MOS Operations Guide

version latest

Contents

Copyright notice

Preface

- About this documentation set
- Intended audience
- Technology Preview support scope
- Documentation history
- Conventions

About this documentation set

This documentation provides information on how to deploy and operate a Mirantis OpenStack for Kubernetes (MOS) environment. The documentation is intended to help operators to understand the core concepts of the product. The documentation provides sufficient information to deploy and operate the solution.

The information provided in this documentation set is being constantly improved and amended based on the feedback and kind requests from the consumers of MOS.

The following table lists the guides included in the documentation set you are reading:

Guides list

Guide	Purpose
MOS Reference Architecture	Learn the fundamentals of MOS reference architecture to appropriately plan your deployment
MOS Deployment Guide	Deploy a MOS environment of a preferred configuration using supported deployment profiles tailored to the demands of specific business cases
MOS Operations Guide	Operate your MOS environment
MOS Release notes	Learn about new features and bug fixes in the current MOS version

The MOS documentation home page contains references to all guides included in this documentation set. For your convenience, we provide all guides in HTML (default), single-page HTML, PDF, and ePUB formats. To use the preferred format of a guide, select the required option from the Formats menu next to the guide title.

Intended audience

This documentation is intended for engineers who have the basic knowledge of Linux, virtualization and containerization technologies, Kubernetes API and CLI, Helm and Helm charts, Mirantis Kubernetes Engine (MKE), and OpenStack.

Technology Preview support scope

This documentation set includes description of the Technology Preview features. A Technology Preview feature provide early access to upcoming product innovations, allowing customers to experience the functionality and provide feedback during the development process. Technology Preview features may be privately or publicly available and neither are intended for production use. While Mirantis will provide support for such features through official channels, normal Service Level Agreements do not apply. Customers may be supported by Mirantis Customer Support or Mirantis Field Support.

As Mirantis considers making future iterations of Technology Preview features generally available, we will attempt to resolve any issues that customers experience when using these features.

During the development of a Technology Preview feature, additional components may become available to the public for testing. Because Technology Preview features are being under development, Mirantis cannot guarantee the stability of such features. As a result, if you are using Technology Preview features, you may not be able to seamlessly upgrade to subsequent releases of that feature. Mirantis makes no guarantees that Technology Preview features will be graduated to a generally available product release.

The Mirantis Customer Success Organization may create bug reports on behalf of support cases filed by customers. These bug reports will then be forwarded to the Mirantis Product team for possible inclusion in a future release.

Documentation history

The following table contains the released revision of the documentation set you are reading:

Release date	Description
November 05, 2020	MOS GA release
December 23, 2020	MOS GA Update release

Conventions

This documentation set uses the following conventions in the HTML format:

Documentation conventions

Convention	Description
boldface font	Inline CLI tools and commands, titles of the procedures and system response examples, table titles
monospaced font	Files names and paths, Helm charts parameters and their values, names of packages, nodes names and labels, and so on
italic font	Information that distinguishes some concept or term
Links	External links and cross-references, footnotes

Main menu > menu item	GUI elements that include any part of interactive user interface and menu navigation
Superscript	Some extra, brief information
Note The Note block	Messages of a generic meaning that may be useful for the user
Caution! The Caution block	Information that prevents a user from mistakes and undesirable consequences when following the procedures
Warning The Warning block	Messages that include details that can be easily missed, but should not be ignored by the user and are valuable before proceeding
Seealso The See also block	List of references that may be helpful for understanding of some related tools, concepts, and so on
Learn more The Learn more block	Used in the Release Notes to wrap a list of internal references to the reference architecture, deployment and operation procedures specific to a newly implemented product feature

Introduction

This guide outlines the post-deployment Day-2 operations for a Mirantis OpenStack for Kubernetes environment. It describes how to configure and manage the MOS components, perform different types of cloud verification, and enable additional features depending on your cloud needs. The guide also contains day-to-day maintenance procedures such as how to back up and restore, update and upgrade, or troubleshoot your MOS cluster.

OpenStack operations

The section covers the management aspects of an OpenStack cluster deployed on Kubernetes.

Add a compute node

This section describes how to add a new compute node to your existing Mirantis OpenStack for Kubernetes deployment.

To add a compute node:

1. Add a bare metal host to the managed cluster with MOS as described in Mirantis Container Cloud Operations Guide: Add a bare metal host.
2. Create a Kubernetes machine in your cluster as described in Mitantis Container Cloud Operations Guide: Add a machine.

   When adding the machine, specify the node labels as required for an OpenStack compute node:

OpenStack node roles

Node role	Description	Kubernetes labels	Minimal count
OpenStack control plane	Hosts the OpenStack control plane services such as database, messaging, API, schedulers, conductors, L3 and L2 agents.	openstack-control-plane=enabled openstack-gateway=enabled openvswitch=enabled	3
OpenStack compute	Hosts the OpenStack compute services such as libvirt and L2 agents.	openstack-compute-node=enabled openvswitch=enabled (for a deployment with Open vSwitch as a back end for networking)	Varies

Delete a compute node

This section describes how to delete an OpenStack compute node from your MOS deployment.

To delete a compute node:

1. Migrate all workloads from the node. For more information, follow Nova official documentation: Migrate instances.
2. Ensure that there are no pods running on the node to delete by draining the node as instructed in the Kubernetes official documentation: Safely drain node.
3. Delete the node through the Mirantis Container Cloud web UI as described in Mirantis Container Cloud Operations Guide: Delete a machine.

Add a controller node

This section describes how to add a new control plane node to the existing MOS deployment.

To add an OpenStack controller node:

1. Add a bare metal host to the managed cluster with MOS as described in Mirantis Container Cloud Operations Guide: Add a bare metal host using CLI.

   When adding the bare metal host YAML file, specify the following OpenStack control plane node labels for the OpenStack control plane services such as database, messaging, API, schedulers, conductors, L3 and L2 agents:

   - openstack-control-plane=enabled
   - openstack-gateway=enabled
   - openvswitch=enabled
2. Create a Kubernetes machine in your cluster as described in Mirantis Container Cloud Operations Guide: Add a machine using CLI.

   When adding the machine, verify that OpenStack control plane node has the following labels:

   - openstack-control-plane=enabled
   - openstack-gateway=enabled
   - openvswitch=enabled

   > Note
   >
   > Depending on the applications that were colocated on the failed controller node, you may need to specify some additional labels, for example, ceph_role_mgr=true and ceph_role_mon=true . To successfuly replace a failed mon and mgr node, refer to Mirantis Container Cloud Operations Guide: Manage Ceph.

3. Verify that the node is in the Ready state through the Kubernetes API:

   ```
   kubectl get node <NODE-NAME> -o wide | grep Ready
   ```

4. Verify that the node has all required labels described in the previous steps:

   ```
   kubectl get nodes --show-labels
   ```

5. Configure new Octavia health manager resources:

   1. Rerun the octavia-create-resources job:

      ```
      kubectl -n osh-system exec -t <OS-CONTROLLER-POD> -c osdpl osctl-job-rerun octavia-create-resources openstack
      ```

   2. Wait until the Octavia health manager pod on the newly added control plane node appears in the Running state:

      ```
      kubectl -n openstack get pods -o wide | grep <NODE_ID> | grep octavia-health-manager
      ```

Note

If the pod is in the crashloopbackoff state, remove it:

```
kubectl -n openstack delete pod <OCTAVIA-HEALTH-MANAGER-POD-NAME>
```

3. Verify that an OpenStack port for the node has been created and the node is in the Active state:

```
kubectl -n openstack exec -t <KEYSTONE-CLIENT-POD-NAME> openstack port show octavia-health-manager-listen-port-<NODE-NAME>
```

Replace a failed controller node

This section describes how to replace a failed control plane node in your MOS deployment. The procedure applies to the control plane nodes that are, for example, permanently failed due to a hardware failure and appear in the NotReady state:

```
kubectl get nodes <CONTAINER-CLOUD-NODE-NAME>
```

Example of system response:

```
NAME                        STATUS     ROLES    AGE   VERSION
<CONTAINER-CLOUD-NODE-NAME>   NotReady   <none>   10d   v1.18.8-mirantis-1
```

To replace a failed controller node:

1. Remove the Kubernetes labels from the failed node by editing the .metadata.labels node object:

   ```
   kubectl edit node <CONTAINER-CLOUD-NODE-NAME>
   ```

2. Add the control plane node to your deployment as described in Add a controller node.
3. Identify all stateful applications present on the failed node:

   ```
   node=<CONTAINER-CLOUD-NODE-NAME>
   claims=$(kubectl -n openstack get pv -o jsonpath="{.items[?(@.spec.nodeAffinity.required.nodeSelectorTerms[0].matchExpressions[0].values[0] == '${node}')].spec.claimRef.name}")
   for i in $claims; do echo $i; done
   ```

   Example of system response:

   ```
   mysql-data-mariadb-server-2
   openstack-operator-bind-mounts-rfr-openstack-redis-1
   etcd-data-etcd-etcd-0
   ```

4. Reschedule stateful applications pods to healthy controller nodes as described in Reschedule stateful applications.

5. Remove the OpenStack port related to the Octavia health manager pod of the failed node:

```
kubectl -n openstack exec -t <KEYSTONE-CLIENT-POD-NAME> openstack port delete octavia-health-manager-listen-port-<NODE-NAME>
```

Reschedule stateful applications

The rescheduling of stateful applications may be required when replacing a permanently failed node, decomissioning a node, migrating applications to nodes with a more suitable set of hardware, and in several other use cases.

MOS deployment profiles include the following stateful applications:

- OpenStack database (MariaDB)
- OpenStack coordination (etcd)
- OpenStack Time Series Database back end (Redis)

Each stateful application from the list above has a persistent volume claim (PVC) based on a local persistent volume per pod. Each of control plane nodes has a set of local volumes available. To migrate an application pod to another node, recreate a PVC with the persistent volume from the target node.

> Caution!
>
> A stateful application pod can only be migrated to a node that does not contain other pods of this application.

> Caution!
>
> When a PVC is removed, all data present in the related persistent volume is removed from the node as well.

To reschedule pods to another control plane node:

1. Recreate a PVC on another control plane node:
    1. Select one of the persistent volumes available on the node:

    > Caution!
    >
    > A stateful application pod can only be migrated to the node that does not contain other pods of this application.

```
NODE_NAME=<NODE-NAME>
STORAGE_CLASS=$(kubectl -n openstack get osdpl <OSDPL_OBJECT_NAME> -o jsonpath='{.spec.local_volume_storage_class}')
kubectl -n openstack get pv -o json | jq --arg NODE_NAME $NODE_NAME --arg STORAGE_CLASS $STORAGE_CLASS -r '.items[] | select(.spec.nodeAffinity.required.nodeSelectorTerms[0].matchExpressions[0].values[0] == $NODE_NAME and .spec.storageClassName == $STORAGE_CLASS and .status.phase == "Available") | .metadata.name'
```

2. As the new PVC should contain the same parameters as the deleted one except for volumeName, save the old PVC configuration in YAML:

```
kubectl -n <NAMESPACE> get pvc <PVC-NAME> -o yaml > <OLD-PVC>.yaml
```

> Note
>
> <NAMESPACE> is a Kubernetes namespace where the PVC is created. For Redis, specify openstack-redis, for other applications specify openstack.

3. Delete the old PVC:

```
kubectl -n <NAMESPACE> delete pvc <PVC-NAME>
```

> Note
>
> If a PVC has stuck in the terminating state, run kubectl -n openstack edit pvc <PVC-NAME> and remove the finalizers section from metadata of the PVC.

4. Create the PVC with a new persistent volume:

```
cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: <PVC-NAME>
     namespace: <NAMESPACE>
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: <STORAGE-SIZE>
     storageClassName: <STORAGE-CLASS>
     volumeMode: Filesystem
     volumeName: <PV-NAME>
  EOF
```

> **Caution!**
>
> <STORAGE-SIZE>, <STORAGE-CLASS>, and <NAMESPACE> should correspond to the storage, storageClassName, and namespace values from the <OLD-PVC>.yaml file with the old PVC configuration.

2. Reschedule the pods:

   - For MariaDB:

     1. Remove the pod:

        > Note
        >
        > To remove a pod from a node in the NotReady state, add --grace-period=0 --force to the following command.

        ```
        kubectl -n openstack delete pod <STATEFULSET-NAME>-<NUMBER>
        ```

     2. Wait until the pod appears in the Ready state.

        When the rescheduling is finalized, the mariadb-server-2 pod joins back to the Galera cluster with a clean MySQL data directory and requests the Galera state transfer from the available nodes.

