Copyright notice

2020 Mirantis, Inc. All rights reserved.

This product is protected by U.S. and international copyright and intellectual property laws. No part of this publication may be reproduced in any written, electronic, recording, or photocopying form without written permission of Mirantis, Inc.

Mirantis, Inc. reserves the right to modify the content of this document at any time without prior notice. Functionality described in the document may not be available at the moment. The document contains the latest information at the time of publication.

Mirantis, Inc. and the Mirantis Logo are trademarks of Mirantis, Inc. and/or its affiliates in the United States and other countries. Third party trademarks, service marks, and names mentioned in this document are the properties of their respective owners.
Preface

This documentation provides information on how to use Mirantis products to deploy cloud environments. The information is for reference purposes and is subject to change.

Intended audience

This documentation is intended for deployment engineers, system administrators, and developers; it assumes that the reader is already familiar with network and cloud concepts.

Documentation history

The following table lists the released revisions of this documentation:

<table>
<thead>
<tr>
<th>Revision date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 30, 2018</td>
<td>Q1`18 GA</td>
</tr>
</tbody>
</table>
Introduction to the MCP deployment

MCP enables you to deploy MCP clusters, that include OpenStack environments and Kubernetes clusters, automatically through the MCP DriveTrain or using manual deployment procedures.

The manual deployment covered in this guide bases on four physical nodes. Three of these nodes are physical servers hosting the Salt Master node, the MaaS node, and OpenStack VCP. The fourth node is a compute host for the OpenStack environment instances.

See also

Minimum hardware requirements
Plan the deployment

The configuration of your MCP installation depends on the individual requirements the cloud environment should meet to serve various purposes. Therefore, the plan of any MCP deployment is discussed on a per-customer basis.

The outcome of this step are input parameters that are used to generate the Reclass metadata model by Cookiecutter.

See also

- Plan an OpenStack environment
- Plan a Kubernetes cluster
Create a project repository

An MCP cluster deployment configuration is stored in a Git repository created on a per-customer basis. This section instructs you on how to manually create and prepare your project repository for an MCP deployment.

Before you start this procedure, create a Git repository in your version control system, such as GitHub.

To create a project repository manually:

1. Log in to any computer.
2. Create an empty directory and change to that directory.
3. Initialize your project repository:
   
   ```
   git init
   ```

   Example of system response:
   
   ```
   Initialized empty Git repository in /Users/crh/Dev/mcpdoc/.git/
   ```

4. Add your repository to the directory you have created:
   
   ```
   git remote add origin <YOUR-GIT-REPO-URL>
   ```

5. Create the following directories for your deployment metadata model:
   
   ```
   mkdir -p classes/cluster
   mkdir nodes
   ```

6. Add the Reclass variable to your bash profile:
   
   ```
   vim ~/.bash_profile
   ```

   Example:
   
   ```
   export RECLASS_REPO=<PATH_TO_YOUR_DEV_DIRECTORY>
   ```

7. Log out and log back in.
8. Verify that your ~/.bash_profile is sourced:
   
   ```
   echo $RECLASS_REPO
   ```

   The command returns the content of the ~/.bash_profile file.
9. Add the Mirantis Reclass module to your repository as a submodule:
git submodule add https://github.com/Mirantis/reclass-system-salt-model ./classes/system/

System response:

Cloning into '<PATH_TO_YOUR_DEV_DIRECTORY>/classes/system'...
remote: Counting objects: 8923, done.
remote: Compressing objects: 100% (214/214), done.
remote: Total 8923 (delta 126), reused 229 (delta 82), pack-reused 8613
Receiving objects: 100% (8923/8923), 1.15 MiB | 826.00 KiB/s, done.
Resolving deltas: 100% (4482/4482), done.
Checking connectivity... done.

10 Update the submodule:

   • git submodule sync
   • git submodule update --init --recursive --remote

11 Add your changes to a new commit:

   • git add -A

12 Commit your changes:

   • git commit

13 Add your commit message.

   • Example of system response:

     [master (root-commit) 9466ada] Initial Commit
     2 files changed, 4 insertions(+)
     create mode 100644 .gitmodules
     create mode 160000 classes/system

14 Push your changes:

   • git push

15 Proceed to Create a deployment metadata model.

   •
Create local mirrors

During an MCP deployment or MCP cluster update, you can make use of local mirrors.

By default, MCP deploys local mirrors with packages in a Docker container on the DriveTrain nodes with GlusterFS volumes. MCP creates and manages mirrors with the help of Aptly, which runs in the container named aptly in the Docker Swarm mode cluster on the DriveTrain nodes, or cid0x in terms of Reclass model.

MCP provides a prebuilt mirror image that you can customize depending on the needs of your MCP deployment, as well as the flexibility to manually create local mirrors. Specifically, the usage of the prebuilt mirror image is essential in the case of an offline MCP deployment scenario.

Get the prebuilt mirror image

The prebuilt mirror image contains the Debian package mirror (Aptly), Docker images mirror (Registry), Python packages mirror (PyPi), Git repositories mirror, and mirror of Mirantis Ubuntu VM cloud images.

To get the prebuilt mirror image:

1. On http://images.mirantis.com, download the latest version of the prebuilt mirror VM in the mcp-offline-image-<MCP_version>.qcow2 format.
2. If required, customize the VM contents as described in Customize the prebuilt mirror image.
3. Proceed to Automated deployment of MCP DriveTrain.

See also

List of repositories

Customize the prebuilt mirror image

You can easily customize mirrored Aptly, Docker, and Git repositories by configuring contents of the mirror VM defined in the cicd/aptly.yml file of the Reclass model.

After you perform the customization, apply the changes to the Reclass model as described in Update mirror image.

