to false.

When making major changes to the backend, such as changing the type or endpoint, it might be better to introduce a new backend instead, if you are not planning to copy old data between storage destinations. This way new instances of `TungstenBackups` can use the latest backend, while old backups still stay on the old backend.

Using the example above, you would then apply this to the cluster as follows:

```
shell> kubectl apply -f tungsten_v1alpha1_backupconfiguration.yaml
```

5.1.2. Scheduling the backup

The backup configuration and schedule is defined via the `MySQLCluster` resource's backups field. The schedule is specified in `cron` format and determines how frequently `TungstenBackup` resources are created and thus, how frequently backups are taken. If retention is specified, then successful `TungstenBackup` resources are deleted when count exceeds the value in `keep`.

An example YAML file is available in the examples directry called `tungsten_v1alpha1_mysqlcluster_with_backup.yaml` and the specific backup properties are shown below:

```
...
spec:
  configuration:
    customPasswordSecret: tungsten-passwords-sample-secret
  backup:
    configurationRef:
      name: tungstenbackupconfiguration-sample
    schedule: "* * * * *"
    keep: 5
...
```

The `name` refers to the name given in the `BackupConfiguration`

The lifecycle of a backup operation can be tracked through the `TungstenBackup` resource's status field. The possible states are `Pending`, `Processing`, `Succeeded`, and `Failed`.

5.2. Restoring Backups

There are two ways to restore a backup. You can either create a new cluster and intialise it with a backup, specifying the `fromBackup` section within the YAML file (See Section 5.3.2, “Initializing using existing TungstenBackup” for details), or you can use the kubectl tungsten commands as outlined below.

The following command will restore the latest backup available:

```
shell> kubectl tungsten cluster reset ClusterName DataSourceName
```

To specify a specific backup, just simple add the `--backup` to the above command, for example;

```
shell> kubectl tungsten cluster reset ClusterName DataSourceName --backup MyBackupName
```

Warning

When you restore a cluster from a backup, the MySQL grants that were auto-generated upon cluster creation, including those for the root, application, and replicator users, along with their passwords, are overwritten. This can lead to a mismatch between the stored passwords in cluster's password secret and the actual passwords in use. To avoid this issue, it's currently recommended to bypass the use of auto-generated passwords and instead set up a custom password secret for production use. For guidance on how to do this, please refer to Section 3.2.2.2, “Configuring Custom Passwords”.

Warning

After restoring a cluster from a backup, it is currently required to manually reset the replicator on the restored primary node. This can be achieved by executing the following commands in the replicator container:

```
shell> trepctl offline
shell> trepctl reset -all -y
shell> trepctl online
```

5.3. Importing Data

There are several methods available to initialize a Tungsten Cluster with data upon creation.

If an `init` block is specified within the YAML, or the `mysql-init-script.yaml` template is used, the cluster is initially created with only MySQL running, without any Tungsten components. Once the MySQL instances are up and running, each instance is populated with the specified data. After the data import process is completed, the Tungsten components are started. The replicator should then begin operating as expected.

Warning

It's crucial to ensure that the SQL statements used for initializing the cluster are deterministic. This means that they should produce the same result each time they are executed, ensuring that each MySQL instance ends up with identical data. Avoid using non-deterministic statements, such as `now()`, which can yield different results each time.

5.3.1. Initializing using SQL or URL

First, configure the init script, the example below is taken from the `mysql-init-script.yaml` file, and the `name` can be changed to suit if you wish. Make a note of the name, as that will then be used in the next step:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-init-script
data:
  import.sql: |
    create database sample;
    use sample;
    CREATE TABLE test(data VARCHAR(255));
    insert into test values("testing");
```

The above example shows raw SQL, however the same can also be achieved by specifying a URL, as shown in the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-init-script
spec:
  init:
    fromExternalArtifact:
      url: https://test.com/sample.sql.gz
```

Then, create the cluster using init block with `fromVolume`, ensuring `name` refers to the name you set in the previous step. The example below is taken from the `tungsten_v1alpha1_mysqlcluster_with_init.yaml` file:

```
...
  name: sample-with-init
spec:
  configuration:
    customPasswordSecret: tungsten-passwords-sample-secret
  init:
    fromVolume:
      configMap:
       name: mysql-init-script
  dataServices:
  - name: alpha
...
```

If you combine all of the above in a single file, you can then simply issue the following:

```
shell> kubectl apply -f <your-filename-here>
```

If you choose to keep two seperate files and use the same name as the templates:

```
shell> kubectl apply -f mysql-init-script.yaml
shell> kubectl apply -f tungsten_v1alpha1_mysqlcluster_with_init.yaml
```

5.3.2. Initializing using existing TungstenBackup

To initialize a cluster using an existing TungstenBackup, you need to ensure that the TungstenBackup is present in the same namespace as the cluster being created.

First, verify the name of the TungstenBackup you wish to use. You can do this by listing all available backups in the current namespace with the following command:

```
shell> kubectl tungsten backup list
```

Once you have confirmed the name of the TungstenBackup, you can proceed to create a new cluster. In the init block of your cluster configuration, specify the `fromBackup` field with the name of your TungstenBackup. Additionally, it's necessary to specify name of the configuration in `configurationRef`. Ensure that the configuration you specify is the same one used when the backup was created. Here is an example taken from the `tungsten_v1alpha1_backupconfiguration.yaml` which would have previously been applied with a name of `backupconfiguration-sample`:

```
...
  name: backupconfiguration-sample
spec:
  destination:
    bucket: "backups"
    directory: "test"
    backend: sample-rclone-backend
  policy:
    purgeRemoteBackupOnDeletion: true
    overwriteExistingPhysicalResource: false
...
```

Here is a sample taken from `tungsten_v1alpha1_mysqlcluster_from_backup.yaml` showing the init block that will use the backup configuration above, and the name of the available backup:

```
...
  name: sample-from-backup
spec:
  configuration:
    customPasswordSecret: tungsten-passwords-sample-secret
  backup:
    configurationRef:
      name: backupconfiguration-sample
  init:
    fromBackup:
      name: tungstenbackup-sample
  dataServices:
  - name: alpha
...
```

When the YAML files are ready, you would simply execute:

```
shell> kubectl apply -f tungsten_v1alpha1_mysqlcluster_from_backup.yaml
```

After creating the cluster, it's important to verify that the initialization jobs have completed successfully and that the cluster is in a healthy state. Monitor the status of your cluster and its jobs to ensure everything is functioning as expected.