   - For etcd:

     1. Scale down the etcd StatefulSet to N-1 replicas, where N is the initial number of replicas in the etcd StatefulSet:

        ```
        kubectl -n openstack scale sts etcd-etcd --replicas=<N-1>
        ```

     2. Add the coordination section to the spec.services section of the OsDpl object:

        ```
        spec:
          services:
            coordination:
              etcd:
                values:
                  conf:
                    etcd:
                      ETCD_INITIAL_CLUSTER_STATE: existing
        ```

     3. Wait until all etcd pods appear in the Ready state.

4. Scale the etcd StatefulSet to the initial number of replicas:

```
kubectl -n openstack scale sts etcd-etcd --replicas=<NUMBER-OF-REPLICAS>
```

5. Wait until all etcd pods appear in the Ready state.
6. Remove the coordination section from the spec.services section of the OsDpl object.
7. Wait until all etcd pods appear in the Ready state.

- For Redis:

1. Remove the pod:

> Note
>
> To remove a pod from a node in the NotReady state, add --grace-period=0 --force to the following command.

```
kubectl -n openstack-redis delete pod <STATEFULSET-NAME>-<NUMBER>
```

2. Wait until the pod appears in the Ready state.

Back up and restore a MariaDB Galera database

MOS uses a MariaDB Galera database cluster to store data generated by OpenStack components. Mirantis recommends backing up your databases daily to ensure the integrity of your data. Also, you should create an instant backup before upgrading your database or for testing purposes.

MOS uses the Mariabackup utility to back up MariaDB Galera cluster data. Mariabackup is launched on periodic basis by a Kubernetes cron job, which is part of the MOS installation and is in the suspended state by default. To start running the job, you need to explicitly enable it in the OpenStackDeployment object.

Also, MOS provides the means for the MariaDB Galera cluster restoration using Mariabackup. During restoration, the job restores all Mariadb Galera nodes.

MariaDB backup workflow

The OpenStack database backup workflow includes the following phases:

1. Backup phase 1:

   The mariadb-phy-backup job launches the mariadb-phy-backup-<TIMESTAMP> pod. This pod contains the main backup script, which is responsible for:

   - Basic sanity checks and choosing right node for backup
   - Verifying the wsrep status and changing the wsrep_desync parameter settings

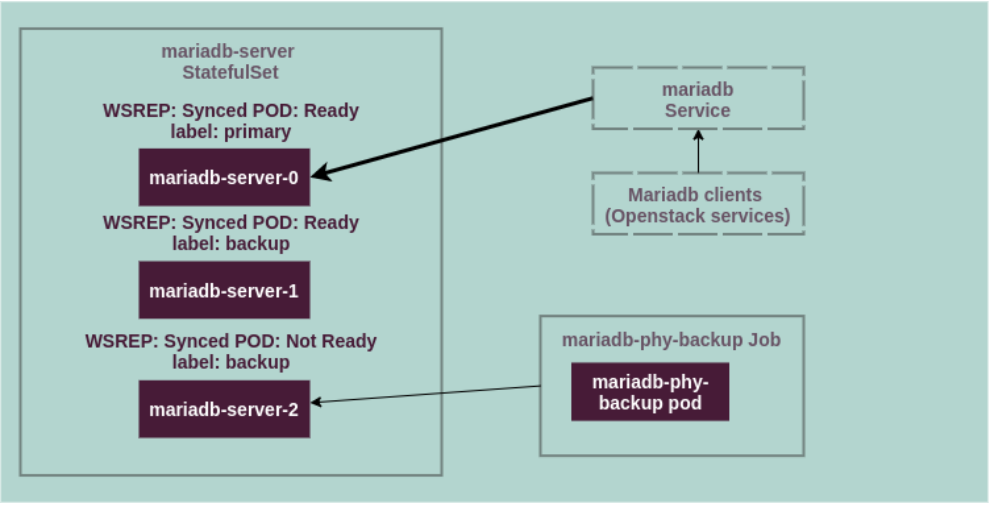
- Managing the mariadb-phy-backup-runner pod

During the first backup phase, the following actions take place:

1. Sanity check: verification of the Kubernetes status and wsrep status of each MariaDB pod. If some pods have wrong statuses, the backup job fails unless the --allow-unsafe-backup parameter is passed to the main script in the Kubernetes backup job.

   > **Note**
   >
   > Mirantis does not recommend setting the --allow-unsafe-backup parameter unless it is absolutely required. To ensure the consistency of a backup, verify that the MariaDB galera cluster is in a working state before you proceed with the backup.

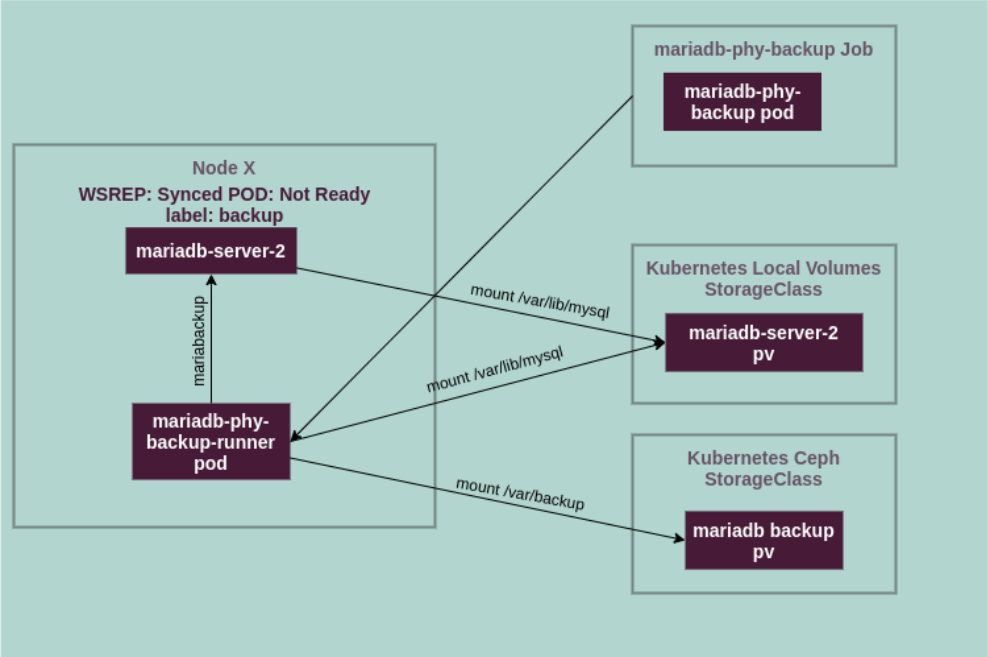
2. Select the replica to back up. The system selects the replica with the highest number in its name as a target replica. For example, if the MariaDB server pods have the mariadb-server-0, mariadb-server-1, and mariadb-server-2 names, the mariadb-server-2 replica will be backed up.
3. Desynchronize the replica from the Galera cluster. The script connects the target replica and sets the wsrep_desync variable to ON. Then, the replica stops receiving write-sets and receives the wsrep status Donor/Desynced. The Kubernetes health check of that mariadb-server pod fails and the Kubernetes status of that pod becomes Not ready. If the pod has the primary label, the MariaDB controller sets the backup label to it and the pod is removed from the endpoints list of the MariaDB service.



2. Backup phase 2:

   1. The main script in the mariadb-phy-backup pod launches the Kubernetes pod mariadb-phy-backup-runner-<TIMESTAMP> on the same node where the target mariadb-server replica is running, which is node X in the example.

2. The mariadb-phy-backup-runner pod has both mysql data directory and backup directory mounted. The pod performs the following actions:
    1. Verifies that there is enough space in the /var/backup folder to perform the backup. The amount of available space in the folder should be greater than <DB-SIZE> * <MARIADB-BACKUP-REQUIRED-SPACE-RATIO in KB.
    2. Performs the actual backup using the mariabackup tool.
    3. If the number of current backups is greater than the value of the MARIADB_BACKUPS_TO_KEEP job parameter, the script removes all old backups exceeding the allowed number of backups.
    4. Exits with 0 code.
3. The script waits untill the mariadb-phy-backup-runner pod is completed and collects its logs.
4. The script puts the backed up replica back to sync with the Galera cluster by setting wsrep_desync to OFF and waits for the replica to become Ready in Kubernetes.



MariaDB restore workflow

The OpenStack database restore workflow includes the following phases:

1. Restoration phase 1:

    The mariadb-phy-restore job launches the mariadb-phy-restore pod. This pod contains the main restore script, which is responsible for:

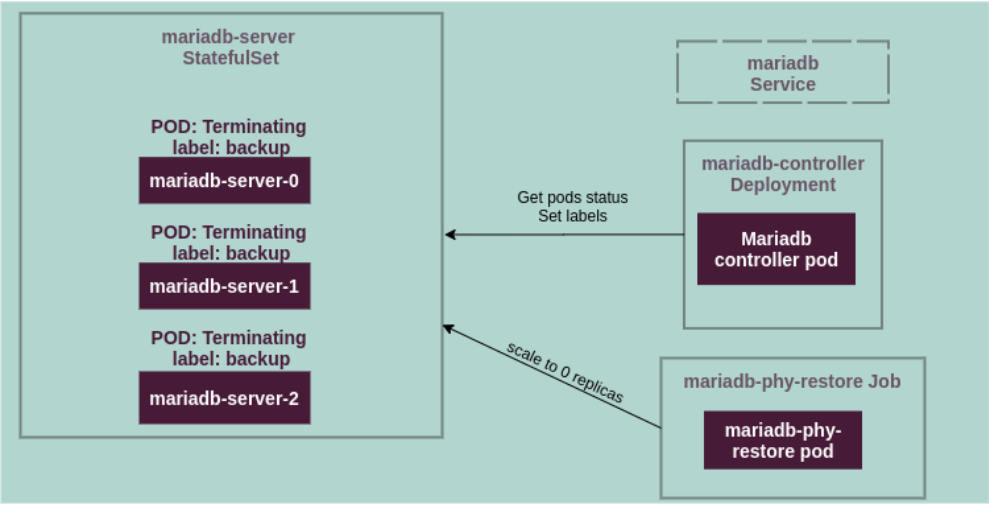
    - Scaling of the mariadb-server StatefulSet
    - Verifying of the mariadb-server pods statuses

- Managing of the openstack-mariadb-phy-restore-runner pods

> **Caution!**
>
> During the restoration, the database is not available for OpenStack services that means a complete outage of all OpenStack services.

During the first phase, the following actions are performed:

1. Save the list of mariadb-server persistent volume claims (PVC).
2. Scale the mariadb server StatefulSet to 0 replicas. At this point, the database becomes unavailable for OpenStack services.



2. Restoration phase 2:

    1. The mariadb-phy-restore pod launches openstack-mariadb-phy-restore-runner with the first mariadb-server replica PVC mounted to the /var/lib/mysql folder and the backup PVC mounted to /var/backup. The openstack-mariadb-phy-restore-runner pod performs the following actions:

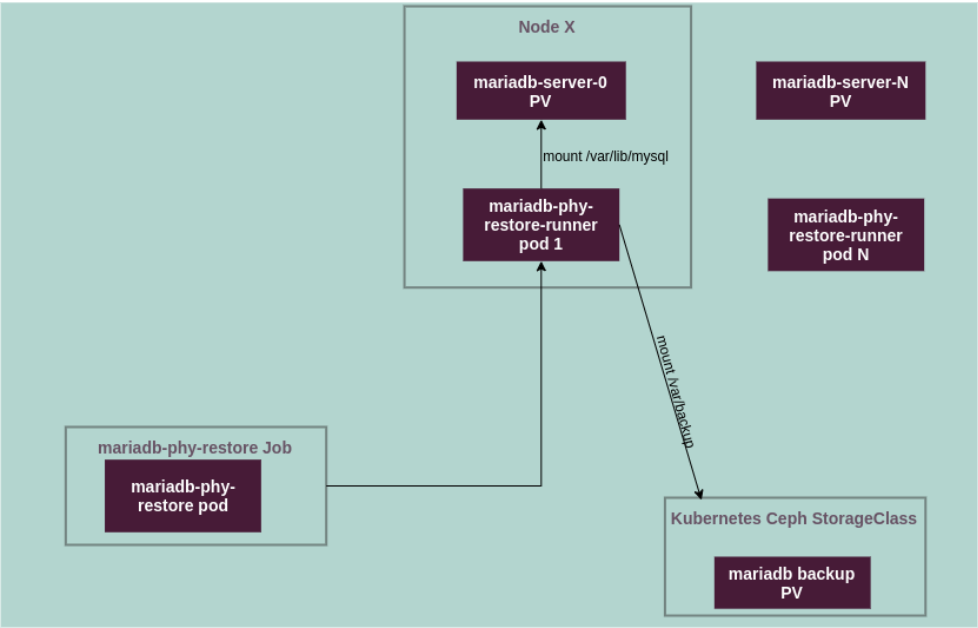
        1. Unarchives the database backup files to a temporary directory within /var/backup.
        2. Executes mariabackup --prepare on the unarchived data.
        3. Creates the .prepared file in the temporary directory in /var/backup.
        4. Restores the backup to /var/lib/mysql.
        5. Exits with 0.