To customize the Aptly repositories mirrors

You can either customize the already existing mirrors content or specify any custom mirror required by your MCP deployment:

• To customize existing mirror sources:

  The sources for existing mirrors can be configured to use different upstream.

  Each Aptly mirror specification includes parameters that define their source on the system level of the Reclass model as well distribution, components, key URL, and GPG keys. To customize a mirror content, redefine these parameters as required.
An example of the apt.mirantis.com mirror specification:

```
_param:
    apt_mk_version: stable
mirror_mirantis_openstack_xenial_extra_source: http://apt.mirantis.com/xenial/
mirror_mirantis_openstack_xenial_extra_distribution: ${_param:apt_mk_version}
mirror_mirantis_openstack_xenial_extra_components: extra
mirror_mirantis_openstack_xenial_extra_key_url: "http://apt.mirantis.com/public.gpg"
mirror_mirantis_openstack_xenial_extra_gpgkeys:
    - A76882D3
aptly:
    server:
        mirror:
            mirantis_openstack_xenial_extra:
                source: ${_param:mirror_mirantis_openstack_xenial_extra_source}
                distribution: ${_param:mirror_mirantis_openstack_xenial_extra_distribution}
                components: ${_param:mirror_mirantis_openstack_xenial_extra_components}
                architectures: amd64
                key_url: ${_param:mirror_mirantis_openstack_xenial_extra_key_url}
                gpgkeys: ${_param:mirror_mirantis_openstack_xenial_extra_gpgkeys}
                publisher:
                    component: extra
                    distributions:
                        - ubuntu-xenial/${_param:apt_mk_version}
```

Note
You can find all mirrors and their parameters that can be overridden in the aptly/server/mirror section of the Reclass System Model

• To add new mirrors, extend the aptly:server:mirror part of the model using the structure as shown in the example above

Note
The aptly:server:mirror:<REPO_NAME>:publisher parameter specifies how the custom repository will be published.

The example of a custom mirror specification:

```
aptly:
    server:
        mirror:
            my_custom_repo_main:
                source: http://my-custom-repo.com
```
To customize the Docker images mirrors

The Docker repositories are defined as an image list that includes a registry and name for each Docker image. Customize the list depending on the needs of your MCP deployment:

- Specify a different Docker registry for the existing image to be pulled from
- Add a new Docker image

Example of customization:

```
docker:
  client:
    registry:
      target_registry: apt:5000
      image:
        - registry: ""
          name: registry:2
        - registry: osixia
          name: openldap:1.1.8
        - registry: tcpcloud
          name: aptly-public:latest
```

Note

The target_registry parameter specifies which registry the images will be pushed into.

To customize the Git repositories mirrors:

The Git repositories are defined as a repository list that includes a name and URL for each Git repository. Customize the list depending on the needs of your MCP deployment.

Example of customization:

```
git:
  server:
    directory: /srv/git/
```
repos:
- name: gerritlib
  url: https://github.com/openstack-infra/gerritlib.git
- name: jeepyb
  url: https://github.com/openstack-infra/jeepyb.git

See also
Update mirror image

Create local mirrors manually

If you prefer to manually create local mirrors for your MCP deployment, check the List of repositories required for an installation of MCP.

To manually create a local mirror:

1. Log in to the Salt Master node.
2. Identify where the container with the aptly service is running in the Docker Swarm cluster.
   
   ```bash
   salt -C 'I@docker:swarm:role:master' cmd.run 'docker service ps aptly|head -n3'
   ```
3. Log in to the node where the container with the aptly service is running.
4. Open the console in the container with the aptly service:
   
   ```bash
   docker exec -it <CONTAINER_ID> bash
   ```
5. In the console, import the public key that will be used to fetch the repository.

Note
The public keys are typically available in the root directory of the repository and are called Release.key or Release.gpg. Also, you can download the public key from the key server keys.gnupg.net.

```bash
gpg --no-default-keyring --keyring trustedkeys.gpg --keyserver keys.gnupg.net \ --recv-keys <PUB_KEY_ID>
```

For example, for the apt.mirantis.com repository:

```bash
gpg --no-default-keyring --keyring trustedkeys.gpg --keyserver keys.gnupg.net \ --recv-keys 24008509A76882D3
```

6. Create a local mirror for the specified repository:

Note
You can find the list of repositories in the Repository planning section of the MCP Reference Architecture guide.

```bash
aptly mirror create <LOCAL_MIRROR_NAME> <REMOTE_REPOSITORY> <DISTRIBUTION>
```

For example, for the http://apt.mirantis.com/xenial repository:

```bash
aptly mirror create local.apt.mirantis.xenial http://apt.mirantis.com/xenial stable
```

7. Update a local mirror:

```bash
aptly mirror update <LOCAL_MIRROR_NAME>
```

For example, for the local.apt.mirantis.xenial local mirror:

```bash
aptly mirror update local.apt.mirantis.xenial
```

8. Verify that the local mirror has been created:

```bash
aptly mirror show <LOCAL_MIRROR_NAME>
```
For example, for the local.apt.mirantis.xenial local mirror:

```
aptly mirror show local.apt.mirantis.xenial
```

Example of system response:

```
Name: local.apt.mirantis.xenial
Status: In Update (PID 9167)
Archive Root URL: http://apt.mirantis.com/xenial/
Distribution: stable
Components: extra, mitaka, newton, oc31, oc311, oc32, oc323, oc40, oc666, ocata,
salt, salt-latest
Architectures: amd64
Download Sources: no
Download .udebs: no
Last update: never

Information from release file:
Architectures: amd64
Codename: stable
Components: extra mitaka newton oc31 oc311 oc32 oc323 oc40 oc666 ocata salt
salt-latest
Date: Mon, 28 Aug 2017 14:12:39 UTC
Description: Generated by aptly

Label: xenial stable
Origin: xenial stable
Suite: stable
```

9. In the Model Designer web UI, set the local_repositories parameter to True to enable using of local mirrors.

10. Add the local_repo_url parameter manually to classes/cluster/<cluster_name>/init.yml after . model generation.

See also

- Repository planning
- GitLab Repository Mirroring
- The aptly mirror
Create a deployment metadata model

In a Reclass metadata infrastructural model, the data is stored as a set of several layers of objects, where objects of a higher layer are combined with objects of a lower layer, that allows for as flexible configuration as required.

