MOS Deployment 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

Mirantis OpenStack for Kubernetes (MOS) enables the operator to create, scale, update, and upgrade OpenStack deployments on Kubernetes through a declarative API.

The Kubernetes built-in features, such as flexibility, scalability, and declarative resource definition make MOS a robust solution.

Plan the deployment

The detailed plan of any Mirantis OpenStack for Kubernetes (MOS) deployment is determined on a per-cloud basis. For the MOS reference architecture and design overview, see MOS Reference Architecture.

Also, read through Mirantis Container Cloud Reference Architecture: Container Cloud bare metal as a MOS managed cluster is deployed on top of a baremetal-based Container Cloud management cluster.

Provision a Container Cloud bare metal management cluster

The bare metal management system enables the Infrastructure Operator to deploy Container Cloud on a set of bare metal servers. It also enables Container Cloud to deploy MOS managed clusters on bare metal servers without a pre-provisioned operating system.

To provision your bare metal management cluster, refer to Mirantis Container Cloud Deployment Guide: Deploy a baremetal-based management cluster

Create a MOS managed cluster

A MOS cluster is deployed as a Container Cloud managed cluster through the Container Cloud web UI. For a detailed procedure, refer to Mirantis Container Cloud Operations Guide: Create a managed cluster.

> Note
>
> Once you have created a MOS managed cluster, some StackLight alerts may raise as false-positive until you deploy the MOS OpenStack environment.

Configure host operating system

Before you proceed with the OpenStack deployment, you need to configure the host operating system to run OpenStack.

To calculate the required number of hosts, read through Hardware requirements.

Configure kernel modules

The OpenStack Nova and Neutron services may require additional kernel modules to be loaded for normal operation. This section instructs you on how to enable the required kernel modules.

To enable KVM

A kernel-based Virtual Machine (KVM) is a virtualization module in the Linux kernel that enables the kernel to function as a hypervisor. The KVM-based virtualization is recommended on compute hosts to provide high performance VMs. See OpenStack Train official documentation: Enable KVM for details.

> Note
>
> Some systems require that you enable the VT support in the system BIOS. To enable the VT support, refer to the BIOS guides of the manufacturer of your server motherboard.

Configure networking

Before you proceed with the networking configuration, read MOS Reference Architecture: Networking.

This section describes how to configure persistent networking on a host with 3 NICs using the Mirantis Container Cloud L2 Templates:

- eno1 is used as a PXE interface
- ens3f1 and ens3f2 are used for bond0

To configure networking:

1. Create subnets:

```
---
apiVersion: ipam.mirantis.com/v1alpha1
kind: Subnet
metadata:
  labels:
    kaas.mirantis.com/provider: baremetal
    kaas.mirantis.com/region: region-one
  name: storage-backend
  namespace: child-ns
spec:
```

```
  cidr: 10.12.0.0/24

---
apiVersion: ipam.mirantis.com/v1alpha1
kind: Subnet
metadata:
  labels:
    kaas.mirantis.com/provider: baremetal
    kaas.mirantis.com/region: region-one
  name: storage-frontend
  namespace: child-ns
spec:
  cidr: 10.12.1.0/24
```

2. Create the openstack-example-3nic L2 template:

> Caution!
>
> The bootstrapping engine automatically assigns an IP address to the PXE nic 0 NIC. To prevent the IP duplication during updates, do not assign the IP address manually.

```
apiVersion: ipam.mirantis.com/v1alpha1
kind: L2Template
metadata:
  labels:
    ipam/Cluster: child-cluster
    kaas.mirantis.com/provider: baremetal
    kaas.mirantis.com/region: region-one
  name: openstack-example-3nic
  namespace: child-ns
spec:
  autoIfMappingPrio:
  - provision
  - eno
  - ens
  - enp
  npTemplate: |-
    version: 2
    ethernets:
      {{nic 0}}:
        # IMPORTANT: Do not assign an IP address here explicitly
        # to prevent IP duplication issues. The IP will be assigned
        # automatically by the bootstrapping engine.
        # addresses: []
        match:
          macaddress: {{mac 0}}
        set-name: {{nic 0}}
```

```
    mtu: 1500
  {{nic 1}}:
    dhcp4: false
    dhcp6: false
    match:
      macaddress: {{mac 1}}
    set-name: {{nic 1}}
    mtu: 1500
  {{nic 2}}:
    dhcp4: false
    dhcp6: false
    match:
      macaddress: {{mac 2}}
    set-name: {{nic 2}}
    mtu: 1500
bonds:
  bond0:
    interfaces:
      - {{nic 1}}
      - {{nic 2}}
vlans:
  pr-floating:
    id: 403
    link: bond0
  stor-frontend:
    id: 404
    link: bond0
    addresses:
      - {{ip "stor-frontend:storage-frontend"}}
  stor-backend:
    id: 405
    link: bond0
    addresses:
      - {{ip "stor-backend:storage-backend"}}
```

Configure hosts files for a deployment with TF

TungstenFabric services use the /etc/hosts file as a source of information for services configuration and registration in the database.