    2. The script in the mariadb-phy-restore pod collects the logs from the openstack-mariadb-phy-restore-runner pod and removes the pod. Then, the script launches the next openstack-mariadb-phy-restore-runner pod for the next

mariadb-server replica PVC. The openstack-mariadb-phy-restore-runner pod restores the backup to /var/lib/mysql and exits with 0.

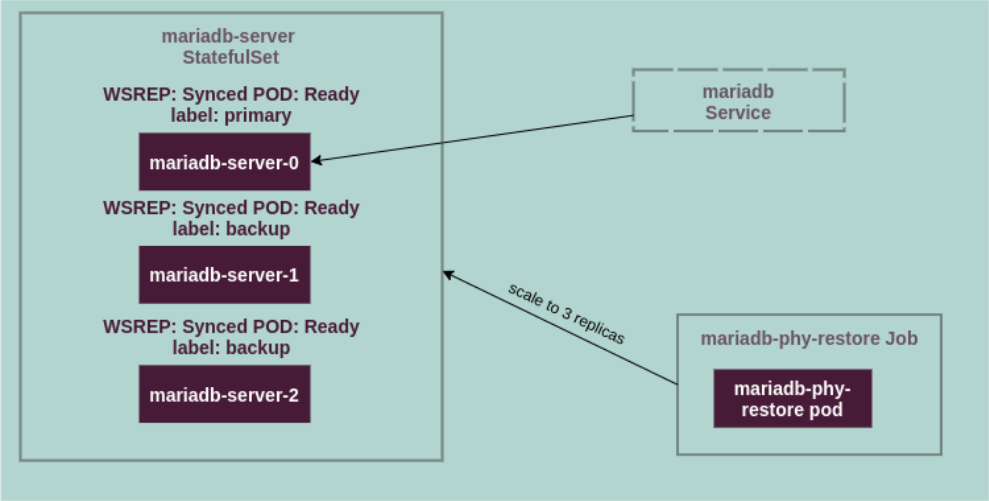
Step 2 is repeated for every mariadb-server replica PVC sequentially.

3. When the last replica's data is restored, the last openstack-mariadb-phy-restore-runner pod removes the .prepared file and the temporary folder with unachieved data from /var/backup.



3. Restoration phase 3:

    1. The mariadb-phy-restore pod scales the mariadb-server StatefulSet back to the configured number of replicas.
    2. The mariadb-phy-restore pod waits until all mariadb-server replicas are ready.



Limitations and prerequisites

The list of prerequisites includes:

- The MOS cluster should contain a preconfigured storage class with Ceph as a storage back end.
- The default sizes of volumes for backups should be configured as follows:
    - 20 GB for the tiny cluster size
    - 40 GB for the small cluster size
    - 80 GB for the medium cluster size

The list of limitations includes:

- Backup and restoration of specific databases and tables are not supported. MOS supports only backup and restoration of all databases in the mysql data directory.
- During the MariaDB Galera cluster restoration, the cron job restores the state on all MariaDB nodes sequentially. You cannot perform parallel restoration because Ceph Kubernetes volumes do not support concurrent mounting from different nodes.

Configure a periodic backup for MariaDB databases

After the MOS deployment, the cluster confguration includes the MariaDB backup functionality. By default, the Kubernetes cron job responsible for the MariaDB backup is in the suspended state.

To enable the MariaDB databases backup:

1. Enable the backup in the OpenStackDeployment object:

```
spec:
  features:
    database:
      backup:
        enabled: true
```

2. Verify that the mariadb-phy-backup CronJob object is present:

```
kubectl -n openstack get cronjob mariadb-phy-backup
```

Example of a positive system response:

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  annotations:
    openstackhelm.openstack.org/release_uuid: ""
  creationTimestamp: "2020-09-08T14:13:48Z"
  managedFields:
  <<<skipped>>>>
  name: mariadb-phy-backup
  namespace: openstack
  resourceVersion: "726449"
  selfLink: /apis/batch/v1beta1/namespaces/openstack/cronjobs/mariadb-phy-backup
  uid: 88c9be21-a160-4de1-afcf-0853697dd1a1
spec:
  concurrencyPolicy: Forbid
  failedJobsHistoryLimit: 1
  jobTemplate:
    metadata:
      creationTimestamp: null
      labels:
        application: mariadb-phy-backup
        component: backup
        release_group: openstack-mariadb
    spec:
      activeDeadlineSeconds: 4200
      backoffLimit: 0
      completions: 1
      parallelism: 1
      template:
        metadata:
          creationTimestamp: null
          labels:
            application: mariadb-phy-backup
            component: backup
            release_group: openstack-mariadb
        spec:
```

```
    containers:
    - command:
      - /tmp/mariadb_resque.py
      - backup
      - --backup-timeout
      - "3600"
      - --backup-type
      - incremental
      env:
      - name: MARIADB_BACKUPS_TO_KEEP
        value: "10"
      - name: MARIADB_BACKUP_PVC_NAME
        value: mariadb-phy-backup-data
      - name: MARIADB_FULL_BACKUP_CYCLE
        value: "604800"
      - name: MARIADB_REPLICAS
        value: "3"
      - name: MARIADB_BACKUP_REQUIRED_SPACE_RATIO
        value: "1.2"
      - name: MARIADB_RESQUE_RUNNER_IMAGE
        value: docker-dev-kaas-local.docker.mirantis.net/general/mariadb:10.4.14-bionic-20200812025059
      - name: MARIADB_RESQUE_RUNNER_SERVICE_ACCOUNT
        value: mariadb-phy-backup-runner
      - name: MARIADB_RESQUE_RUNNER_POD_NAME_PREFIX
        value: openstack-mariadb
      - name: MARIADB_POD_NAMESPACE
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.namespace
      image: docker-dev-kaas-local.docker.mirantis.net/general/mariadb:10.4.14-bionic-20200812025059
      imagePullPolicy: IfNotPresent
      name: phy-backup
      resources: {}
      securityContext:
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /tmp
        name: pod-tmp
      - mountPath: /tmp/mariadb_resque.py
        name: mariadb-bin
        readOnly: true
        subPath: mariadb_resque.py
      - mountPath: /tmp/resque_runner.yaml.j2
        name: mariadb-bin
```

```
        readOnly: true
        subPath: resque_runner.yaml.j2
      - mountPath: /etc/mysql/admin_user.cnf
        name: mariadb-secrets
        readOnly: true
        subPath: admin_user.cnf
    dnsPolicy: ClusterFirst
    initContainers:
    - command:
      - kubernetes-entrypoint
      env:
      - name: POD_NAME
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.name
      - name: NAMESPACE
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.namespace
      - name: INTERFACE_NAME
        value: eth0
      - name: PATH
        value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
      - name: DEPENDENCY_SERVICE
      - name: DEPENDENCY_DAEMONSET
      - name: DEPENDENCY_CONTAINER
      - name: DEPENDENCY_POD_JSON
      - name: DEPENDENCY_CUSTOM_RESOURCE
      image: docker-dev-kaas-local.docker.mirantis.net/openstack/extra/kubernetes-entrypoint:v1.0.0-20200311160233
      imagePullPolicy: IfNotPresent
      name: init
      resources: {}
      securityContext:
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        runAsUser: 65534
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    nodeSelector:
      openstack-control-plane: enabled
    restartPolicy: Never
    schedulerName: default-scheduler
    securityContext:
      runAsUser: 999
    serviceAccount: mariadb-phy-backup
    serviceAccountName: mariadb-phy-backup
    terminationGracePeriodSeconds: 30
    volumes:
    - emptyDir: {}
      name: pod-tmp
    - name: mariadb-secrets
      secret:
        defaultMode: 292
        secretName: mariadb-secrets
    - configMap:
        defaultMode: 365
        name: mariadb-bin
      name: mariadb-bin
  schedule: 0 1 * * *
  successfulJobsHistoryLimit: 3
  suspend: false
```

3. If required, modify the default backup configuration. By default, the backup is set up as follows:

- Runs on a daily basis at 01:00 AM
- Creates incremental backups daily and full backups weekly
- Keeps 10 latest full backups
- Saves backups to the mariadb-phy-backup-data PVC
- The backup timeout is 3600 seconds
- The backup type is incremental

As illustrated in the cron job exmample, the mariadb_resque.py script launches backups of the MariaDB Galera cluster. The script accepts settings through parameters and environment variables.

The following table describes the parameters that you can pass to the cron job and override from the OpenStackDeployment object.

MariaDB backup: Configuration parameters

Parameter	Type	Default	Description

--backup-type	String	incremental	Type of a backup. The list of possible values include:
			• incremental
			If the newest full backup is older than the value of the full_backup_cycle parameter, the system performs a full backup. Otherwise, the system performs an incremental backup of the newest full backup.
			• full
			Always performs only a full backup.
			Usage example:
			`spec:`
			`  features:`
			`  database:`
			`  backup:`
			`  backup_type: incremental`

--backup-timeout	Integer	21600	Timeout in seconds for the system to wait for the backup operation to succeed.
			Usage example:
			`spec:`
			`  services:`
			`  database:`
			`  mariadb:`
			`  values:`
			`  conf:`
			`  phy_backup:`
			`  backup_timeout: 30000`

--allow-unsafe-backup	Boolean	false	If set to true, enables the MariaDB cluster backup in a not fully operational cluster, where:
			• The current number of ready pods is not equal to MARIADB_REPLICAS.
			• Some replicas do not have healthy wsrep statuses.
			Usage example:
			`spec:`
			`  services:`
			`  database:`
			`  mariadb:`
			`  values:`
			`  conf:`
			`  phy_backup:`
			`  allow_unsafe_backup: true`

The following table describes the environment variables that you can pass to the cron job and override from the OpenStackDeployment object.

MariaDB backup: Environment variables

Variable	Type	Default	Description

MARIADB_BACKUPS_TO_KEEP	Integer	10	Number of full backups to keep.
			Usage example:
			`spec:`
			`  features:`
			`  database:`
			`  backup:`
			`  backups_to_keep: 3`

MARIADB_BACKUP_PVC_NAME	String	mariadb-phy-backup-data	Persistent volume claim used to store backups.
			Usage example:
			`spec:`
			`  services:`
			`  database:`
			`  mariadb:`
			`  values:`
			`  conf:`
			`  phy_backup:`
			`  backup_pvc_name: mariadb-phy-backup-data`

MARIADB_FULL_BACKUP_CYCLE	Integer	604800	Number of seconds that defines a period between 2 full backups. During this period, incremental backups are performed. The parameter is taken into account only if backup_type is set to incremental. Otherwise, it is ignored. For example, with full_backup_cycle set to 604800 seconds a full backup is taken weekly and, if cron is set to 0 0 * * *, an incremental backup is performed on daily basis.
			Usage example:
			`spec:`
			`  features:`
			`  database:`
			`  backup:`
			`  full_backup_cycle: 70000`

MARIADB_BACKUP_REQUIRED_SPACE_RATIO	Floating	1.2	Multiplier for the database size to predict the space required to create a backup, either full or incremental, and perform a restoration keeping the uncompressed backup files on the same file system as the compressed ones.
			To estimate the size of MARIADB_BACKUP_REQUIRED_SPACE_RATIO, use the following formula: size of (1 uncompressed full backup + all related incremental uncompressed backups + 1 full compressed backup) in KB =< (DB_SIZE * MARIADB_BACKUP_REQUIRED_SPACE_RATIO) in KB.
			The DB_SIZE is the disk space allocated in the MySQL data directory, which is /var/lib/mysql, for databases data excluding galera.cache and ib_logfile* files. This parameter prevents the backup PVC from being full in the middle of the restoration and backup procedures. If the current available space is lower than DB_SIZE * MARIADB_BACKUP_REQUIRED_SPACE_RATIO, the backup script fails before the system starts the actual backup and the overall status of the backup job is failed.
			Usage example:
			`spec:`
			`  services:`
			`  database:`
			`  mariadb:`
			`  values:`
			`  conf:`
			`  phy_backup:`
			`  backup_required_space_ratio: 1.4`

For example, to perform full backups monthly and incremental backups daily at 02:30 AM and keep the backups for the last six months, configure the database backup in your OpenStackDeployment object as follows:

```
spec:
  features:
    database:
      backup:
        enabled: true
        backups_to_keep: 6
        schedule_time: '30 2 * * *'
        full_backup_cycle: 2628000
```

> Seealso
>
> Verify operationability of the MariaDB backup jobs

Restore MariaDB databases

During the restoration procedure, the Mariadb service will be unavailable, because of the MariaDB StatefulSet scale down to 0 replicas. Therefore, plan the maintenance window according to the database size. The speed of the restoration may depend on the following:

- Network throughput
- Storage performance where backups are kept (Ceph by default)
- Local disks performance of nodes where MariaDB local volumes are present

To restore MariaDB databases:

1. Obtain an image of the MariaDB container:

   ```
   kubectl -n openstack get pods mariadb-server-0 -o jsonpath='{.spec.containers[0].image}'
   ```

2. Create the check_pod.yaml file to create the helper pod required to view the backup volume content:

   ```
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: check-backup-helper
     namespace: openstack
   ---
   apiVersion: v1
   kind: Pod
   metadata:
     name: check-backup-helper
     namespace: openstack
     labels:
       application: check-backup-helper
   spec:
     nodeSelector:
       openstack-control-plane: enabled
     containers:
       - name: helper
         securityContext:
           allowPrivilegeEscalation: false
           runAsUser: 0
           readOnlyRootFilesystem: true
         command:
   ```

```
      - sleep
      - infinity
    image: << image of mariadb container >>
    imagePullPolicy: IfNotPresent
    volumeMounts:
      - name: pod-tmp
        mountPath: /tmp
      - mountPath: /var/backup
        name: mysql-backup
  restartPolicy: Never
  serviceAccount: check-backup-helper
  serviceAccountName: check-backup-helper
  volumes:
    - name: pod-tmp
      emptyDir: {}
    - name: mariadb-secrets
      secret:
        secretName: mariadb-secrets
        defaultMode: 0444
    - name: mariadb-bin
      configMap:
        name: mariadb-bin
        defaultMode: 0555
    - name: mysql-backup
      persistentVolumeClaim:
        claimName: mariadb-phy-backup-data
```

3. Create the helper pod:

```
kubectl -n openstack apply -f check_pod.yaml
```

4. Obtain the name of the backup to restore:

```
kubectl -n openstack exec -t check-backup-helper -- tree /var/backup
```

Example of system response:

```
/var/backup
|-- base
|   `-- 2020-09-09_11-35-48
|       |-- backup.stream.gz
|       |-- backup.successful
|       |-- grastate.dat
|       |-- xtrabackup_checkpoints
|       `-- xtrabackup_info
|-- incr
|   `-- 2020-09-09_11-35-48
|       |-- 2020-09-10_01-02-36
```

```
|       |-- 2020-09-11_01-02-02
|       |-- 2020-09-12_01-01-54
|       |-- 2020-09-13_01-01-55
|       `-- 2020-09-14_01-01-55
`-- lost+found

10 directories, 5 files
```

If you want to restore the full backup, the name from the example above is 2020-09-09_11-35-48. To restore a specific incremental backup, the name from the example above is 2020-09-09_11-35-48/2020-09-12_01-01-54.

In the example above, the backups will be restored in the following strict order:

1. 2020-09-09_11-35-48 - full backup, path /var/backup/base/2020-09-09_11-35-48
2. 2020-09-10_01-02-36 - incremental backup, path /var/backup/incr/2020-09-09_11-35-48/2020-09-10_01-02-36
3. 2020-09-11_01-02-02 - incremental backup, path /var/backup/incr/2020-09-09_11-35-48/2020-09-11_01-02-02
4. 2020-09-12_01-01-54 - incremental backup, path /var/backup/incr/2020-09-09_11-35-48/2020-09-12_01-01-54

5. Delete the helper pod:

```
kubectl -n openstack delete -f check_pod.yaml
```

6. Pass the following parameters to the mariadb_resque.py script from the OsDpl object:

Parameter	Type	Default	Description
--backup-name	String		Name of a folder with backup in <BASE_BACKUP> or <BASE_BACKUP>/<INCREMENTAL_BACKUP>.
--replica-restore-timeout	Integer	3600	Timeout in seconds for 1 replica data to be restored to the mysql data directory. Also, includes time for spawning a rescue runner pod in Kubernetes and extracting data from a backup archive.

7. Edit the OpenStackDeployment object as follows:

```
spec:
  services:
    database:
      mariadb:
        values:
          manifests:
            job_mariadb_phy_restore: true
```

```
    conf:
      phy_restore:
        backup_name: "2020-09-09_11-35-48/2020-09-12_01-01-54"
        replica_restore_timeout: 7200
```

8. Wait until the mariadb-phy-restore job suceeds:

```
kubectl -n openstack get jobs mariadb-phy-restore -o jsonpath='{.status}'
```

Verify operationability of the MariaDB backup jobs

To verify operationability of the MariaDB backup jobs:

1. Verify pods in the openstack namespace. After the backup jobs have succeeded, the pods stay in the Completed state:

   ```
   kubectl -n openstack get pods -l application=mariadb-phy-backup
   ```

   Example of a posistive system response:

   ```
   NAME                                READY   STATUS      RESTARTS   AGE
   mariadb-phy-backup-1599613200-n7jqv   0/1     Completed   0          43h
   mariadb-phy-backup-1599699600-d79nc   0/1     Completed   0          30h
   mariadb-phy-backup-1599786000-d5kc7   0/1     Completed   0          6h17m
   ```

   > Note
   >
   > By default, the system keeps three latest successful and one latest failed pods.

2. Obtain an image of the MariaDB container:

   ```
   kubectl -n openstack get pods mariadb-server-0 -o jsonpath='{.spec.containers[0].image}'
   ```

3. Create the check_pod.yaml file to create the helper pod required to view the backup volume content.

   Configuration example:

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: check-backup-helper
     namespace: openstack
   ---
   ```

```
apiVersion: v1
kind: Pod
metadata:
  name: check-backup-helper
  namespace: openstack
  labels:
    application: check-backup-helper
spec:
  nodeSelector:
    openstack-control-plane: enabled
  containers:
    - name: helper
      securityContext:
        allowPrivilegeEscalation: false
        runAsUser: 0
        readOnlyRootFilesystem: true
      command:
        - sleep
        - infinity
      image: << image of mariadb container >>
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: pod-tmp
          mountPath: /tmp
        - mountPath: /var/backup
          name: mysql-backup
  restartPolicy: Never
  serviceAccount: check-backup-helper
  serviceAccountName: check-backup-helper
  volumes:
    - name: pod-tmp
      emptyDir: {}
    - name: mariadb-secrets
      secret:
        secretName: mariadb-secrets
        defaultMode: 0444
    - name: mariadb-bin
      configMap:
        name: mariadb-bin
        defaultMode: 0555
    - name: mysql-backup
      persistentVolumeClaim:
        claimName: mariadb-phy-backup-data
```

4. Apply the helper service account and pod resources:

```
kubectl -n openstack apply -f check_pod.yaml
kubectl -n openstack get pods -l application=check-backup-helper
```

Example of a positive system response:

```
NAME                READY   STATUS    RESTARTS   AGE
check-backup-helper   1/1     Running   0          27s
```

5. Verify the directories structure within the /var/backup directory of the spawned pod:

```
kubectl -n openstack exec -t check-backup-helper -- tree /var/backup
```

Example of a system response:

```
/var/backup
|-- base
|   `-- 2020-09-09_11-35-48
|       |-- backup.stream.gz
|       |-- backup.successful
|       |-- grastate.dat
|       |-- xtrabackup_checkpoints
|       `-- xtrabackup_info
|-- incr
|   `-- 2020-09-09_11-35-48
|       |-- 2020-09-10_01-02-36
|       |   |-- backup.stream.gz
|       |   |-- backup.successful
|       |   |-- grastate.dat
|       |   |-- xtrabackup_checkpoints
|       |   `-- xtrabackup_info
|       `-- 2020-09-11_01-02-02
|           |-- backup.stream.gz
|           |-- backup.successful
|           |-- grastate.dat
|           |-- xtrabackup_checkpoints
|           `-- xtrabackup_info
```

The base directory contains full backups. Each directory in the incr folder contains incremental backups related to a certain full backup in the base folder. All incremental backups always have the base backup name as parent folder.

6. Delete the helper pod:

```
kubectl delete -f check_pod.yaml
```

Run Tempest tests

The OpenStack Integration Test Suite (Tempest), is a set of integration tests to be run against a live OpenStack cluster. This section instructs you on how to verify the workability of your OpenStack deployment using Tempest.

To verify an OpenStack deployment using Tempest:

1. Configure Tempest as required.

   To change the Tempest run parameters, use the following structure in the OsDpl CR:

   ```
   spec:
     services:
       tempest:
         tempest:
           values:
             conf:
               script: |
                 tempest run --config-file /etc/tempest/tempest.conf \\
                 --concurrency 4 --blacklist-file /etc/tempest/test-blacklist --smoke
   ```

   The following example structure from the OsDpl CR will set image:build_timeout to 600 in the tempest.conf file:

   ```
   spec:
     services:
       tempest:
         tempest:
           values:
             conf:
               tempest:
                 image:
                   build_timeout: 600
   ```

2. Run Tempest. The OpenStack Tempest is deployed like other OpenStack services in a dedicated openstack-tempest Helm release by adding tempest to spec:features:services in the OSDPL resource.

   ```
   spec:
     features:
       services:
         - tempest
   ```

3. Wait until Tempest is ready. The Tempest tests are launched by the openstack-tempest-run-tests job. To keep track of the tests execution, run:

   ```
   kubectl -n openstack logs -l application=tempest,component=run-tests
   ```

4. Get the Tempest results. The Tempest results can be stored in a pvc-tempest PersistentVolumeClaim (PVC). To get them from a PVC, use:

   ```
   # Run pod and mount pvc to it
   cat <<EOF | kubectl apply -f -
   ```

```
apiVersion: v1
kind: Pod
metadata:
  name: tempest-test-results-pod
  namespace: openstack
spec:
  nodeSelector:
    openstack-control-plane: enabled
  volumes:
    - name: tempest-pvc-storage
      persistentVolumeClaim:
        claimName: pvc-tempest
  containers:
    - name: tempest-pvc-container
      image: ubuntu
      command: ['sh', '-c', 'sleep infinity']
      volumeMounts:
        - mountPath: "/var/lib/tempest/data"
          name: tempest-pvc-storage
EOF
```

5. If required, copy the results locally:

```
kubectl -n openstack cp tempest-test-results-pod:/var/lib/tempest/data/report_file.xml .
```

6. Remove the Tempest test results pod:

```
kubectl -n openstack delete pod tempest-test-results-pod
```

7. To rerun Tempest:

    1. Remove Tempest from the list of enabled services.
    2. Wait until Tempest jobs are removed.
    3. Add Tempest back to the list of the enabled services.

Seealso

MOS Reference Architecture: OpenStackDeployment custom resource

Update OpenStack

> Caution!
>
> Before you update a managed cluster, reconfigure the default value of the max_pods option to allow scheduling of more than 110 pods per a Kubernetes worker. For details, see MOS Release Notes: MOS Ussuri Update known issues.

The update of the OpenStack components is performed during the MOS managed cluster release update through the Container Cloud web UI as described in Mirantis Container Cloud Operations Guide: Update a managed cluster.

Calculate a maintenance window duration

This section provides the background information on the approximate time spent on operations for pods of different purposes, possible data plane impact during these operations, and the possibility of a parallel pods update. Such data helps the cloud administrators to correctly estimate maintenance windows and impacts on the workloads for your OpenStack deployment.