The MCP metadata model has the following levels:

- Service level includes metadata fragments for individual services that are stored in Salt formulas and can be reused in multiple contexts.
- System level includes sets of services combined in a such way that the installation of these services results in a ready-to-use system.
- Cluster level is a set of models that combine already created system objects into different solutions. The cluster module settings override any settings of service and system levels and are specific for each deployment.

The model layers are firmly isolated from each other. They can be aggregated on a south-north direction using service interface agreements for objects on the same level. Such approach allows reusing of the already created objects both on service and system levels.

Mirantis provides the following methods to create a deployment metadata model:

Create a deployment metadata model using the Model Designer UI

This section describes how to generate the cluster level metadata model for your MCP cluster deployment using the Model Designer UI. The tool used to generate the model is Cookiecutter, a command-line utility that creates projects from templates.

Note

The Model Designer web UI is only available within Mirantis. The Mirantis deployment engineers can access the Model Designer web UI using their Mirantis corporate username and password.

Alternatively, you can generate the deployment model manually as described in Create a deployment metadata model manually.

The workflow of a model creation includes the following stages:

1. Defining the model through the Model Designer web UI.
2. Tracking the execution of the model creation pipeline in the Jenkins web UI if required.
3. Obtaining the generated model to your email address or getting it published to the project repository directly.
As a result, you get a generated deployment model and can customize it to fit specific use-cases. Otherwise, you can proceed with the base infrastructure deployment.

Define the deployment model

This section instructs you on how to define the cluster level metadata model through the web UI using Cookiecutter. Eventually, you will obtain a generic deployment configuration that can be overridden afterwards.

To define the deployment model:

1. Log in to the web UI.
2. Go to Integration dashboard > Models > Model Designer.
3. Click Create Model. The Create Model page opens.
4. Configure your model by selecting a corresponding tab and editing as required:
   1. Configure General deployment parameters. Click Next.
   2. Configure Infrastructure related parameters. Click Next.
   3. Configure Product related parameters. Click Next.
5. Verify the model on the Output summary tab. Edit if required.
6. Click Confirm to trigger the Generate reclass cluster separated-products-auto Jenkins pipeline. If required, you can track the success of the pipeline execution in the Jenkins web UI.

If you selected the Send to e-mail address publication option on the General parameters tab, you will receive the generated model to the e-mail address you specified in the Publication options > Email address field on the Infrastructure parameters tab. Otherwise, the model will automatically be pushed to your project repository.

Seealso
- Create a project repository
- Publish the deployment model to a project repository

General deployment parameters

The tables in this section outline the general configuration parameters that you can define for your deployment model through the Model Designer web UI. Consult the Define the deployment model section for the complete procedure.

The General deployment parameters wizard includes the following sections:

- Basic deployment parameters cover basic deployment parameters
- Services deployment parameters define the platform you need to generate the model for
- Networking deployment parameters cover the generic networking setup for a dedicated management interface and two interfaces for the workload. The two interfaces for the workload are in bond and have tagged sub-interfaces for the Control plane (Control network/VLAN) and Data plane (Tenant network/VLAN) traffic. The PXE interface is not managed and is left to default DHCP from installation. Setups for the NFV scenarios are not covered and require manual configuration.

Basic deployment parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster name</td>
<td>cluster_name: deployment_name</td>
<td>The name of the cluster that will be used as cluster/&lt;cluster_name&gt;/ in the project directory structure</td>
</tr>
<tr>
<td>Cluster domain</td>
<td>cluster_domain: deploy-name.local</td>
<td>The name of the domain that will be used as part of the cluster FQDN</td>
</tr>
<tr>
<td>Public host</td>
<td>public_host: ${_param:openstack_proxy_address}</td>
<td>The name or IP address of the public endpoint for the deployment</td>
</tr>
<tr>
<td>Reclass repository</td>
<td>reclass_repository: <a href="https://github.com/Mirantis/mk-lab-salt-model.git">https://github.com/Mirantis/mk-lab-salt-model.git</a></td>
<td>The URL to your project Git repository containing your models</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default JSON output</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cookiecutter template URL</td>
<td>cookiecutter_template_url: <a href="mailto:git@github.com">git@github.com</a>:Mirantis/mk2x-cookiecutter-recass-model.git</td>
<td>The URL to the Cookiecutter template repository</td>
</tr>
<tr>
<td>Cookiecutter template branch</td>
<td>cookiecutter_template_branch: master</td>
<td>The branch of the Cookiecutter template repository to use, master by default. Use refs/tags/&lt;mcp_version&gt; to generate the model that corresponds to a specific MCP release version. For example, 2017.12. Other possible values include stable and testing.</td>
</tr>
<tr>
<td>Shared Reclass URL</td>
<td>shared_reclass_url: ssh://mcp-jenkins@gerrit.mcp.mirantis.net:29418/salt-models/recass-system.git</td>
<td>The URL to the shared system model to be used as a Git submodule for the MCP cluster</td>
</tr>
<tr>
<td>MCP version</td>
<td>mcp_version: stable</td>
<td>Version of MCP to use, stable by default. Enter the release version number, for example, 2017.12. Other possible values are: nightly, testing. For nightly, use cookiecutter_template_branch: master.</td>
</tr>
<tr>
<td>Cookiecutter template credentials</td>
<td>cookiecutter_template_credentials: gerrit</td>
<td>Credentials to Gerrit to fetch the Cookiecutter templates repository. The parameter is used by Jenkins</td>
</tr>
<tr>
<td>Deployment type</td>
<td>deployment_type: physical</td>
<td>The supported deployment types include: • Physical for the OpenStack platform • Physical and Heat for the Kubernetes platform</td>
</tr>
<tr>
<td>Publication method</td>
<td>publication_method: email</td>
<td>The method to obtain the template. Available options include: • Send to the e-mail address • Commit to repository</td>
</tr>
</tbody>
</table>
Platform