> Caution!
>
> For a MOS with Tungsten Fabric deployment, the proper hosts file configuration is a requirement.

To configure the hosts file:

On each node that will be used for running TungstenFabric services, verify that the /etc/hosts file has an entry with a local hostname and appropriate IP address from the Management network according to MOS Reference Architecture: Networking. For example:

```
127.0.0.1 localhost
<management-ip> <node-hostname>
....
```

Add bare metal hosts to the MOS cluster

After you create a MOS managed cluster and configure operating systems for the bare metal hosts, proceed with adding the bare metal hosts to your deployment through the Mirantis Container Cloud web UI.

For the detailed procedure, refer to Mirantis Container Cloud Operations Guide: Add a bare metal host.

> Note
>
> To calculate the required number of hosts, read through Hardware requirements.

Add machines to the MOS cluster

After you add bare metal hosts to the MOS managed cluster, create Kubernetes machines in your cluster using the Mirantis Container Cloud web UI.

For the detailed procedure, refer to Mirantis Container Cloud Operations Guide: Add a machine.

When adding the machines, verify that you label the Kubernetes nodes according to the OpenStack node roles:

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

Deploy a Ceph cluster

Deploy Ceph in the same Kubernetes cluster as described in Mirantis Container Cloud Operations Guide: Add a Ceph cluster. For Ceph cluster limitations, see Mirantis Container Cloud Reference Architecture: Limitations.

> Caution!
>
> Production deployments support only Ceph with a host network. To enable it, verify that the following snippet is present in the KaaSCephCluster configuration:

```
network:
  hostNetwork: true
  clusterNet: 10.10.10.0/24
  publicNet: 10.10.11.0/24
```

An example configuration of pools from the KaasCephCluster object that includes OpenStack required pools for Image, Block Storage, and Compute services:

```
spec:
  pools:
    - default: true
      deviceClass: hdd
      name: kubernetes
      replicated:
        size: 2
      role: kubernetes
    - default: false
      deviceClass: hdd
      name: volumes
      replicated:
        size: 2
      role: volumes
    - default: false
      deviceClass: hdd
      name: vms
      replicated:
        size: 2
      role: vms
    - default: false
      deviceClass: hdd
      name: backup
      replicated:
        size: 2
      role: backup
```

```
  - default: false
    deviceClass: hdd
    name: images
    replicated:
      size: 2
    role: images
  - default: false
    deviceClass: hdd
    name: other
    replicated:
      size: 2
    role: other
```

When all pools are created, verify that an appropriate secret required for a successful deployment of the OpenStack services that rely on Ceph is created in the openstack-ceph-shared namespace:

```
kubectl -n openstack-ceph-shared get secrets openstack-ceph-keys
```

Example of a positive system response:

```
NAME                TYPE     DATA   AGE
openstack-ceph-keys   Opaque   7      36m
```

Deploy OpenStack

This section instructs you on how to deploy OpenStack on top of Kubernetes as well as how to troubleshoot the deployment and access your OpenStack environment after deployment.

Deploy an OpenStack cluster

This section instructs you on how to deploy OpenStack on top of Kubernetes using the OpenStack Controller and openstackdeployments.lcm.mirantis.com (OsDpl) CR.

To deploy an OpenStack cluster:

1. Verify that you have pre-configured the networking according to MOS Reference Architecture: Networking.
2. Verify that the TLS certificates that will be required for the OpenStack cluster deployment have been pre-generated.

   > Note
   >
   > The Transport Layer Security (TLS) protocol is mandatory on public endpoints.

   > Caution!
   >
   > To avoid certificates renewal with subsequent OpenStack updates during which additional services with new public endpoints may appear, we recommend using wildcard SSL certificates for public endpoints. For example, *.it.just.works, where it.just.works is a cluster public domain.