Maintenance window calculation

Pod name	Pod description	Kubernetes kind	Readiness time	Data plane impact	Parallel update
[*]-api	Contains API services of OpenStack components. Horizontally well scalable.	Deployment	<30s	NO	YES (batches 10% of overall count)
[*]-conductor	Contains proxy service between OpenStack and database.	Deployment	<30s	NO	YES (batches 10% of overall count)
[*]-scheduler	Spreads OpenStack resources between nodes.	Deployment	<30s	NO	YES (batches 10% of overall count)
[*]-worker [*]-engine [*]-volume [*]-backup [*]	Process user requests.	Deployment	<30s	NO	YES (batches 10% of overall count)

nova-compute	Processes user requests, interacts with the data plane services.	Daemon Set	<120s	NO	YES (batches 10% of overall count)
neutron-l3-agent	Creates virtual routers (spawns keepalived processes for the HA routers).	Daemon Set	10-15m (for 100 routers)	YES	NO (one by one)
neutron-openvswitch-agent	Configures tunnels between nodes.	Daemon Set	<120s	NO	YES (batches 10% of overall count)
neutron-dhcp-agent	Configures the DHCP server for the networking service.	Daemon Set	<30s	Partially (only if the downtime exceeds the lease timeout.	YES (batches 10% of overall count)
neutron-metadata-agent	Provides metadata information to user workloads (VMs).	Daemon Set	<30s	NO	YES (batches 10% of overall count)
libvirt	Starts the libvirtd communication daemon.	Daemon Set	<30s	NO	YES (batches 10% of overall count)
openvswitch-[*]	Sets up the Open vSwitch datapaths and then operates the switching across each bridge.	Daemon Set	<30s	YES	NO (one by one)
mariadb-[*]	Contains persistent storage (database) for OpenStack deployment.	Stateful Set	<180s	NO	NO (one by one)
memcached-[*]	Contains the memory object caching system.	Deployment	<30s	NO	NO (one by one)
[*]-rabbitmq-[*]	Contains the messaging service for OpenStack.	Stateful Set	<30s	NO	NO (one by one)

Remove an OpenStack cluster

This section instructs you on how to remove an OpenStack cluster, deployed on top of Kubernetes, by deleting the openstackdeployments.lcm.mirantis.com (OsDpl) CR.

To remove an OpenStack cluster:

1. Verify that the OsDpl object is present:

   ```
   kubectl get osdpl -n openstack
   ```

2. Delete the OsDpl object:

   ```
   kubectl delete osdpl osh-dev -n openstack
   ```

   The deletion may take a certain amount of time.

3. Verify that all pods and jobs have been deleted and no objects are present in the command output:

   ```
   kubectl get pods,jobs -n openstack
   ```

4. Delete the Redis failover object:

   ```
   kubectl get redisfailover -n openstack-redis
   kubectl delete redisfailover openstack-redis -n openstack-redis
   ```

5. Delete the Redis operator:

   ```
   kubectl delete helmbundle redis-operator -n osh-system
   ```

6. Delete Persistent Volume Claims (PVCs) using the following snippet. Deletion of PVCs causes data deletion on Persistent Volumes. The volumes themselves will become available for further operations.

   > **Caution!**
   >
   > Before deleting PVCs, save valuable data in a safe place.

   ```
   #!/bin/bash
   PVCS=$(kubectl get pvc  --all-namespaces | egrep "redis|etcd|mariadb" | awk '{print $1" "$2" "$4}'| column -t | awk 'NR>1')
   echo  "$PVCS" | while read line; do
   PVC_NAMESPACE=$(echo "$line" | awk '{print $1}')
   PVC_NAME=$(echo "$line" | awk '{print $2}')
   echo "Deleting PVC ${PVC_NAME}"
   kubectl delete pvc ${PVC_NAME} -n ${PVC_NAMESPACE}
   done
   ```

   > Note
   >
   > Deletion of PVCs may get stuck if a resource that uses the PVC is still running. Once the resource is deleted, the PVC deletion process will proceed.

7. Delete the MariaDB state ConfigMap:

```
kubectl delete configmap openstack-mariadb-mariadb-state -n openstack
```

8. Delete secrets using the following snippet:

```
#!/bin/bash
SECRETS=$(kubectl get secret  -n openstack | awk '{print $1}'| column -t | awk 'NR>1')
echo  "$SECRETS" | while read line; do
echo "Deleting Secret ${line}"
kubectl delete secret ${line} -n openstack
done
```

9. Verify that OpenStack ConfigMaps and secrets have been deleted:

```
kubectl get configmaps,secrets -n openstack
```

Ceph operations

To manage a running Ceph cluster, for example, to add or remove a Ceph OSD, remove a Ceph node, replace a failed physical disk with a Ceph node, or update your Ceph cluster, refer to Mirantis Container Cloud Operations Guide: Manage Ceph.

Before you proceed with any reading or writing operation, first check the cluster status using the ceph tool as described in Mirantis Container Cloud Operations Guide: Verify the Ceph core services.

StackLight operations

The section covers the StackLight management aspects.

View Grafana dashboards

Using the Grafana web UI, you can view the visual representation of the metric graphs based on the time series databases.

This section describes only the OpenStack-related Grafana dashboards. For other dashboards, including the system, Kubernetes, Ceph, and StackLight dashboards, see Mirantis Container Cloud Operations Guide: View Grafana dashboards.

To view the Grafana dashboards:

1. Log in to the Grafana web UI as described in Mirantis Container Cloud Operations Guide: Access StackLight web UIs.
2. From the drop-down list, select the required dashboard to inspect the status and statistics of the corresponding service deployed in Mirantis OpenStack on Kubernetes:

Dashboard	Description
OpenStack - Overview	Priovides general information on OpenStack services resources consumption, API errors, deployed OpenStack compute nodes and block storage usage.
KPI - Downtime	Provides uptime statistics for OpenStack compute instances, including graphs on created VMs status and availability.
KPI - Provisioning	Provides provisioning statistics for OpenStack compute instances, including graphs on VM creation results by day.
Cinder	Provides graphs on the OpenStack Block Storage service health, HTTP API availability, pool capacity and utilization, number of created volumes and snapshots.
Glance	Provides graphs on the OpenStack Image service health, HTTP API availability, number of created images and snapshots.
Gnocchi	Provides panels and graphs on the Gnocchi health and HTTP API availability.
Heat	Provides graphs on the OpenStack Orchestration service health, HTTP API availability and usage.
Ironic	Provides graphs on the OpenStack Bare Metal Provisioning service health, HTTP API availability, provisioned nodes by state and installed ironic-conductor back-end drivers.
Keystone	Provides graphs on the OpenStack Identity service health, HTTP API availability, number of tenants and users by state.
Neutron	Provides graphs on the OpenStack networking service health, HTTP API availability, agents status and usage of Neutron L2 and L3 resources.

NGINX Ingress controller	Monitors the number of requests, response times and statuses, as well as the number of Ingress SSL certificates including expiration time and resources usage.
Nova - Availability Zones	Provides detailed graphs on the OpenStack availability zones and hypervisor usage.
Nova - Hypervisor Overview	Provides a set of single-stat panels presenting resources usage by host.
Nova - Instances	Provides graphs on libvirt Prometheus exporter health and resources usage. Monitors the number of running instances and tasks and allows sorting the metrics by top instances.
Nova - Overview	Provides graphs on the OpenStack compute services (nova-scheduler, nova-conductor, and nova-compute) health, as well as HTTP API availability.
Nova - Tenants	Provides graphs on CPU, RAM, disk throughput, IOPS, and space usage and allocation and allows sorting the metrics by top tenants.
Nova - Users	Provides graphs on CPU, RAM, disk throughput, IOPS, and space usage and allocation and allows sorting the metrics by top users.
Nova - Utilization	Provides detailed graphs on Nova hypervisor resources capacity and consumption.
Memcached	Memcached Prometheus exporter dashboard. Monitors Kubernetes Memcached pods and displays memory usage, hit rate, evicts and reclaims rate, items in cache, network statistics, and commands rate.
MySQL	MySQL Prometheus exporter dashboard. Monitors Kubernetes MySQL pods, resources usage and provides details on current connections and database performance.
RabbitMQ	RabbitMQ Prometheus exporter dashboard. Monitors Kubernetes RabbitMQ pods, resources usage and provides details on cluster utilization and performance.

View Kibana dashboards

Using the Kibana web UI, you can view the visual representation of your OpenStack deployment notifications.

This section describes only the Openstack-related Kibana dashboards. For other dashboards, including the logs and Kubernetes events dashboards, see Mirantis Container Cloud Operations Guide: View Kibana dashboards.

To view the Kibana dashboards:

1. Log in to the Kibana web UI as described in Mirantis Container Cloud Operations Guide: Access StackLight web UIs.
2. Click the required dashboard to inspect the visualizations or perform a search:

Dashboard	Description
Notifications	Provides visualizations on the number of notifications over time per source and severity, host, and breakdowns. Includes search.
Audit	Provides search on audit notifications.

Available StackLight alerts

This section provides an overview of the available predefined OpenStack-related StackLight alerts. To view the alerts, use the Prometheus web UI. To view the firing alerts, use Alertmanager or Alerta web UI.

For other alerts, including the node, Kubernetes, Ceph, and StackLight alerts, see Mirantis Container Cloud Operations Guide: Available StackLight alerts.

Core services

This section describes the alerts available for the core services.

libvirt

This section lists the alerts for the libvirt service.

- LibvirtDown

LibvirtDown

Severity	Critical
Summary	Failure to gather libvirt metrics.
Description	The libvirt metric exporter fails to gather metrics on the {{ $labels.host }} host for 2 minutes.

MariaDB

This section lists the alerts for the MariaDB service.

- MariadbGaleraDonorFallingBehind
- MariadbGaleraNotReady
- MariadbGaleraOutOfSync
- MariadbInnodbLogWaits
- MariadbInnodbReplicationFallenBehind
- MariadbTableLockWaitHigh

MariadbGaleraDonorFallingBehind

Severity	Warning
Summary	MariaDB cluster donor node is falling behind.
Description	The MariaDB cluster node is falling behind (the queue size is {{ $value }}).

MariadbGaleraNotReady

Severity	Major
Summary	MariaDB cluster is not ready.
Description	The MariaDB cluster is not ready to accept queries.

MariadbGaleraOutOfSync

Severity	Minor
Summary	MariaDB cluster node is out of sync.
Description	The MariaDB cluster node is not in sync ({{ $value }} != 4).

MariadbInnodbLogWaits

Severity	Warning
Summary	MariaDB InnoDB log writes are stalling.
Description	The MariaDB InnoDB logs are waiting for the disk at a rate of {{ $value }} per second.

MariadbInnodbReplicationFallenBehind

Severity	Warning
Summary	MariaDB InnoDB replication is lagging.
Description	The MariaDB InnoDB replication has fallen behind and is not recovering.

MariadbTableLockWaitHigh

Severity	Minor
Summary	MariaDB table lock waits are high.
Description	MariaDB has {{ $value }}% of table lock waits.

Memcached

This section lists the alerts for the Memcached service.

- MemcachedServiceDown
- MemcachedConnectionsNoneMinor
- MemcachedConnectionsNoneMajor
- MemcachedEvictionsLimit

MemcachedServiceDown

Severity	Minor
Summary	The Memcached service is down.
Description	The Memcached service in the {{ $labels.namespace }} namespace is down.

MemcachedConnectionsNoneMinor

Severity	Minor
Summary	Memcached has no open connections.
Description	The Memcached database in the {{ $labels.namespace }} namespace has no open connections.

MemcachedConnectionsNoneMajor

Severity	Major
Summary	Memcached has no open connections on all nodes.
Description	The Memcached database has no open connections on all nodes.

MemcachedEvictionsLimit

Severity	Warning
Summary	Memcached has 10 evictions.
Description	The average of {{ $value }} evictions in the Memcached database occurred in the {{ $labels.namespace }} namespace during the last minute.

SSL certificates

This section describes the alerts for the OpenStack SSL certificates.

- OpenstackSSLCertExpirationMajor
- OpenstackSSLCertExpirationWarning
- OpenstackSSLProbesFailing

OpenstackSSLCertExpirationMajor

Severity	Major

Summary	SSL certificate for an OpenStack service expires in 10 days.
Description	The SSL certificate for the Openstack {{ $labels.service }} service endpoint {{ $labels.instance }} expires in 10 days.

OpenstackSSLCertExpirationWarning

Severity	Warning
Summary	SSL certificate for an OpenStack service expires in 30 days.
Description	The SSL certificate for the OpenStack {{ $labels.service }} service endpoint {{ $labels.instance }} expires in 30 days.