- `platform: openstack_enabled`
- `platform: kubernetes_enabled`

The platform to generate the model for:

- The OpenStack platform supports OpenContrail, StackLight LMA, Ceph, CI/CD, and OSS sub-clusters enablement. If the OpenContrail is not enabled, the model will define OVS as a network engine.
- The Kubernetes platform supports StackLight LMA and CI/CD sub-clusters enablement, OpenContrail networking, and presupposes Calico networking. To use the default Calico plugin, uncheck the OpenContrail enabled check box.

<table>
<thead>
<tr>
<th>StackLight version</th>
<th>1</th>
<th>Specifies the StackLight LMA version. Available values are 1 for legacy StackLight LMA and 2 for Prometheus-based StackLight LMA.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceph enabled</td>
<td>ceph_enabled: 'True'</td>
<td>Enables a Ceph sub-cluster.</td>
</tr>
<tr>
<td>CI/CD enabled</td>
<td>cicd_enabled: 'True'</td>
<td>Enables a CI/CD sub-cluster.</td>
</tr>
<tr>
<td>OSS enabled</td>
<td>oss_enabled: 'True'</td>
<td>Enables an OSS sub-cluster.</td>
</tr>
<tr>
<td>Benchmark node enabled</td>
<td>bmk_enabled: 'False'</td>
<td>Enables a benchmark node. False, by default.</td>
</tr>
</tbody>
</table>

Networking deployment parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS Server 01</td>
<td>dns_server01: 8.8.8.8</td>
<td>The IP address of the dns01 server</td>
</tr>
<tr>
<td>DNS Server 02</td>
<td>dns_server02: 8.8.4.4</td>
<td>The IP address of the dns02 server</td>
</tr>
<tr>
<td>Deploy network subnet</td>
<td>deploy_network_subnet: 10.0.0.0/24</td>
<td>The IP address of the deploy network with the network mask</td>
</tr>
<tr>
<td>Deploy network gateway</td>
<td>deploy_network_gateway: 10.0.0.1</td>
<td>The IP gateway address of the deploy network</td>
</tr>
<tr>
<td>Control network subnet</td>
<td>control_network_subnet: 10.0.1.0/24</td>
<td>The IP address of the control network with the network mask</td>
</tr>
</tbody>
</table>
| Tenant network subnet    | tenant_network_subnet: 10.0.2.0/24      | The IP address of the tenant network with the network mask
### Tenant network gateway

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant_network_gateway</td>
<td>tenant_network_gateway: 10.0.2.1</td>
<td>The IP gateway address of the tenant network</td>
</tr>
</tbody>
</table>

### Control VLAN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control_vlan</td>
<td>control_vlan: '10'</td>
<td>The Control plane VLAN ID</td>
</tr>
</tbody>
</table>

### Tenant VLAN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant_vlan</td>
<td>tenant_vlan: '20'</td>
<td>The Data plane VLAN ID</td>
</tr>
</tbody>
</table>

### Infrastructure related parameters

The tables in this section outline the infrastructure configuration parameters you can define for your deployment model through the Model Designer web UI. Consult the Define the deployment model section for the complete procedure.

The Infrastructure deployment parameters wizard includes the following sections:

- Salt Master
- Ubuntu MAAS
- Publication options
- OpenStack or Kuberbetes networking
- Ceph
- CI/CD
- OSS
- Repositories

#### Salt Master

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salt Master address</td>
<td>salt_master_address: 10.0.1.15</td>
<td>The IP address of the Salt Master node on the control network</td>
</tr>
<tr>
<td>Salt Master management address</td>
<td>salt_master_management_address: 10.0.1.15</td>
<td>The IP address of the Salt Master node on the management network</td>
</tr>
<tr>
<td>Salt Master hostname</td>
<td>salt_master_hostname: cfg01</td>
<td>The hostname of the Salt Master node</td>
</tr>
</tbody>
</table>

#### Ubuntu MAAS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAAS hostname</td>
<td>maas_hostname: mas01</td>
<td>The hostname of the MAAS virtual server</td>
</tr>
<tr>
<td>MAAS deploy address</td>
<td>maas_deploy_address: 10.0.0.16</td>
<td>The IP address of the MAAS node on the deploy network</td>
</tr>
</tbody>
</table>

#### Publication options
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email address</td>
<td>email_address: &lt;your-email&gt;</td>
<td>The email address where the generated Reclass model will be sent to</td>
</tr>
</tbody>
</table>

**OpenStack or Kubernetes networking**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack network engine</td>
<td>openstack_network_engine: opencontrail</td>
<td>Available options include opencontrail and ovs. NFV feature generation is experimental. The OpenStack Nova compute NFV req enabled parameter is for enabling Hugepages and CPU pinning without DPDK.</td>
</tr>
<tr>
<td>Kubernetes network engine</td>
<td>kubernetes_network_engine: opencontrail</td>
<td>Available options include calico and opencontrail. This parameter is set automatically. If you uncheck the OpenContrail enabled field in the General parameters section, the default Calico plugin is set as the Kubernetes networking.</td>
</tr>
</tbody>
</table>