Chapter 6. Operations

Operations provide a mechanism to execute sequential tasks that are not part of the regular reconciliation process. These tasks are implemented using the OpsRequest Custom Resource Definition (CRD). Each operation follows a specific flow until completion. Once an operation is completed, its corresponding OpsRequest CRD can be safely removed from the Kubernetes API.

6.1. Replicator Reset

The Replicator Reset operation is a specific type of operation that allows you to forcefully reset a replicator.

Automatic Execution

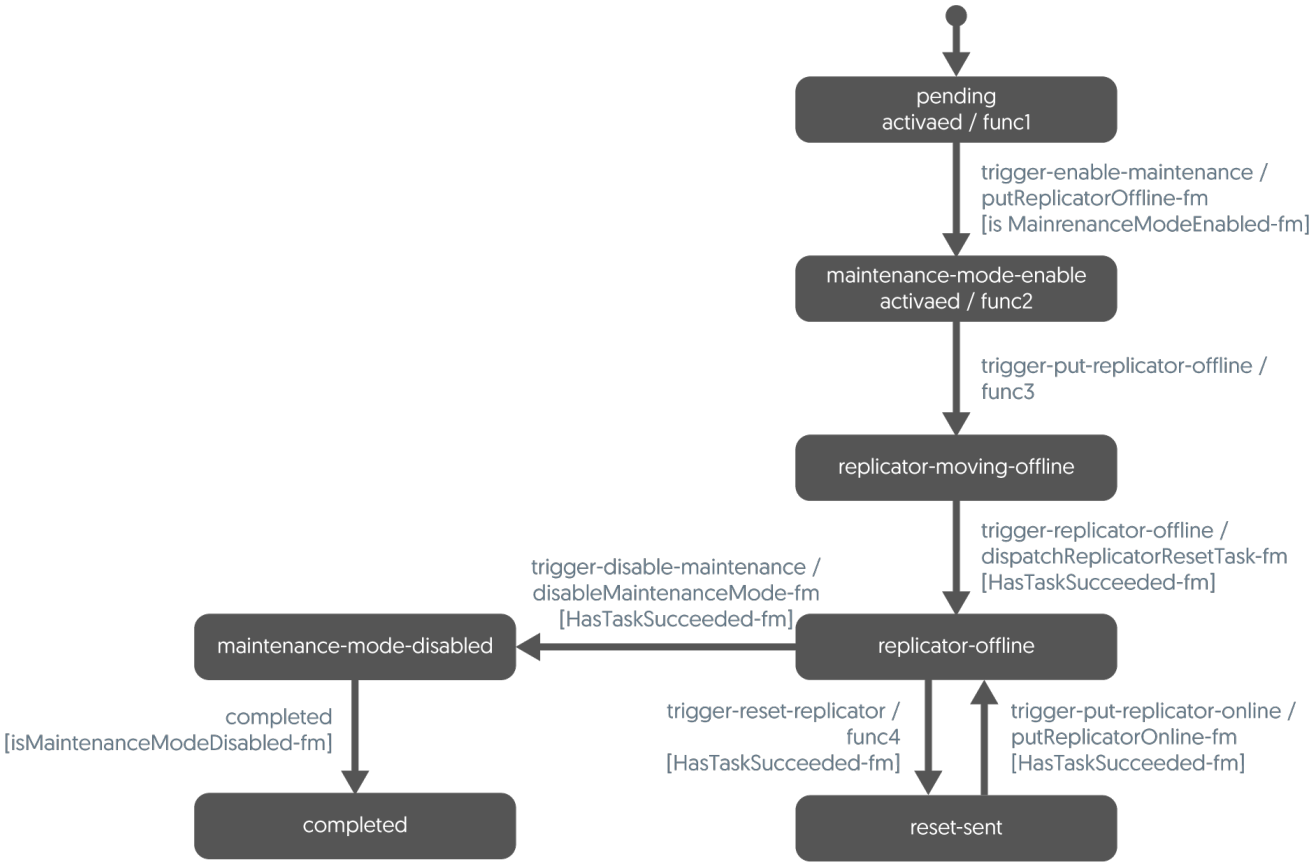
This operation is automatically executed under the following circumstances:

- After the creation and data import of a new dataservice on the primary (first) node of a dataservice, for exmaple

```
apiVersion: tungsten.continuent.com/v1alpha1
kind: OpsRequest
metadata:
  name: opsrequest-sample
spec:
  target:
    name: sample
  type: replicatorReset
  replicatorReset:
    dataService: alpha
    ordinal: 0
```

The reset flow can be best understood by the following diagram

Figure 6.1. Operations: Replicator Reset Flow



Appendix A. Release Notes

A.1. Tungsten Operator 8.0.0 GA (6 May 2025)

Version End of Life. Not Yet Set

Release 8.0.0 marks the first release of the Tungsten Operator.