OpenstackSSLProbesFailing

Severity	Critical
Summary	SSL certificate probes for an OpenStack service are failing.
Description	The SSL certificate probes for the OpenStack {{ $labels.service }} service endpoint {{ $labels.instance }} are failing.

RabbitMQ

This section lists the alerts for the RabbitMQ service.

- RabbitMQNetworkPartitionsDetected
- RabbitMQDown
- RabbitMQFileDescriptorUsageWarning
- RabbitMQNodeDiskFreeAlarm
- RabbitMQNodeMemoryAlarm

RabbitMQNetworkPartitionsDetected

Severity	Warning

Summary	RabbitMQ network partitions detected.
Description	The RabbitMQ server {{ $labels.node }} in the {{ $labels.namespace }} namespace detected {{ $value }} network partitions.

RabbitMQDown

Severity	Critical
Summary	RabbitMQ is down.
Description	The RabbitMQ server monitored by {{ $labels.service }} in the {{ $labels.namespace }} namespace is down for the last 2 minutes.

RabbitMQFileDescriptorUsageWarning

Severity	Warning
Summary	RabbitMQ file descriptors consumption is high for the last 10 minutes.
Description	The RabbitMQ server {{ $labels.node }} in the {{ $labels.namespace }} namespace uses {{ $value }}% of file descriptors.

RabbitMQNodeDiskFreeAlarm

Severity	Warning
Summary	RabbitMQ disk space consumption is high.
Description	The RabbitMQ server {{ $labels.node }} in the {{ $labels.namespace }} namespace has low free disk space available.

RabbitMQNodeMemoryAlarm

Severity	Minor
Summary	RabbitMQ memory consumption is high.
Description	The RabbitMQ server {{ $labels.node }} in the {{ $labels.namespace }} namespace has low free memory.

OpenStack

This section describes the alerts available for the OpenStack services.

OpenStack services API

This section describes the alerts for the OpenStack services API.

- OpenstackServiceApiDown
- OpenstackServiceApiOutage

OpenstackServiceApiDown

Severity	Critical
Summary	The {{ $labels.service_name }} endpoint is not accessible.
Description	The OpenStack service {{ $labels.service_name }} {{ $labels.interface }} API is not accessible for the {{ $labels.endpoint_id }} endpoint.

OpenstackServiceApiOutage

Severity	Critical
Summary	OpenStack service {{ $labels.service_name }} API outage.
Description	The OpenStack service API is not accessible for all available {{ $labels.service_name }} endpoints in the OpenStack service catalog.

Cinder

This section lists the alerts for Cinder.

- CinderServiceDown
- CinderServiceOutage

- CinderServicesDownMajor
- CinderServicesDownMinor

CinderServiceDown

Severity	Minor
Summary	The {{ $labels.binary }} service is down.
Description	The {{ $labels.binary }} service on the {{ $labels.hostname }} node is down.

CinderServiceOutage

Severity	Critical
Summary	The {{ $labels.binary }} service outage.
Description	All {{ $labels.binary }} services are down.

CinderServicesDownMajor

Severity	Major
Summary	60% of {{ $labels.binary }} services are down.
Description	{{ $value }} {{ $labels.binary }} services (>= 60%) are down.

CinderServicesDownMinor

Severity	Minor

Summary	30% of {{ $labels.binary }} services are down.
Description	{{ $value }} {{ $labels.binary }} services (>= 30%) are down.

Ironic

This section lists the alerts for Ironic.

- IronicDriverMissing

IronicDriverMissing

Severity	Major
Summary	The ironic-conductor {{ $labels.driver }} back-end driver is missing.
Description	The {{ $labels.driver }} back-end driver of the ironic-conductor service is missing on the {{ $value }} node(s).

Neutron

This section lists the alerts for Neutron.

- NeutronAgentDown
- NeutronAgentsDownMajor
- NeutronAgentsDownMinor
- NeutronAgentsOutage

NeutronAgentDown

Severity	Minor
Summary	The {{ $labels.binary }} agent is down.
Description	The {{ $labels.binary }} agent on the {{ $labels.hostname }} node is down.

NeutronAgentsDownMajor

Severity	Major
Summary	60% of {{ $labels.binary }} agents are down.
Description	{{ $value }} {{ $labels.binary }} agents (>= 60%) are down.

NeutronAgentsDownMinor

Severity	Minor
Summary	30% of {{ $labels.binary }} agents are down.
Description	{{ $value }} {{ $labels.binary }} agents (>= 30%) are down.

NeutronAgentsOutage

Severity	Critical
Summary	{{ $labels.binary }} agents outage.
Description	All {{ $labels.binary }} agents are down.

Nova

This section lists the alerts for Nova.

- NovaComputeServicesDownMajor
- NovaComputeServicesDownMinor
- NovaServiceDown
- NovaServiceOutage
- NovaServicesDownMajor
- NovaServicesDownMinor

NovaComputeServicesDownMajor

Severity	Major

Summary	More than 50% of nova-compute services are down.
Description	More than 50% of nova-compute services are down.

NovaComputeServicesDownMinor

Severity	Minor
Summary	More than 25% of nova-compute services are down.
Description	More than 25% of nova-compute services are down.

NovaServiceDown

Severity	Minor
Summary	The {{ $labels.binary }} service is down.
Description	The {{ $labels.binary }} service on the {{ $labels.hostname }} node is down.

NovaServiceOutage

Severity	Critical
Summary	The {{ $labels.binary }} service outage.
Description	All {{ $labels.binary }} services are down.

NovaServicesDownMajor

Severity	Major
Summary	More than 60% of {{ $labels.binary }} services are down.
Description	More than 60% of {{ $labels.binary }} services are down.

NovaServicesDownMinor

Severity	Minor
Summary	30% of {{ $labels.binary }} services are down.
Description	More than 30% of {{ $labels.binary }} services are down.

Configure StackLight

This section describes how to configure StackLight in your Mirantis OpenStack on Kubernetes deployment and includes the description of OpenStack-related StackLight parameters and their verification. For other available configuration keys and their configuration verification, see Mirantis Container Cloud Operations Guide: Manage StackLight.

Configure StackLight

This section describes the StackLight configuration workflow.

To configure StackLight:

1. Obtain kubeconfig of the Mirantis Container Cloud management cluster and open the Cluster object manifest of the managed cluster for editing as described in the steps 1-7 in Mirantis Container Cloud Operations Guide: Configure StackLight.

2. In the following section of the opened manifest, configure the StackLight parameters as required:

   - For the OpenStack-based parameters, see StackLight configuration parameters.
   - For other parameters, see Mirantis Container Cloud Operations Guide: StackLight configuration parameters.

```
spec:
  providerSpec:
    value:
      helmReleases:
      - name: stacklight
        values:
```

3. Verify StackLight configuration depending on the modified parameters:

   - For OpenStack-related parameters, see Verify StackLight after configuration.
   - For other parameters, see Mirantis Container Cloud Operations Guide: Verify StackLight after configuration.

StackLight configuration parameters

This section describes the OpenStack-related StackLight configuration keys that you can specify in the values section to change StackLight settings as required. For other available configuration keys, see Mirantis Container Cloud Operations Guide: StackLight configuration parameters.

Prior to making any changes to StackLight configuration, perform the steps described in Configure StackLight. After changing StackLight configuration, verify the changes as described in Verify StackLight after configuration.

- OpenStack
- Ironic
- RabbitMQ
- Telegraf
- SSL certificates

OpenStack

Key	Description	Example values

openstack.enabled (bool)	Enables OpenStack monitoring. Set to true by default.	true or false
openstack.namespace (string)	Defines the namespace within which the OpenStack virtualized control plane is installed. Set to openstack by default.	openstack

Ironic

Key	Description	Example values

openstack.ironic.enabled (bool)	Enables Ironic monitoring. Set to false by default.	true or false

RabbitMQ

Key	Description	Example values

openstack.rabbitmq.credentialsConfig (map)	Defines the RabbitMQ credentials to use if credentials discovery is disabled or some required parameters were not found during the discovery.	credentialsConfig:
		  username: "stacklight"
		  password: "stacklight"
		  host: "rabbitmq.openstack.svc"
		  queue: "notifications"
		  vhost: "openstack"

openstack.rabbitmq.credentialsDiscovery (map)	Enables the credentials discovery to obtain the username and password from the secret object.	credentialsDiscovery:
		  enabled: true
		  namespace: openstack
		  secretName: os-rabbitmq-user-credentials

Telegraf

Key	Description	Example values

openstack.telegraf.credentialsConfig (map)	Specifies the OpenStack credentials to use if the credentials discovery is disabled or some required parameters were not found during the discovery.	credentialsConfig: identityEndpoint: "" # "http://keystone-api.openstack.svc:5000/v3" domain: "" # "default" password: "" # "workshop" project: "" # "admin" region: "" # "RegionOne" username: "" # "admin"

openstack.telegraf.credentialsDiscovery (map)	Enables the credentials discovery to obtain all required parameters from the secret object.	credentialsDiscovery:
		  enabled: true
		  namespace: openstack
		  secretName: keystone-keystone-admin

openstack.telegraf.interval (string)	Specifies the interval of metrics gathering from the OpenStack API. Set to 1m by default.	1m, 3m

openstack.telegraf.insecure (bool) Available since MOS Ussuri Update	Enables or disables the server certificate chain and host name verification. Set to true by default.	true or false

openstack.telegraf.skipPublicEndpoints (bool) Available since MOS Ussuri Update	Enables or disables HTTP probes for public endpoints from the OpenStack service catalog. Set to false by default, meaning that Telegraf verifies all endpoints from the OpenStack service catalog, including the public, admin, and internal endpoints.	true or false

SSL certificates

Key	Description	Example values
openstack.externalFQDN (string)	External FQDN used to communicate with OpenStack services. Used for certificates monitoring.	https://os.ssl.mirantis.net/

openstack.insecure (string)
Available since MOS Ussuri Update
	Defines whether to verify the trust chain of the OpenStack endpoint SSL certificates during monitoring.	**insecure**:
		  internal: true
		  external: false

Verify StackLight after configuration

This section describes how to verify StackLight after configuring its OpenStack-related parameters as described in Configure StackLight and StackLight configuration parameters. Perform the verification procedure for a particular modified StackLight key.

To verify StackLight after configuring other parameters, see Mirantis Container Cloud Operations Guide: Verify StackLight after configuration.

To verify StackLight after configuration:

Key	Verification procedure

• openstack.enabled • openstack.namespace	1. In the Grafana web UI, verify that the OpenStack dashboards are present and not empty. 2. In the Prometheus web UI, click Alerts and verify that the OpenStack alerts are present in the list of alerts.
openstack.ironic.enabled	1. In the Grafana web UI, verify that the Ironic dashboard is present and not empty. 2. In the Prometheus web UI, click Alerts and verify that the Ironic* alerts are present in the list of alerts.
• openstack.rabbitmq.credentialsConfig • openstack.rabbitmq.credentialsDiscovery	In the Kibana web UI, click Discover and verify that the audit-* and notifications-* indexes contain documents.
• openstack.telegraf.credentialsConfig • openstack.telegraf.credentialsDiscovery • openstack.telegraf.interval • openstack.telegraf.insecure Available since MOS Ussuri Update • openstack.telegraf.skipPublicEndpoints Available since MOS Ussuri Update	In the Grafana web UI, verify that the OpenStack dashboards are present and not empty.
• openstack.externalFQDN • openstack.externalFQDNs Available since MOS Ussuri Update	1. In the Prometheus web UI, navigate to Status > Targets. 2. Verify that the blackbox-external-endpoint target contains the configured domains (URLs).

Tungsten Fabric operations

The section covers the management aspects of a Tungsten Fabric cluster deployed on Kubernetes.

> Caution!
>
> Before you proceed with the Tungsten Fabric management, read through MOS Reference Architecture: Tungsten Fabric known limitations.

Update Tungsten Fabric

The Tungsten Fabric cluster update is performed during the MOS managed cluster release update through the Container Cloud web UI as described in Mirantis Container Cloud Operations Guide: Update a managed cluster.

Replace a failed TF controller node

If one of the Tungsten Fabric (TF) controller nodes has failed, follow this procedure to replace it with a new node.

To replace a TF controller node:

> Note
>
> Pods that belong to the failed node can stay in the Terminating state.

1. Delete the failed TF controller node from the Kubernetes cluster:

   ```
   kubectl delete node <FAILED-TF-CONTROLLER-NODE-NAME>
   ```

   > Note
   >
   > Once the failed node has been removed from the cluster, all pods that hanged in the Terminating state should be removed.