**Ceph**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceph version</td>
<td>luminous</td>
<td>The Ceph version</td>
</tr>
<tr>
<td>Ceph OSD back end</td>
<td>bluestore</td>
<td>The OSD back-end type</td>
</tr>
</tbody>
</table>

**CI/CD**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenLDAP enabled</td>
<td>openldap_enabled: 'True'</td>
<td>Enables OpenLDAP authentication</td>
</tr>
</tbody>
</table>

**OSS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS CIS enabled</td>
<td>cis_enabled: 'True'</td>
<td>Enables the Cloud Intelligence Service</td>
</tr>
<tr>
<td>OSS Security Audit enabled</td>
<td>oss_security_audit_enabled: 'True'</td>
<td>Enables the Security Audit service</td>
</tr>
<tr>
<td>OSS Cleanup Service enabled</td>
<td>oss_cleanup_service_enabled: 'True'</td>
<td>Enables the Cleanup Service</td>
</tr>
<tr>
<td>OSS SFDC support enabled</td>
<td>oss_sfdc_support_enabled: 'True'</td>
<td>Enables synchronization of your SalesForce account with OSS</td>
</tr>
</tbody>
</table>

**Repositories**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local repositories</td>
<td>local_repositories: 'False'</td>
<td>If true, changes repositories URLs to local mirrors. The local_repo_url parameter should be added manually after model generation.</td>
</tr>
</tbody>
</table>

**Product related parameters**

The tables in this section outline the product configuration parameters including infrastructure, CI/CD, OpenContrail, OpenStack, Kubernetes, Stacklight LMA, and Ceph hosts details. You can configure your product infrastructure for the deployment model through the Model Designer web UI. Consult the Define the deployment model section for the complete procedure.

The Product deployment parameters wizard includes the following sections:

- Infrastructure product parameters
- CI/CD product parameters
- OSS parameters
- OpenContrail service parameters
- OpenStack product parameters
- Kubernetes product parameters
- StackLight LMA product parameters
- Ceph product parameters

**Infrastructure product parameters**

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infra kvm01 hostname</td>
<td>infra_kvm01_hostname: kvm01</td>
<td>The hostname of the first KVM node</td>
</tr>
<tr>
<td>Infra kvm01 control address</td>
<td>infra_kvm01_control_address: 10.0.1.24 1</td>
<td>The IP address of the first KVM node on the control network</td>
</tr>
<tr>
<td>Infra kvm01 deploy address</td>
<td>infra_kvm01_deploy_address: 10.0.0.24</td>
<td>The IP address of the first KVM node on the management network</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>Infra kvm02 hostname</td>
<td>infra_kvm02_hostname: kvm02</td>
<td>The hostname of the second KVM node</td>
</tr>
<tr>
<td>Infra kvm02 control address</td>
<td>infra_kvm02_control_address: 10.0.1.24</td>
<td>The IP address of the second KVM node on the control network</td>
</tr>
<tr>
<td>Infra kvm02 deploy address</td>
<td>infra_kvm02_deploy_address: 10.0.0.24</td>
<td>The IP address of the second KVM node on the management network</td>
</tr>
<tr>
<td>Infra kvm03 hostname</td>
<td>infra_kvm03_hostname: kvm03</td>
<td>The hostname of the third KVM node</td>
</tr>
<tr>
<td>Infra kvm03 control address</td>
<td>infra_kvm03_control_address: 10.0.1.24</td>
<td>The IP address of the third KVM node on the control network</td>
</tr>
<tr>
<td>Infra kvm03 deploy address</td>
<td>infra_kvm03_deploy_address: 10.0.0.24</td>
<td>The IP address of the third KVM node on the management network</td>
</tr>
<tr>
<td>Infra KVM VIP address</td>
<td>infra_kvm_vip_address: 10.0.1.240</td>
<td>The virtual IP address of the KVM cluster</td>
</tr>
<tr>
<td>Infra deploy NIC</td>
<td>infra_deploy_nic: eth0</td>
<td>The NIC used for PXE of the KVM hosts</td>
</tr>
<tr>
<td>Infra primary first NIC</td>
<td>infra_primary_first_nic: eth1</td>
<td>The first NIC in the KVM bond</td>
</tr>
<tr>
<td>Infra primary second NIC</td>
<td>infra_primary_second_nic: eth2</td>
<td>The second NIC in the KVM bond</td>
</tr>
<tr>
<td>Infra bond mode</td>
<td>infra_bond_mode: active-backup</td>
<td>The bonding mode for the KVM nodes. Available options include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• active-backup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balance-xor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• broadcast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 802.3ad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balance-ltb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balance-alb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To decide which bonding mode best suits the needs of your deployment, you can consult the official Linux bonding documentation.</td>
</tr>
</tbody>
</table>
OpenStack compute count

openstack_compute_count: '100'

The number of compute nodes to be generated. The naming convention for compute nodes is cmp000 - cmp$\{\text{openstack}\_\text{compute}\_\text{count}\}$. If the value is 100, for example, the host names for the compute nodes expected by Salt include cmp000, cmp001, ..., cmp100.