   The sample code block below illustrates how to generate a self-signed certificate for the it.just.works domain. The procedure presumes the cfssl and cfssljson tools are installed on the machine.

```
mkdir cert && cd cert

tee ca-config.json << EOF
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
          "client auth"
```

```
      ],
      "expiry": "8760h"
    }
   }
  }
}
EOF

tee ca-csr.json << EOF
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names":[{
    "C": "<country>",
    "ST": "<state>",
    "L": "<city>",
    "O": "<organization>",
    "OU": "<organization unit>"
  }]
}
EOF

cfssl gencert -initca ca-csr.json | cfssljson -bare ca

tee server-csr.json << EOF
{
    "CN": "*.it.just.works",
    "hosts":     [
        "*.it.just.works"
    ],
    "key":     {
        "algo": "rsa",
        "size": 2048
    },
    "names": [    {
        "C": "US",
        "L": "CA",
        "ST": "San Francisco"
    }]
}
EOF
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem --config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server
```

3. Configure the OsDpl resource depending on the needs of your deployment.

   OsDpl is a Kubernetes CR that describes the OpenStack cluster deployment. The resource is validated with the help of the OpenAPI v3 schema. For more information about the fields and their description, refer to MOS Reference Architecture: OpenStackDeployment resource.

4. If required, enable DPDK, huge pages, and other supported Telco features as described in Advanced OpenStack configuration (optional).

5. To the openstackdeployment object, add information about the TLS certificates:

   - ssl:public_endpoints:ca_cert - CA certificate content (ca.pem)
   - ssl:public_endpoints:api_cert - server certificate content (server.pem)
   - ssl:public_endpoints:api_key - server private key (server-key.pem)

6. Verify that the Load Balancer network does not overlap your corporate or internal Kubernetes networks, for example, Calico IP pools. Also, verify that the pool of Load

Balancer network is big enough to provide IP addresses for all Amphora VMs (loadbalancers).

If required, reconfigure the Octavia network settings using the following sample structure:

```
spec:
  services:
    load-balancer:
      octavia:
        values:
          octavia:
            settings:
              lbmgmt_cidr: "10.255.0.0/16"
              lbmgmt_subnet_start: "10.255.1.0"
              lbmgmt_subnet_end: "10.255.255.254"
```

7. Trigger the OpenStack deployment:

   ```
   kubectl apply -f openstackdeployment.yaml
   ```

8. Monitor the status of your OpenStack deployment:

   ```
   kubectl -n openstack get pods
   kubectl -n openstack describe osdpl osh-dev
   ```

9. Assess the current status of the OpenStack deployment using the status section output in the OsDpl resource:

   1. Get the OsDpl YAML file:

      ```
      kubectl -n openstack get osdpl osh-dev -o yaml
      ```

   2. Analyze the status output using the detailed description in MOS Reference Architecture: OpenStackDeployment resource: The Status elements.

10. Verify that the OpenStack cluster has been deployed:

    ```
    clinet_pod_name=$(kubectl -n openstack get pods -l application=keystone,component=client  | grep keystone-client | head -1 | awk '{print $1}')
    kubectl -n openstack exec -it $clinet_pod_name -- openstack service list
    ```

    Example of a positive system response:

    ```
    +----------------------------------+--------------+----------------+
    | ID                               | Name         | Type           |
    +----------------------------------+--------------+----------------+
    | 159f5c7e59784179b589f933bf9fc6b0 | cinderv3     | volumev3       |
    | 6ad762f04eb64a31a9567c1c3e5a53b4 | keystone     | identity       |
    | 7e265e0f37e34971959ce2dd9eafb5dc | heat         | orchestration  |
    | 8bc263babe9944cdb51e3b5981a0096b | nova         | compute        |
    | 9571a49d1fdd4a9f9e33972751125f3f | placement    | placement      |
    ```

```
| a3f9b25b7447436b85158946ca1c15e2 | neutron       | network        |
| af20129d67a14cadbe8d33ebe4b147a8 | heat-cfn      | cloudformation |
| b00b5ad18c324ac9b1c83d7eb58c76f5 | radosgw-swift | object-store   |
| b28217da1116498fa70e5b8d1b1457e5 | cinderv2      | volumev2       |
| e601c0749ce5425c8efb789278656dd4 | glance        | image          |
+----------------------------------+---------------+----------------+
```

Seealso

MOS Reference Architecture: Networking

Advanced OpenStack configuration (optional)

This section includes configuration information for available advanced Mirantis OpenStack for Kubernetes features that include DPDK with the Neutron OVS back end, huge pages, CPU pinning, and other Enhanced Platform Awareness (EPA) capabilities.

Enable DPDK with OVS

> Note
>
> Consider this section as part of Deploy an OpenStack cluster.

> 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.

This section instructs you on how to enable DPDK with the Neutron OVS back end.

To enable DPDK with OVS:

1. Verify that your deployment meets the following requirements:
   - The required drivers have been installed on the host operating system.