2. Assign the TF labels for the new control plane node as per the table below using the following command:

   ```
   kubectl label node <NODE-NAME> <LABEL-KEY=LABEL-VALUE> ...
   ```

Tungsten Fabric (TF) node roles

Node role	Description	Kubernetes labels	Minimal count
TF control plane	Hosts the TF control plane services such as database, messaging, api, svc, config.	tfconfig=enabled tfcontrol=enabled tfwebui=enabled tfconfigdb=enabled	3
TF analytics	Hosts the TF analytics services.	tfanalytics=enabled tfanalyticsdb=enabled	3
TF vRouter	Hosts the TF vRouter module and vRouter agent.	tfvrouter=enabled	Varies
TF vRouter DPDK Technical Preview	Hosts the TF vRouter agent in DPDK mode.	tfvrouter-dpdk=enabled	Varies

Note

TF supports only Kubernetes OpenStack workloads. Therefore, you should label OpenStack compute nodes with the tfvrouter=enabled label.

Note

Do not specify the openstack-gateway=enabled and openvswitch=enabled labels for the MOS deployments with TF as a networking back end for OpenStack.

3. Once you label the new Kubernetes node, new pods start scheduling on the node. Though, pods that use Persistent Volume Claims are stuck in the Pending state as their volume claims stay bounded to the local volumes from the deleted node. To resolve the issue:

    1. Delete the PersistentVolumeClaim (PVC) bounded to the local volume from the failed node:

    ```
    kubectl -n tf delete pvc <PVC-BOUNDED-TO-NON-EXISTING-VOLUME>
    ```

> Note
>
> Clustered services that use PVC, such as Cassandra, Kafka, and ZooKeeper, start the replication process when new pods move to the Ready state.

2. Delete the pod that is using the removed PVC:

```
kubectl -n tf delete pod <POD-NAME>
```

4. Verify that the pods have successfully started on the replaced controller node and stay in the Ready state.

Verify the status of Tungsten Fabric services

> Caution!
>
> This feature is available starting from MOS Ussuri Update.

This section describes how to verify the status of the Tungsten Fabric services using the TF Operator tf-status tool that resides in tf-tool-status containers among the services. Using this tool, you can also verify the status of the Cassandra, ZooKeeper, Kafka, Redis, and RabbitMQ third-party services supported by TF.

A tf-tool-status container provides access to the contrail-status Python script through HTTP. To verify the TF services status, use the following options:

- Run contrail-status.py directly from the tf-tool-status container console.
- Send an HTTP request to the Kubernetes node IP, using the tf-tool-status container port, which is 8888 by default. The tf-status pod is deployed as a DaemonSet and does not expose any Kubernetes service. For third-party services, the tool uses the tf-tool-status-party pod.
- Send an HTTP request to tf-tool-status-aggregator, a Kubernetes microservice that aggregates statuses from all hosts that handle at least one TF service or a third-party service.

> Caution!
>
> For third-party services, due to a limitation, tf-tool-status-aggregator may return an error. To obtain the status of a service pod, use direct access or send an HTTP request to the tf-tool-status container.

- Verify the status of TF services
- Verify the status of third-party services

To verify the status of the TF services:

Select from the following options:

- Run contrail-status.py directly from the console of the tf-tool-status container:

  ```
  kubectl -n tf exec <TF-TOOL-STATUS-POD> -- python /root/contrail-status.py
  ```

  For the debug mode, use the --debug flag:

  ```
  kubectl -n tf exec <TF-TOOL-STATUS-POD> -- python /root/contrail-status.py --debug
  ```

- Send an HTTP request to the tf-tool-status container:

  1. Obtain the list of nodes that handle tf-tool-status containers:

     ```
     kubectl -n tf get pod --selector app=tf-tool-status -o wide
     ```

  2. For the required host, obtain its port provided by the container:

     ```
     kubectl -n tf get pod <TF-TOOL-STATUS-POD> -o custom-columns=NAME:.metadata.name,HOST:.spec.nodeName,HOST-IP:.status.hostIP,PORT:.spec.containers[0].ports[0].hostPort
     ```

     Example of system response:

     ```
     NAME                HOST       HOST-IP      PORT
     tf-tool-status-xxxx   host_name   10.10.0.12   8888
     ```

  3. Using the obtained IP and port, request tf-tool-status from any node in the environment. For example:

     ```
     curl 10.10.0.12:8888

     # or

     curl 10.10.0.12:8888/json   # for output in JSON format

     # or

     curl 10.10.0.12:8888/debug  # for the debug mode
     ```

     This process may take a certain amount of time depending on the services on the host.

- Send an HTTP request to the tf-tool-status-aggregator Kubernetes service:

  1. Obtain the CLUSTER-IP address of the Kubernetes aggregator service:

```
kubectl -n tf get svc --selector app=tf-tool-status-aggregator -o wide
```

Example of system response:

```
NAME                        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
tf-tool-status-aggregator   ClusterIP   10.96.145.46   <none>        80/TCP    46h
```

The service from the example above is provided on port 80.

2. Obtain the status of the service. Substitute <CLUSTER-IP> with the obtained tf-tool-status-aggregator service CLUSTER-IP.

> Note
>
> For help with tf-tool-status-aggregator, use curl <CLUSTER-IP>.

- To obtain the list of tf-status pods and basic information about them:

```
curl <CLUSTER-IP>/pod-list
```

Example of system response:

```
Name: tf-tool-status-2h4pq
IP: 10.10.0.27
UUID: a2feb3d1-62d5-11ea-b302-02420a0a010b
HostIP: 10.10.0.27
Host: ps-ws-w2myzajhovir-1-xskgrqlroqob-server-hqrfdu5kxhcc
```

- To obtain an aggregated status of all TF services:

```
# shows aggregated statuses sorted by host/pod
curl <CLUSTER-IP>/status

# output in JSON format by host/pod
curl <CLUSTER-IP>/status/json

# the same as /status
curl <CLUSTER-IP>/status/node

# shows aggregated statuses sorted by services, such as config-api, webui
curl <CLUSTER-IP>/status/group
```

To verify the status of third-party services:

1. Obtain the IP address and port of the tf-tool-status containers for third-party services:

   ```
   kubectl -n tf get pod -l tungstenfabric=status-party -o custom-columns=NAME:.metadata.name,IP:.status.podIP,PORT:.spec.containers[0].ports[0].containerPort
   ```

   Example of system response:

   ```
   NAME                                IP            PORT
   tf-tool-status-party-7dfdb9dbb6-5pw8q   192.168.62.79   8080
   ```

2. Request the third-party service status. For example:

   ```
   curl 192.168.62.79:8080
   ```

Verify the Tungsten Fabric deployment

This section explains how to use the tungsten-pytest test set to verify your Tungsten Fabric (TF) deployment. The tungsten-pytest test set is part of the TF operator and allows for prompt verification of the Kubernetes objects related to TF and basic verification of the TF services.

To verify the TF deployment using tungsten-pytest:

1. Enable the tf-test controller in the TF Operator resource for the Operator to start the pod with the test set:

   ```
   spec:
     controllers:
       tf-test:
         tungsten-pytest:
           enabled: true
   ```

2. Wait until the tungsten-pytest pod is ready. To keep track of the tests execution and view the results, run:

   ```
   kubectl -n tf logs -l app=tf-test
   ```

3. Optional. The test results are stored in the tf-test-tungsten-pytest PVC. To obtain the results:

   1. Deploy the pod with the mounted volume:

      ```
      # Run pod and mount pvc to it
      cat <<EOF | kubectl apply -f -
      apiVersion: v1
      kind: Pod
      metadata:
        name: tf-test-results-pod
        namespace: tf
      spec:
        volumes:
      ```

```
    - name: tf-test-data-volume
      persistentVolumeClaim:
        claimName: tf-test-tungsten-pytest
  containers:
    - name: tempest-pvc-container
      image: ubuntu
      command: ['sh', '-c', 'sleep infinity']
      volumeMounts:
        - mountPath: "/tungsten-pytest/data"
          name: tf-test-data-volume
EOF
```

2. Copy the results locally:

```
kubectl -n tf cp tf-test-results-pod:tungsten-pytest/data/results.xml results.xml
```

3. Remove the Tempest test results pod:

```
kubectl -n tf delete pod tf-test-results-pod
```

4. Disable the tf-test controller:

```
kubectl -n tf patch tfoperator openstack-tf --type='json' -p='[{"op": "replace", "path": "/spec/controllers/tf-test/tungsten-pytest/enabled", "value": false}]'
```

5. Manually remove the pod with tests:

```
kubectl -n tf delete pod -l app=tf-test
```

Configure load balancing

This section describes a simple load balancing configuration. As an example, we use a topology for balancing the traffic between two HTTP servers listening on port 80. The example topology includes the following parameters:

- Back-end servers 10.10.0.4 and 10.10.0.3 in the private-subnet subnet run an HTTP application that listens on the TCP port 80.
- The public-subnet subnet is a shared external subnet created by the cloud operator and accessible from the Internet.
- The created load balancer is accessible through an IP address from the public subnet that will distribute web requests between the back-end servers.

To configure load balancing:

1. Log in to a keystone-client pod.
2. Create a load balancer:

```
openstack loadbalancer create --vip-subnet-id=private-subnet --name test-lb
```

3. Create an HTTP listener:

```
openstack loadbalancer listener create --name test-listener \
--protocol HTTP --protocol-port 80 test-lb
```

4. Create a LBaaS pool that will be used by default for test-listener:

```
openstack loadbalancer pool create  --protocol HTTP \
--lb-algorithm ROUND_ROBIN --name test-pool --listener test-listener
```

5. Create a health monitor that ensures health of the pool members:

```
openstack loadbalancer healthmonitor create --delay 5 --name test-hm \
--timeout 3 --max-retries 3 --type HTTP test-pool
```

6. Add back-end servers to the pool. The following example adds the 10.10.0.3 and 10.10.0.4 back-end servers:

```
openstack loadbalancer member create --address 10.10.0.3 --protocol-port 80 test-pool
openstack loadbalancer member create --address 10.10.0.4 --protocol-port 80 test-pool
```

7. Create a floating IP address in a public network and associate it with a port of the load balancer VIP:

```
vip_port_id=$(openstack loadbalancer show test-lb -c vip_port_id \
-f value)
fip_id=$openstack floating ip create public -c floating_ip_address \
-f value)
openstack floating ip set --port $vip_port_id $fip_id
```

8. All load balancer ports from the Tungsten Fabric (TF) side have secuirty_port_enabled to restrict access to the load balancer from outside. Disable secuirty_port_enabled:

    1. In the TF web UI, navigate to Configure > Networking > Ports.
    2. Find the load balancer ports and click the gear icon next to a load balancer that does not have neutron:LOADBALANCER in the Device column.
    3. Disable Security Groups and click Save.
    4. Repeat the steps 8.2 and 8.3 for the remaining load balancers that do not have neutron:LOADBALANCER in the Device column.

9. Access the VIP floating IP address and verify that requests are distributed between the two servers. For example:

```
curl http://10.11.12.103:80
Welcome to addr:10.10.10.4
```

```
curl http://10.11.12.103:80
Welcome to addr:10.10.10.3
```

In the example above, an HTTP application that runs on the back-end servers returns an IP address of the host on which it runs.

Enable DPDK for Tungsten Fabric

This section describes how to enable DPDK mode for the Tungsten Fabric (TF) vRouter.

> Note
>
> This feature is available as technical preview. Use such configuration for testing and evaluation purposes only.

> Caution!
>
> This feature is available starting from MOS Ussuri Update.

To enable DPDK for TF:

1. Install the required drivers on the host operating system. The vfio-pci, uio_pci_generic, or mlnx drivers can be used with the TF vRouter agent in DPDK mode. For details about DPDK drivers, see Linux Drivers.
2. Enable huge pages on the host as described in Mirantis Container Cloud Operations Guide: Enable huge pages in a host profile.
3. Mark the hosts for deployment with DPDK with the tfvrouter-dpdk=enabled label.
4. Open the TF Operator custom resource for editing:

   ```
   kubectl -n tf edit tfoperators.operator.tf.mirantis.com openstack-tf
   ```

5. Enable DPDK:

   ```
   spec:
     tf-vrouter:
       agent-dpdk:
         enabled: true
   ```

> Seealso
>
> Tungsten Fabric official documentation: DPDK vRouter

Back up TF databases

To back up the TF databases, use db_json_exim.py, located on the tf-config pods. This script will create a dump of Cassandra and ZooKeeper databases. Cassandra is a fault-tolerant and horizontally scalable database that provides persistent storage of configuration and analytics data. ZooKeeper is used by TF for allocation of unique object identifiers and transactions implementation.