### CI/CD product parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI/CD control node01 address</td>
<td>cicd_control_node01_address: 10.0.1.91</td>
<td>The IP address of the first CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control node01 hostname</td>
<td>cicd_control_node01_hostname: cid01</td>
<td>The hostname of the first CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control node02 address</td>
<td>cicd_control_node02_address: 10.0.1.92</td>
<td>The IP address of the second CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control node02 hostname</td>
<td>cicd_control_node02_hostname: cid02</td>
<td>The hostname of the second CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control node03 address</td>
<td>cicd_control_node03_address: 10.0.1.93</td>
<td>The IP address of the third CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control node03 hostname</td>
<td>cicd_control_node03_hostname: cid03</td>
<td>The hostname of the third CI/CD control node</td>
</tr>
<tr>
<td>CI/CD control VIP address</td>
<td>cicd_control_vip_address: 10.0.1.90</td>
<td>The virtual IP address of the CI/CD control cluster</td>
</tr>
<tr>
<td>CI/CD control VIP hostname</td>
<td>cicd_control_vip_hostname: cid</td>
<td>The hostname of the CI/CD control cluster</td>
</tr>
</tbody>
</table>

### OSS parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS address</td>
<td>oss_address: ${_param:stacklight_monitor_address}</td>
<td>VIP address of the OSS cluster</td>
</tr>
<tr>
<td>OSS node01 address</td>
<td>oss_node01_address: ${_param:stacklight_monitor01_address}</td>
<td>The IP address of the first OSS node</td>
</tr>
<tr>
<td>OSS node02 address</td>
<td>oss_node02_address: ${_param:stacklight_monitor02_address}</td>
<td>The IP address of the second OSS node</td>
</tr>
<tr>
<td>OSS node03 address</td>
<td>oss_node03_address: ${_param:stacklight_monitor03_address}</td>
<td>The IP address of the third OSS node</td>
</tr>
<tr>
<td>OSS OpenStack auth URL</td>
<td>oss_openstack_auth_url: <a href="http://172.17.16.190:5000/v3">http://172.17.16.190:5000/v3</a></td>
<td>OpenStack auth URL for OSS tools</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>OSS OpenStack username</td>
<td>oss_openstack_username: admin</td>
<td>Username for access to OpenStack</td>
</tr>
<tr>
<td>OSS OpenStack password</td>
<td>oss_openstack_password: nova</td>
<td>Password for access to OpenStack</td>
</tr>
<tr>
<td>OSS OpenStack project</td>
<td>oss_openstack_project: admin</td>
<td>OpenStack project name</td>
</tr>
<tr>
<td>OSS OpenStack domain ID</td>
<td>oss_openstack_domain_id: default</td>
<td>OpenStack domain ID</td>
</tr>
<tr>
<td>OSS OpenStack SSL verify</td>
<td>oss_openstack_ssl_verify: 'False'</td>
<td>OpenStack SSL verification mechanism</td>
</tr>
<tr>
<td>OSS OpenStack certificate</td>
<td>oss_openstack_cert: &quot;&quot;</td>
<td>OpenStack plain CA certificate</td>
</tr>
<tr>
<td>OSS OpenStack credentials path</td>
<td>oss_openstack_credentials_path: /srv/volumes/rundeck/storage</td>
<td>OpenStack credentials path</td>
</tr>
<tr>
<td>OSS OpenStack endpoint type</td>
<td>oss_openstack_endpoint_type: public</td>
<td>Interface type of OpenStack endpoint for service connections</td>
</tr>
<tr>
<td>OSS Rundeck external datasource enabled</td>
<td>oss_rundeck_external_datasource_enabled: False</td>
<td>Enabled external datasource (PostgreSQL) for Rundeck</td>
</tr>
<tr>
<td>OSS Rundeck forward iframe</td>
<td>rundeck_forward_iframe: False</td>
<td>Forward iframe of Rundeck through proxy</td>
</tr>
<tr>
<td>OSS Rundeck iframe host</td>
<td>rundeck_iframe_host: ${_param:openstack_proxy_address}</td>
<td>IP address for Rundeck configuration for proxy</td>
</tr>
<tr>
<td>OSS Rundeck iframe port</td>
<td>rundeck_iframe_port: ${_param:haproxy_rundeck_exposed_port}</td>
<td>Port for Rundeck through proxy</td>
</tr>
<tr>
<td>OSS Rundeck iframe ssl</td>
<td>rundeck_iframe_ssl: False</td>
<td>Secure Rundeck iframe with SSL</td>
</tr>
<tr>
<td>OSS webhook from</td>
<td>oss_webhook_from: TEXT</td>
<td>Required. Notification email sender.</td>
</tr>
<tr>
<td>OSS webhook recipients</td>
<td>oss_webhook_recipients: TEXT</td>
<td>Required. Notification email recipients.</td>
</tr>
<tr>
<td>OSS Pushkin SMTP host</td>
<td>oss_pushkin_smtp_host: 127.0.0.1</td>
<td>The address of SMTP host for alerts notifications</td>
</tr>
<tr>
<td>OSS Pushkin SMTP port</td>
<td>oss_pushkin_smtp_port: 587</td>
<td>The address of SMTP port for alerts notifications</td>
</tr>
<tr>
<td>Feature</td>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OSS notification SMTP with TLS</td>
<td><code>oss_pushkin_smtp_use_tls: 'True'</code></td>
<td>Enable using of the SMTP server under TLS (for alert notifications)</td>
</tr>
<tr>
<td>OSS Pushkin email sender password</td>
<td><code>oss_pushkin_email_sender_password: password</code></td>
<td>The sender-mail password for alerts notifications</td>
</tr>
<tr>
<td>SFDC auth URL</td>
<td>N/A</td>