     Different Poll Mode Driver (PMD) types may require different kernel drivers to properly work with NIC. For more information about the DPDK drivers, read DPDK official documentation: Linux Drivers and Overview of Networking Drivers.
   - The DPDK NICs are not used on the host operating system.
   - The huge pages feature is enabled on the host. See Enable huge pages for OpenStack for details.
2. Enable DPDK in the OsDpl custom resource through the node specific overrides settings. For example:

```
spec:
  nodes:
    <NODE-LABEL>::<NODE-LABEL-VALUE>:
      features:
        neutron:
          dpdk:
            bridges:
            - ip_address: 10.12.2.80/24
              name: br-phy
```

```
      driver: igb_uio
      enabled: true
      nics:
      - bridge: br-phy
        name: nic01
        pci_id: "0000:05:00.0"
    tunnel_interface: br-phy
```

Seealso

- MOS Reference Architecture: Node-specific settings
- Open vSwitch official documentation: Open vSwitch with DPDK
- Container Cloud Operations Guide: IAM roles

Advanced configuration for OpenStack computes nodes

Note

Consider this section as part of Deploy an OpenStack cluster.

Caution!

This feature is available starting from MOS Ussuri Update.

The section describes how to perform advanced configuration for the OpenStack compute nodes. Such configuration can be required in some specific use cases, such as DPDK, SR-IOV, or huge pages features usage.

Enable huge pages for OpenStack

> Note
>
> Consider this section as part of Deploy an OpenStack cluster.

The huge pages OpenStack feature provides essential performance improvements for applications that are highly memory IO-bound. Huge pages should be enabled on a per compute node basis.

Since NUMATopologyFilter is enabled in a MOS deployment by default, to activate the feature, you need to enable huge pages on the host as described in Mirantis Container Cloud Operations Guide: Enable huge pages in a host profile.

> Note
>
> The multi-size huge pages are not fully supported by Kubernetes before 1.19. Therefore, define only one size in kernel parameters.

> Seealso
>
> OpenStack official documentation: Huge pages

Configure CPU isolation for an instance

> Note
>
> Consider this section as part of Deploy an OpenStack cluster.

CPU isolation allows for better performance of some HPC applications, such as Open vSwitch with DPDK. To configure CPU isolation, add the isolcpus parameter to the GRUB configuration. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash isolcpus=8-19"
```

Use the instruction from Mirantis Container Cloud Operations Guide: Enable huge pages in a host profile as an example procedure for GRUB parameters configuration.

> Seealso
>
> OpenStack official documentation: CPU topologies

Configure custom CPU topologies

> Note
>
> Consider this section as part of Deploy an OpenStack cluster.

The majority of CPU topologies features are activated by NUMATopologyFilter that is enabled by default. Such features do not require any further service configuration and can be used directly on a vanilla MOS deployment. The list of the CPU topologies features includes, for example:

- NUMA placement policies
- CPU pinning policies
- CPU thread pinning policies
- CPU topologies

To enable libvirt CPU pinning through the node-specific overrides in the OpenStackDeployment custom resource, use the following sample configuration structure:

```
spec:
  nodes:
    <NODE-LABEL>::<NODE-LABEL-VALUE>:
      services:
        compute:
          nova_compute:
            values:
              conf:
                nova:
                  compute:
                    cpu_dedicated_set: 2-17
                    cpu_shared_set: 18-47
```

> Seealso
>
> OpenStack official documentation: CPU topologies

Access OpenStack after deployment

This section contains the guidelines on how to access your MOS OpenStack environment.

Configure DNS to access OpenStack

The OpenStack services are exposed through the Ingress NGINX controller.

To configure DNS to access your OpenStack environment:

1. Obtain the external IP address of the Ingress service:

   ```
   kubectl -n openstack get services ingress
   ```

   Example of system response:

   ```
   NAME      TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)                                AGE
   ingress   LoadBalancer  10.96.32.97   10.172.1.101   80:34234/TCP,443:34927/TCP,10246:33658/TCP   4h56m
   ```

2. Select from the following options:

   - If you have a corporate DNS server, update your corporate DNS service and create appropriate DNS records for all OpenStack public endpoints.

     To obtain the full list of public endpoints:

     ```
     kubectl -n openstack get ingress -ocustom-columns=NAME:.metadata.name,HOSTS:spec.rules[*].host | awk '/cluster-fqdn/ {print $2}'
     ```

     Example of system response:

     ```
     barbican.it.just.works
     cinder.it.just.works
     cloudformation.it.just.works
     designate.it.just.works
     glance.it.just.works
     heat.it.just.works
     horizon.it.just.works
     keystone.it.just.works
     neutron.it.just.works
     nova.it.just.works
     novncproxy.it.just.works
     octavia.it.just.works
     placement.it.just.works
     ```

   - If you do not have a corporate DNS server, perform one of the following steps:

     - Add the appropriate records to /etc/hosts locally. For example:

       ```
       10.172.1.101 barbican.it.just.works
       10.172.1.101 cinder.it.just.works
       10.172.1.101 cloudformation.it.just.works
       10.172.1.101 designate.it.just.works
       10.172.1.101 glance.it.just.works
       10.172.1.101 heat.it.just.works
       10.172.1.101 horizon.it.just.works
       10.172.1.101 keystone.it.just.works
       10.172.1.101 neutron.it.just.works
       10.172.1.101 nova.it.just.works
       10.172.1.101 novncproxy.it.just.works
       ```