To prevent data loss, Mirantis recommends that you simultaneously back up the ZooKeeper database dedicated to configuration services and the Cassandra database.

> Caution!
>
> The backup of database must be consistent across all systems because the state of the Tungsten Fabric databases is associated with other system databases, such as OpenStack databases.

To back up TF databases in JSON Format

1. Disable the Neutron server that is used by OpenStack to communicate with the Tungsten Fabric API:

   > Note
   >
   > The database changes associated with northbound APIs must be stopped on all systems before performing any backup operations.

   1. Scale the neutron-server deployment to 0 replicas:

      ```
      kubectl -n openstack scale deploy neutron-server --replicas 0
      ```

   2. Verify the number of replicas:

      ```
      kubectl -n openstack get deploy neutron-server
      ```

      Example of a positive system response:

      ```
      NAME            READY   UP-TO-DATE   AVAILABLE   AGE
      neutron-server  0/0     0            0           6d22h
      ```

2. Join the Tungsten Fabric API that is part of the config DaemonSet:

   1. Obtain the tf-config pod:

```
kubectl -n tf get pod -l tungstenfabric=config
```

Example of a system response:

```
NAME              READY   STATUS    RESTARTS   AGE
tf-config-6ppvc   5/5     Running   0          5d4h
tf-config-rgqqq   5/5     Running   0          5d4h
tf-config-sb4kk   5/5     Running   0          5d4h
```

2. Join the Bash shell of one of the api container from the previous step:

```
kubectl -n tf exec -it tf-config-<hash> -c api -- bash
```

Example of a system response:

```
(config-api[hostname])[<USER>@<HOSTNAME> /]$
```

3. Inside the api container, change the directory to the Python config management packages:

```
cd /usr/lib/python2.7/site-packages/cfgm_common
```

4. Back up data using db_json_exim in JSON format:

```
python db_json_exim.py --export-to /tmp/db-dump.json
```

5. Verify the created dump:

```
cat /tmp/db-dump.json | python -m json.tool | less
```

6. Copy the backup from the container:

```
kubectl -n tf cp tf-config-<hash>:/tmp/db-dump.json <DESTINATION-PATH-FOR-BACKUP>
```

7. Enable the Neutron server:

    1. Scale the neutron-server deployment back to the desired number of replicas. Default is 3.

    ```
    kubectl -n openstack scale deploy neutron-server --replicas <DESIRED-NUM-REPLICAS>
    ```

    2. Verify the number of replicas:

    ```
    kubectl -n openstack get deploy neutron-server
    ```

    Example of a system response:

```
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
neutron-server 3/3     3            3           6d23h
```

Seealso

Restore TF databases

Restore TF databases

This section describes how to restore the Cassandra and ZooKeeper TF databases from a db-dump file created as described in Back up TF databases.

Caution!

The backup of database must be consistent across all systems because the state of the Tungsten Fabric databases is associated with other system databases, such as OpenStack databases.

To restore TF databases:

1. Obtain the config API image repository and tag.

   ```
   kubectl -n tf get tfconfig tf-config -o=jsonpath='{.spec.api.containers[?(@.name=="api")].image}'
   ```

   From the output, copy the entire image link.

2. Terminate the configuration and analytics services and stop the database changes associated with northbound APIs on all systems.

   Note

   The TF operator watches related resources and keeps them updated and healthy. If any resource is deleted or changed, the TF Operator automatically runs reconciling to create a resource or change the configuration back to the desired state. Therefore, the TF Operator must not be running during the databases restoration.

   1. Scale the tungstenfabric-operator deployment to 0 replicas:

      ```
      kubectl -n tf scale deploy tungstenfabric-operator --replicas 0
      ```

   2. Verify the number of replicas:

```
kubectl -n tf get deploy tungstenfabric-operator
```

Example of a positive system response:

```
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
tungstenfabric-operator   0/0     0            0           10h
```

3. Delete the TF configuration and analytics daemonsets:

```
kubectl -n tf delete daemonset tf-config
kubectl -n tf delete daemonset tf-config-db
kubectl -n tf delete daemonset tf-analytics
kubectl -n tf delete daemonset tf-analytics-snmp
```

The TF configuration pods should be automatically terminated.

4. Verify that the TF configuration pods are terminated:

```
kubectl -n tf get pod -l app=tf-config
kubectl -n tf get pod -l tungstenfabric=analytics
kubectl -n tf get pod -l tungstenfabric=analytics-snmp
```

Example of a positive system response:

```
No resources found.
```

3. Stop Kafka:

1. Scale the kafka-operator deployment to 0 replicas:

```
kubectl -n tf scale deploy kafka-operator --replicas 0
```

2. Scale the tf-kafka statefulSet to 0 replicas:

```
kubectl -n tf scale sts tf-kafka --replicas 0
```

3. Verify the number of replicas:

```
kubectl -n tf get sts tf-kafka
```

Example of a positive system response:

```
NAME       READY   AGE
tf-kafka   0/0     10h
```

4. Stop and wipe the Cassandra database:

    1. Scale the cassandra-operator deployment to 0 replicas:

        ```
        kubectl -n tf scale deploy cassandra-operator --replicas 0
        ```

    2. Scale the tf-cassandra-config-dc1-rack1 statefulSet to 0 replicas:

        ```
        kubectl -n tf scale sts tf-cassandra-config-dc1-rack1 --replicas 0
        ```

    3. Verify the number of replicas:

        ```
        kubectl -n tf get sts tf-cassandra-config-dc1-rack1
        ```

        Example of a positive system response:

        ```
        NAME                          READY   AGE
        tf-cassandra-config-dc1-rack1   0/0     10h
        ```

    4. Delete Persistent Volume Claims (PVCs) for the Cassandra configuration pods:

        ```
        kubectl -n tf delete pvc -l app=cassandracluster,cassandracluster=tf-cassandra-config
        ```

        Once PVCs are deleted, the related Persistent Volumes are automatically released. The release process takes approximately one minute.

5. Stop and wipe the ZooKeeper database:

    1. Scale the zookeeper-operator deployment to 0 replicas:

        ```
        kubectl -n tf scale deploy zookeeper-operator --replicas 0
        ```

    2. Scale the tf-zookeeper statefulSet to 0 replicas:

        ```
        kubectl -n tf scale sts tf-zookeeper --replicas 0
        ```

    3. Verify the number of replicas:

        ```
        kubectl -n tf get sts tf-zookeeper
        ```

        Example of a positive system response:

        ```
        NAME          READY   AGE
        tf-zookeeper   0/0     10h
        ```

    4. Delete PVCs for the ZooKeeper configuration pods:

```
kubectl -n tf delete pvc -l app=tf-zookeeper
```

Once PVCs are deleted, the related Persistent Volumes are automatically released. The release process takes approximately one minute.

6. Restore the number of replicas to run Cassandra and ZooKeeper and restore the deleted PVCs.

   1. Restore the cassandra-operator deployment replicas:

      ```
      kubectl -n tf scale deploy cassandra-operator --replicas 1
      ```

   2. Restore the tf-cassandra-config-dc1-rack1 statefulSet replicas:

      ```
      kubectl -n tf scale sts tf-cassandra-config-dc1-rack1 --replicas 3
      ```

   3. Verify that Cassandra pods have been created and are running:

      ```
      kubectl -n tf get pod -l app=cassandracluster,cassandracluster=tf-cassandra-config
      ```

      Example of a positive system response:

      ```
      NAME                          READY   STATUS    RESTARTS   AGE
      tf-cassandra-config-dc1-rack1-0   1/1     Running   0          4m43s
      tf-cassandra-config-dc1-rack1-1   1/1     Running   0          3m30s
      tf-cassandra-config-dc1-rack1-2   1/1     Running   0          2m6s
      ```

   4. Restore the zookeeper-operator deployment replicas:

      ```
      kubectl -n tf scale deploy zookeeper-operator --replicas 1
      ```

   5. Restore the tf-zookeeper statefulSet replicas:

      ```
      kubectl -n tf scale sts tf-zookeeper --replicas 3
      ```

   6. Verify that ZooKeeper pods have been created and are running:

      ```
      kubectl -n tf get pod -l app=tf-zookeeper
      ```

      Example of a positive system response:

      ```
      NAME             READY   STATUS    RESTARTS   AGE
      tf-zookeeper-0   1/1     Running   0          3m23s
      tf-zookeeper-1   1/1     Running   0          2m56s
      tf-zookeeper-2   1/1     Running   0          2m20s
      ```

7. Restore the databases from the backup:

> Note
>
> Do not use the TF API container used for the backup file creation. In this case, a session with the Cassandra and ZooKeeper databases is created once the TF API service starts but the TF configuration services are stopped. The tools for the database backup and restore are available only in the TF configuration API container. Using the steps below, start a blind container based on the config-api image.

1. Deploy a pod using the configuration API image obtained in the first step:

```
# Run pod and mount pvc to it
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/psp: privileged
  labels:
    app: tf-restore-db
  name: tf-restore-db
  namespace: tf
spec:
  containers:
    - envFrom:
      - configMapRef:
        name: tf-rabbitmq-cfgmap
      - configMapRef:
        name: tf-zookeeper-cfgmap
      - configMapRef:
        name: tf-cassandra-cfgmap
      - configMapRef:
        name: tf-services-cfgmap
      - secretRef:
        name: tf-os-secret
    image: <PUT_LINK_TO_CONFIG_API_IMAGE_FROM_STEP_ABOVE>
    imagePullPolicy: Always
    name: api
    command: [ "sleep", "infinity" ]
    args: [ "while true; do sleep 30; done;" ]
  dnsPolicy: ClusterFirstWithHostNet
  enableServiceLinks: true
  hostNetwork: true
  priority: 0
  restartPolicy: Always
  serviceAccount: default
```

```
  serviceAccountName: default
EOF
```

2. Copy the database dump to the container:

```
kubectl cp <PATH_TO_DB_DUMP> tf/tf-restore-db:/tmp/db-dump.json
```

3. Join to the restored container and build the configuration files:

```
kubectl -n tf exec -it tf-restore-db -- bash
(config-api) $ ./entrypoint.sh
```

4. Restore the Cassandra database from the backup:

```
(config-api) $ cd /usr/lib/python2.7/site-packages/cfgm_common
(config-api) $ python db_json_exim.py --import-from /tmp/db-dump.json
```

5. Delete the restore container:

```
kubectl -n tf delete pod tf-restore-db
```

8. Restore the replica number to run Kafka:

1. Restore the kafka-operator deployment replicas:

```
kubectl -n tf scale deploy kafka-operator --replicas 1
```

Kafka operator should automatically restore the number of replicas of the appropriate StatefulSet.

2. Verify the number of replicas:

```
kubectl -n tf get sts tf-kafka
```

Example of a positive system response:

```
NAME      READY   AGE
tf-kafka  3/3     10h
```

9. Run TF Operator to restore the TF configuration and analytics services:

1. Restore the TF Operator deployment replica:

```
kubectl -n tf scale deploy tungstenfabric-operator --replicas 1
```

2. Verify that the TF Operator is running properly without any restarts:

```
kubectl -n tf get pod -l name=tungstenfabric-operator
```

3. Verify that the configuration pods have been automatically started:

```
kubectl -n tf get pod -l app=tf-config
kubectl -n tf get pod -l tungstenfabric=analytics
kubectl -n tf get pod -l tungstenfabric=analytics-snmp
```

10 Restart the tf-control services:
.

> Caution!
>
> To avoid network downtime, do not restart all pods simultaneously.

1. List the tf-control pods

```
kubectl -n tf get pods -l app=tf-control
```

2. Restart the tf-control pods one by one.

> Caution!
>
> Before restarting the tf-control pods:
>
> - Verify that the new pods are successfully spawned.
> - Verify that no vRouters are connected to only one tf-control pod that will be restarted.

```
kubectl -n tf delete pod tf-control-<hash>
```

Limitations

The section covers the limitations of Mirantis OpenStack for Kubernetes (MOS).

[3544] Due to a community issue, Kubernetes pods may occasionally not be rescheduled on the nodes that are in the NotReady state. As a workaround, manually reschedule the pods from the node in the NotReady state using the kubectl drain --ignore-daemonsets --force <node-uuid> command.