<td>Authentication URL for the Salesforce service. For example, <code>sfdc_auth_url: https://login.salesforce.com/services/oauth2/token</code></td>
</tr>
<tr>
<td>SFDC username</td>
<td>N/A</td>
<td>Username for logging in to the Salesforce service. For example, <code>sfdc_username: user@example.net</code></td>
</tr>
<tr>
<td>SFDC password</td>
<td>N/A</td>
<td>Password for logging in to the Salesforce service. For example, <code>sfdc_password: secret</code></td>
</tr>
<tr>
<td>SFDC consumer key</td>
<td>N/A</td>
<td>Consumer Key in Salesforce required for Open Authorization (OAuth). For example, <code>sfdc_consumer_key: example_consumer_key</code></td>
</tr>
<tr>
<td>SFDC consumer secret</td>
<td>N/A</td>
<td>Consumer Secret from Salesforce required for OAuth. For example, <code>sfdc_consumer_secret: example_consumer_secret</code></td>
</tr>
<tr>
<td>SFDC organization ID</td>
<td>N/A</td>
<td>Salesforce Organization ID in Salesforce required for OAuth. For example, <code>sfdc_organization_id: example_organization_id</code></td>
</tr>
<tr>
<td>SFDC environment ID</td>
<td><code>sfdc_environment_id: 0</code></td>
<td>The cloud ID in Salesforce</td>
</tr>
<tr>
<td>SFDC Sandbox enabled</td>
<td><code>sfdc_sandbox_enabled: True</code></td>
<td>Sandbox environments are isolated from production Salesforce clouds. Enable sandbox to use it for testing and evaluation purposes. Verify that you specify the correct sandbox-url value in the <code>sfdc_auth_url</code> parameter. Otherwise, set the parameter to False.</td>
</tr>
<tr>
<td>OSS CIS username</td>
<td>oss_cis_username: ${param:oss_openstack_username}</td>
<td>CIS username</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>OSS CIS password</td>
<td>oss_cis_password: ${param:oss_openstack_password}</td>
<td>CIS password</td>
</tr>
<tr>
<td>OSS CIS OpenStack auth URL</td>
<td>oss_cis_os_auth_url: ${param:oss_openstack_auth_url}</td>
<td>CIS OpenStack authentication URL</td>
</tr>
<tr>
<td>OSS CIS OpenStack endpoint type</td>
<td>oss_cis_endpoint_type: ${param:oss_openstack_endpoint_type}</td>
<td>CIS OpenStack endpoint type</td>
</tr>
<tr>
<td>OSS CIS project</td>
<td>oss_cis_project: ${param:oss_openstack_project}</td>
<td>CIS OpenStack project</td>
</tr>
<tr>
<td>OSS CIS domain ID</td>
<td>oss_cis_domain_id: ${param:oss_openstack_domain_id}</td>
<td>CIS OpenStack domain ID</td>
</tr>
<tr>
<td>OSS CIS certificate</td>
<td>oss_cis_cacert: ${param:oss_openstack_cacert}</td>
<td>OSS CIS certificate</td>
</tr>
<tr>
<td>OSS CIS jobs repository</td>
<td>oss_cis_jobs_repository: <a href="https://github.com/Mirantis/rundeck-cis-jobs.git">https://github.com/Mirantis/rundeck-cis-jobs.git</a></td>
<td>CIS jobs repository</td>
</tr>
<tr>
<td>OSS CIS jobs repository branch</td>
<td>oss_cis_jobs_repository_branch: master</td>
<td>CIS jobs repository branch</td>
</tr>
<tr>
<td>OSS Security Audit username</td>
<td>oss_security_audit_username: ${param:oss_openstack_username}</td>
<td>Security audit service username</td>
</tr>
<tr>
<td>OSS Security Audit password</td>
<td>oss_security_audit_password: ${param:oss_openstack_password}</td>
<td>Security Audit service password</td>
</tr>
<tr>
<td>OSS Security Audit auth URL</td>
<td>name: oss_security_audit_os_auth_url: ${param:oss_openstack_auth_url}</td>
<td>Security Audit service authentication URL</td>
</tr>
<tr>
<td>OSS Security Audit project</td>
<td>oss_security_audit_project: ${param:oss_openstack_project}</td>
<td>Security Audit project name</td>
</tr>
<tr>
<td>OSS Security Audit user domain ID</td>
<td>oss_security_audit_user_domain_id: ${param:oss_openstack_domain_id}</td>
<td>Security Audit user domain ID</td>
</tr>
<tr>
<td>OSS Security Audit project domain ID</td>
<td>oss_security_audit_project_domain_id: ${param:oss_openstack_domain_id}</td>
<td>Security Audit project domain ID</td>
</tr>
<tr>
<td>OSS Security Audit OpenStack credentials path</td>
<td>oss_security_audit_os_credentials_path: ${param:oss_openstack_credentials_path}</td>
<td>Path to credentials for OpenStack cloud for the Security Audit service</td>
</tr>
<tr>
<td>OSS Cleanup service OpenStack credentials path</td>
<td>oss_cleanup_service_os_credentials_path: ${param:oss_openstack_credentials_path}</td>
<td>Path to credentials for OpenStack cloud for the Cleanup service</td>
</tr>
<tr>
<td>OSS Cleanup service username</td>
<td>oss_cleanup_username: ${param:oss_openstack_username}</td>
<td>Cleanup service username</td>
</tr>
<tr>
<td>OSS Cleanup service password</td>
<td>oss_cleanup_password: ${_param:oss_openstack_password}</td>
<td>Cleanup service password</td>
</tr>
<tr>
<td>OSS Cleanup service auth URL</td>
<td>oss_cleanu_service_os_auth_url: ${_param:oss_openstack_auth_url}</td>
<td>Cleanup service authentication URL</td>
</tr>
<tr>
<td>OSS Cleanup service project</td>
<td>oss_cleanup_project: ${_param:oss_openstack_project}</td>
<td>Cleanup service project name</td>
</tr>
<tr>
<td>OSS Cleanup service project domain ID</td>
<td>oss_cleanup_project_domain_id: ${_param:oss_openstack_domain_id}</td>
<td>Cleanup service project domain ID</td>
</tr>
</tbody>
</table>