```
10.172.1.101 octavia.it.just.works
10.172.1.101 placement.it.just.works
```

- Deploy your DNS server on top of Kubernetes:

  1. Deploy a standalone CoreDNS server by including the following configuration into coredns.yaml:

```
apiVersion: lcm.mirantis.com/v1alpha1
kind: HelmBundle
metadata:
  name: coredns
  namespace: osh-system
spec:
  repositories:
  - name: hub_stable
    url: https://kubernetes-charts.storage.googleapis.com
  releases:
  - name: coredns
    chart: hub_stable/coredns
    version: 1.8.1
    namespace: coredns
    values:
      image:
        repository: mirantis.azurecr.io/openstack/extra/coredns
        tag: "1.6.9"
      isClusterService: false
      servers:
      - zones:
        - zone: .
          scheme: dns://
          use_tcp: false
        port: 53
        plugins:
        - name: cache
          parameters: 30
        - name: errors
        # Serves a /health endpoint on :8080, required for livenessProbe
        - name: health
        # Serves a /ready endpoint on :8181, required for readinessProbe
        - name: ready
        # Required to query kubernetes API for data
        - name: kubernetes
          parameters: cluster.local
        - name: loadbalance
          parameters: round_robin
        # Serves a /metrics endpoint on :9153, required for serviceMonitor
        - name: prometheus
          parameters: 0.0.0.0:9153
        - name: forward
          parameters: . /etc/resolv.conf
        - name: file
          parameters: /etc/coredns/it.just.works.db it.just.works
      serviceType: LoadBalancer
      zoneFiles:
      - filename: it.just.works.db
        domain: it.just.works
        contents: |
          it.just.works.          IN     SOA    sns.dns.icann.org. noc.dns.icann.org. 2015082541 7200 3600 1209600 3600
          it.just.works.          IN     NS     b.iana-servers.net.
          it.just.works.          IN     NS     a.iana-servers.net.
          it.just.works.          IN     A      1.2.3.4
          *.it.just.works.         IN     A     1.2.3.4
```

2. Update the public IP address of the Ingress service:

```
sed -i 's/1.2.3.4/10.172.1.101/' release/ci/30-coredns.yaml
kubectl apply -f release/ci/30-coredns.yaml
```

3. Verify that the DNS resolution works properly:

   1. Assign an external IP to the service:

```
kubectl -n coredns patch service coredns-coredns --type='json' -p='[{"op": "replace", "path": "/spec/ports", "value": [{"name": "udp-53", "port": 53, "protocol": "UDP", "targetPort": 53}]}]'
kubectl -n coredns patch service coredns-coredns --type='json' -p='[{"op": "replace", "path": "/spec/type", "value":"LoadBalancer"}]'
```

   2. Obtain the external IP address of CoreDNS:

```
kubectl -n coredns get service coredns-coredns
```

      Example of system response:

```
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
coredns-coredns   ClusterIP   10.96.178.21   10.172.1.102    53/UDP,53/TCP   25h
```

4. Point your machine to use the correct DNS. It is 10.172.1.102 in the example system response above.
5. If you plan to launch Tempest tests or use the OpenStack client from a keystone-client-XXX pod, verify that the Kubernetes built-in DNS service is configured to resolve your public FQDN records by adding your public domain to the stubdomain list. For example, to add the it.just.works domain:

```
kubectl -n kube-system get configmap kube-dns -o jsonpath='{.data.stubDomains}'
{"mirantis.net": ["172.18.208.44"],
 "it.just.works": ["10.96.178.21"]
}
```

Seealso

- Kubernetes official documentation: Customizing DNS Service
- Kubernetes official documentation: Ingress
- NGINX official documentation: NGINX controller

Access your OpenStack environment

This section explains how to access your OpenStack environment as the Admin user.

Before you proceed, verify that you can access the Kubernetes API and have privileges to read secrets from the openstack namespace in Kubernetes or you are able to exec to the pods in this namespace.

Access OpenStack using the Kubernetes built-in admin CLI

You can use the built-in admin CLI client and execute the openstack CLI commands from a dedicated pod deployed in the openstack namespace:

```
kubectl -n openstack exec \
  $(kubectl -n openstack get pod -l application=keystone,component=client -ojsonpath='{.items[*].metadata.name}') \
  -ti -- bash
```

This pod has python-openstackclient and all required plugins already installed. Also, this pod has cloud admin credentials stored as appropriate shell environment variables for the openstack CLI command to consume.

Access an OpenStack environment through Horizon

1. Configure the external DNS resolution for OpenStack services as described in Configure DNS to access OpenStack.
2. Obtain the password of the Admin user:

   ```
   kubectl -n openstack get secret keystone-keystone-admin -ojsonpath='{.data.OS_PASSWORD}' | base64 -d
   ```

3. Access Horizon through your browser using its public service. For example, https://horizon.it.just.works.

   To log in, specify the admin user name and default domain. If the OpenStack Identity service has been deployed with the OpenID Connect integration:

   1. From the Authenticate using drop-down menu, select OpenID Connect.
   2. Click Connect. You will be redirected to your identity provider to proceed with the authentication.

   > Note
   >
   > If OpenStack has been deployed with self-signed TLS certificates for public endpoints, you may get a warning about an untrusted certificate. To proceed, allow the connection.

Access OpenStack through CLI from your local machine

To be able to access your OpenStack environment using CLI, you need to set the required environment variables that are stored in an OpenStack RC environment file. You can either download a project-specific file from Horizon, which is the easiest way, or create an environment file.

To access OpenStack through CLI, select from the following options:

- Download and source the OpenStack RC file:

  1. Log in to Horizon as described in Access an OpenStack environment through Horizon.
  2. Download the openstackrc or clouds.yaml file from the Web interface.

3. On any shell from which you want to run OpenStack commands, source the environment file for the respective project.

- Create and source the OpenStack RC file:

1. Configure the external DNS resolution for OpenStack services as described in Configure DNS to access OpenStack.
2. Create a stub of the OpenStack RC file:

```
cat << EOF > openstackrc
export OS_PASSWORD=$(kubectl -n openstack get secret keystone-keystone-admin -ojsonpath='{.data.OS_PASSWORD}' | base64 -d)
export OS_USERNAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_PROJECT_DOMAIN_NAME=Default
export OS_REGION_NAME=RegionOne
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION="3"
EOF
```

3. Add the Keystone public endpoint to this file as the OS_AUTH_URL variable. For example, for the domain name used throughout this guide:

```
echo export OS_AUTH_URL=https://keystone.it.just.works >> openstackrc
```

4. Source the obtained data into the shell:

```
source <openstackrc>
```

Now, you can use the openstack CLI as usual. For example:

```
openstack user list
+----------------------------------+-----------------+
| ID                               | Name            |
+----------------------------------+-----------------+
| dc23d2d5ee3a4b8fae322e1299f7b3e6 | internal_cinder |
| 8d11133d6ef54349bd014681e2b56c7b | admin           |
+----------------------------------+-----------------+
```

> Note
>
> If OpenStack was deployed with self-signed TLS certificates for public endpoints, you may need to use the openstack CLI client with certificate validation disabled. For example:
>
> ```
> openstack --insecure user list
> ```

Troubleshoot an OpenStack deployment

This section provides the general debugging instructions for your OpenStack on Kubernetes deployment. Start your troubleshooting with the determination of the failing component that can include the OpenStack Operator, HelmBundle Controller, a particular pod or service.

> Note
>
> For Kubernetes cluster debugging and troubleshooting, refer to Kubernetes official documentation: Troubleshoot clusters and Docker Enterprise v3.0 documentation: Monitor and troubleshoot.

Debugging the HelmBundle controller

The HelmBundle controller is running in the following containers:

- tiller - runs Tiller, the server portion of Helm
- controller - contains processes that handle the HelmBundle object and deploy it though Tiller

Verify the Helm releases statuses

1. Download the Helm client to connect to Tiller:

   ```
   export HELM_VERSION=v2.13.1
   wget https://get.helm.sh/helm-${HELM_VERSION}-linux-amd64.tar.gz
   tar -xf helm-${HELM_VERSION}-linux-amd64.tar.gz
   mv linux-amd64/helm /usr/local/bin/helm
   ```

2. On the Kubernetes node, run the following command to set up port forwarding:

   ```
   kubectl port-forward -n osh-system helm-controller-0 44134:44134
   ```

3. Set up alias to use port forwarding:

   ```
   alias helm="helm --host=localhost:44134"
   ```

4. Verify the Helm releases statuses:

   ```
   helm list
   ```

   If a Helm release is not in the DEPLOYED state, obtain the details from the output of the following command:

   ```
   helm history --col-width 1000000 <release-name>
   ```

Verify the status of a HelmBundle release

To verify the status of a HelmBundle release:

```
kubectl -n osh-system get helmbundles openstack-operator -o yaml
```

Example of a system response:

```
status:
  releaseStatuses:
    openstack-operator:
      attempt: 1
      chart: openstack-controller/openstack-operator
      finishedAt: "2020-04-21T11:24:08Z"
      hash: caa6a811e5540333b4a02d74f7ec855460d97f81efdd69ed13d516256cbf9f81
      notes: ""
      revision: 1
      status: DEPLOYED
      success: true
      version: 0.1.9-55
```

Debugging the OpenStack Controller

The OpenStack Controller is running in several containers in the openstack-operator-xxxx pod in the osh-system namespace. For the full list of containers and their roles, refer to MOS Reference Architecture: Openstack controller containers.

To verify the status of the OpenStack Controller, run:

```
kubectl -n osh-system get pods
```

Example of a system response:

```
NAME                                READY   STATUS    RESTARTS   AGE
helm-controller-0                   2/2     Running   0          70m
local-volume-provisioner-2xsl5      1/1     Running   0          70m
local-volume-provisioner-fqdf9      1/1     Running   0          70m
local-volume-provisioner-ncclp      1/1     Running   0          70m
openstack-operator-7f4b5cf88c-4tj26 5/5     Running   0          70m
```

To verify the logs for the osdpl container, run:

```
kubectl -n osh-system logs -f <openstack-operator-xxxx> -c osdpl
```

The openstack-operator pod is missing

The openstack-operator pod is created by the openstack-operator HelmBundle object. If the OpenStack Controller pod is not present in the namespace, Debugging the HelmBundle controller.

Debugging the OsDpl CR

This section includes the ways to mitigate the most common issues with the OsDpl CR. We assume that you have already debugged the HelmBundle and OpenStack Controllers to rule out possible failures with these components as described in Debugging the HelmBundle controller and Debugging the OpenStack Controller.

The osdpl has DEPLOYED=false

Possible root cause: One or more HelmBundle releases have not been deployed successfully.

To determine if you are affected:

Verify the status of the osdpl object:

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

Example of a system response:

```
NAME     AGE   DEPLOYED   DRAFT
osh-dev   22h   false      false
```

To debug the issue:

1. Identify the failed release by assessing the status:children section in the OsDpl resource:
    1. Get the OsDpl YAML file:

        ```
        kubectl -n openstack get osdpl osh-dev -o yaml
        ```

    2. Analyze the status output using the detailed description in MOS Reference Architecture: OpenStackDeployment resource: The Status elements.
2. For further debugging, refer to Debugging the HelmBundle controller.

Some pods are stuck in Init

Possible root cause: MOS uses the Kubernetes entrypoint init container to resolve dependencies between objects. If the pod is stuck in Init:0/X, this pod may be waiting for its dependencies.

To debug the issue:

Verify the missing dependencies:

```
kubectl -n openstack logs -f placement-api-84669d79b5-49drw -c init
```

Example of a system response:

```
Entrypoint WARNING: 2020/04/21 11:52:50 entrypoint.go:72: Resolving dependency Job placement-ks-user in namespace openstack failed: Job Job placement-ks-user in namespace openstack is not completed yet .
Entrypoint WARNING: 2020/04/21 11:52:52 entrypoint.go:72: Resolving dependency Job placement-ks-endpoints in namespace openstack failed: Job Job placement-ks-endpoints in namespace openstack is not completed yet .
```

Some HelmBundles are not present

Possible root cause: some OpenStack services depend on Ceph. These services include OpenStack Image, OpenStack Compute, and OpenStack Block Storage. If the HelmBundle releases for these services are not present, the openstack-ceph-keys secret may be missing in the openstack-ceph-shared namespace.

To debug the issue:

Verify that the Ceph Controller has created the openstack-ceph-keys secret in the openstack-ceph-shared namespace:

```
kubectl -n openstack-ceph-shared get secrets openstack-ceph-keys
```

Example of a positive system response:

```
NAME                TYPE     DATA   AGE
openstack-ceph-keys   Opaque   7      23h
```

If the secret is not present, create one manually.

Deploy Tungsten Fabric

This section describes how to deploy Tungsten Fabric as a back end for networking for your MOS environment.

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

Tungsten Fabric deployment prerequisites

Before you proceed with the actual Tungsten Fabric (TF) deployment, verify that your deployment meets the following prerequisites:

1. Your MOS OpenStack cluster is deployed as described in Deploy OpenStack with the compute-tf preset enabled.
2. Your MOS OpenStack cluster uses the correct value of features:neutron:tunnel_interface in the openstackdeployment object. The TF Operator will consume this value through the shared secret and use it as a network interface from the underlay network to create encapsulated tunnels with the tenant networks.

   > Warning
   >
   > TF uses features:neutron:tunnel_interface to create the vhost0 virtual interface and transfers the IP configuration from the tunnel_interface to the virtual one. Therefore, plan this interface as a dedicated physical interface for TF overlay networks.

3. The Kubernetes nodes are labeled according to the TF node roles:

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.

Deploy Tungsten Fabric

Deployment of Tungsten Fabric (TF) is managed by the tungstenfabric-operator Helm resource in a respective MOS ClusterRelease.

To deploy TF:

1. Verify that you have completed all prerequisite steps as described in Tungsten Fabric deployment prerequisites.
2. Create tungstenfabric.yaml with the TF resource configuration. For example:

```
apiVersion: operator.tf.mirantis.com/v1alpha1
kind: TFOperator
metadata:
  name: openstack-tf
  namespace: tf
spec:
  settings:
    orchestrator: openstack
```

3. Trigger the TF deployment:

```
kubectl apply -f tungstenfabric.yaml
```

4. Verify that TF has been successfully deployed:

```
kubectl get pods -n tf
```

The successfully deployed TF services should appear in the Running status in the system response.

Access the Tungsten Fabric web UI

The Tungsten Fabric (TF) web UI allows for easy and fast TF resources configuration, monitoring, and debugging. You can access the TF web UI through either the Ingress service or the Kubernetes Service directly. TLS termination for the https protocol is performed through the Ingress service.

To access the TF web UI through Ingress:

1. Log in to a local machine running Ubuntu 18.04 where kubectl is installed.
2. Obtain and export kubeconfig of your managed cluster as described in Mirantis Container Cloud Operations Guide: Connect to a Container Cloud managed cluster.
3. Obtain the password of the Admin user:

```
kubectl -n openstack exec -it $(kubectl -n openstack get pod -l application=keystone,component=client -o jsonpath='{.items[0].metadata.name}') -- env | grep PASS
```

4. Obtain the external IP address of the Ingress service:

```
kubectl -n openstack get services ingress
```

   Example of system response:

```
NAME      TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)                                AGE
ingress   LoadBalancer  10.96.32.97   10.172.1.101   80:34234/TCP,443:34927/TCP,10246:33658/TCP   4h56m
```

> Note
>
> Do not use the EXTERNAL-IP value to directly access the TF web UI. Instead, use the FQDN from the list below.

5. Obtain the FQDN of tf-webui:

> Note
>
> The command below outputs all host names assigned to the TF web UI service. Use one of them.

```
kubectl -n tf get ingress tf-webui -o custom-columns=HOSTS:.spec.rules[*].host
```

6. Configure DNS to access the TF web UI host as described in Configure DNS to access OpenStack.
7. Use your favorite browser to access the TF web UI at https://<FQDN-WEBUI>.

Seealso

Troubleshoot access to the Tungsten Fabric web UI

Troubleshoot the Tungsten Fabric deployment

This section provides the general debugging instructions for your Tungsten Fabric (TF) on Kubernetes deployment.

Enable debug logs for the Tungsten Fabric services

To enable debug logging for the Tungsten Fabric (TF) services:

1. Open the TF custom resource for modification:

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

2. Specify the LOG_LEVEL variable with the SYS_DEBUG value for the required TF service. For example, for the config-api service:

   ```
   spec:
     controllers:
       tf-config:
         api:
           containers:
           - name: api
             env:
             - name: LOG_LEVEL
               value: SYS_DEBUG
   ```

Warning

After the TF custom resource modification, the pods related to the affected services will be restarted. This rule does not apply to the tf-vrouter-agent-<XXXXX> pods as their update strategy differs. Therefore, if you enable the debug logging for the services in a tf-vrouter-agent-<XXXXX> pod, restart this pod manually after you modify the custom resource.

Troubleshoot access to the Tungsten Fabric web UI

If you cannot access the Tungsten Fabric (TF) web UI service, verify that the FQDN of the TF web UI is resolvable on your PC by running one of the following commands:

```
host tf-webui.it.just.works
# or
ping tf-webui.it.just.works
# or
dig host tf-webui.it.just.works
```

All commands above should resolve the web UI domain name to the IP address that should match the EXTERNAL-IPs subnet dedicated to Kubernetes.

If the TF web UI domain name has not been resolved to the IP address, your PC is using a different DNS or the DNS does not contain the record for the TF web UI service. To resolve the issue, define the IP address of the Ingress service from the openstack namespace of Kubernetes in the hosts file of your machine. To obtain the Ingress IP address:

```
kubectl -n openstack get svc ingress -o custom-columns=HOSTS:.status.loadBalancer.ingress[*].ip
```

If the web UI domain name is resolvable but you still cannot access the service, verify the connectivity to the cluster.