### OpenContrail service parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenContrail analytics address</td>
<td>opencontrail_analytics_address: 10.0.1.30</td>
<td>The virtual IP address of the OpenContrail analytics cluster</td>
</tr>
<tr>
<td>OpenContrail analytics hostname</td>
<td>opencontrail_analytics_hostname: nal</td>
<td>The hostname of the OpenContrail analytics cluster</td>
</tr>
<tr>
<td>OpenContrail analytics node01 address</td>
<td>opencontrail_analytics_node01_address: 10.0.1.31</td>
<td>The virtual IP address of the first OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail analytics node01 hostname</td>
<td>opencontrail_analytics_node01_hostname: nal01</td>
<td>The hostname of the first OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail analytics node02 address</td>
<td>opencontrail_analytics_node02_address: 10.0.1.32</td>
<td>The virtual IP address of the second OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail analytics node02 hostname</td>
<td>opencontrail_analytics_node02_hostname: nal02</td>
<td>The hostname of the second OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail analytics node03 address</td>
<td>opencontrail_analytics_node03_address: 10.0.1.33</td>
<td>The virtual IP address of the third OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail analytics node03 hostname</td>
<td>opencontrail_analytics_node03_hostname: nal03</td>
<td>The hostname of the second OpenContrail analytics node on the control network</td>
</tr>
<tr>
<td>OpenContrail control address</td>
<td>opencontrail_control_address: 10.0.1.20</td>
<td>The virtual IP address of the OpenContrail control cluster</td>
</tr>
<tr>
<td>OpenContrail control hostname</td>
<td>opencontrail_control_hostname: ntw</td>
<td>The hostname of the OpenContrail control cluster</td>
</tr>
<tr>
<td>OpenContrail control node01 address</td>
<td>opencontrail_control_node01_address: 10.0.1.21</td>
<td>The virtual IP address of the first OpenContrail control node on the control network</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OpenContrail control node01 hostname</td>
<td>opencontrail_control_node01_hostname: ntw01</td>
<td>The hostname of the first OpenContrail control node on the control network</td>
</tr>
<tr>
<td>OpenContrail control node02 address</td>
<td>opencontrail_control_node02_address: 10.0.1.22</td>
<td>The virtual IP address of the second OpenContrail control node on the control network</td>
</tr>
<tr>
<td>OpenContrail control node02 hostname</td>
<td>opencontrail_control_node02_hostname: ntw02</td>
<td>The hostname of the second OpenContrail control node on the control network</td>
</tr>
<tr>
<td>OpenContrail control node03 address</td>
<td>opencontrail_control_node03_address: 10.0.1.23</td>
<td>The virtual IP address of the third OpenContrail control node on the control network</td>
</tr>
<tr>
<td>OpenContrail control node03 hostname</td>
<td>opencontrail_control_node03_hostname: ntw03</td>
<td>The hostname of the third OpenContrail control node on the control network</td>
</tr>
<tr>
<td>OpenContrail router01 address</td>
<td>opencontrail_router01_address: 10.0.1.100</td>
<td>The IP address of the first OpenContrail gateway router for BGP</td>
</tr>
<tr>
<td>OpenContrail router01 hostname</td>
<td>opencontrail_router01_hostname: rtr01</td>
<td>The hostname of the first OpenContrail gateway router for BGP</td>
</tr>
<tr>
<td>OpenContrail router02 address</td>
<td>opencontrail_router02_address: 10.0.1.101</td>
<td>The IP address of the second OpenContrail gateway router for BGP</td>
</tr>
<tr>
<td>OpenContrail router02 hostname</td>
<td>opencontrail_router02_hostname: rtr02</td>
<td>The hostname of the second OpenContrail gateway router for BGP</td>
</tr>
</tbody>
</table>

**OpenStack product parameters**

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute primary first NIC</td>
<td>compute_primary_first_nic: eth1</td>
<td>The first NIC in the OpenStack compute bond</td>
</tr>
<tr>
<td>Compute primary second NIC</td>
<td>compute_primary_second_nic: eth2</td>
<td>The second NIC in the OpenStack compute bond</td>
</tr>
<tr>
<td>Compute bond mode</td>
<td>compute_bond_mode: active-backup</td>
<td>The bond mode for the compute nodes</td>
</tr>
<tr>
<td>OpenStack compute rack01 hostname</td>
<td>openstack_compute_rack01_hostname: cmp</td>
<td>The compute hostname prefix</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>OpenStack compute rack01 single subnet</td>
<td>openstack_compute_rack01_single_subnet: 10.0.0.1</td>
<td>The Control plane network prefix for compute nodes</td>
</tr>
<tr>
<td>OpenStack compute rack01 tenant subnet</td>
<td>openstack_compute_rack01_tenant_subnet: 10.0.2</td>
<td>The data plane network prefix for compute nodes</td>
</tr>
<tr>
<td>OpenStack control address</td>
<td>openstack_control_address: 10.0.1.10</td>
<td>The virtual IP address of the control cluster on the control network</td>
</tr>
<tr>
<td>OpenStack control hostname</td>
<td>openstack_control_hostname: ctl</td>
<td>The hostname of the VIP control cluster</td>
</tr>
<tr>
<td>OpenStack control node01 address</td>
<td>openstack_control_node01_address: 10.0.1.11</td>
<td>The IP address of the first control node on the control network</td>
</tr>
<tr>
<td>OpenStack control node01 hostname</td>
<td>openstack_control_node01_hostname: ctl01</td>
<td>The hostname of the first control node</td>
</tr>
<tr>
<td>OpenStack control node02 address</td>
<td>openstack_control_node02_address: 10.0.1.12</td>
<td>The IP address of the second control node on the control network</td>
</tr>
<tr>
<td>OpenStack control node02 hostname</td>
<td>openstack_control_node02_hostname: ctl02</td>
<td>The hostname of the second control node</td>
</tr>
<tr>
<td>OpenStack control node03 address</td>
<td>openstack_control_node03_address: 10.0.1.13</td>
<td>The IP address of the third control node on the control network</td>
</tr>
<tr>
<td>OpenStack control node03 hostname</td>
<td>openstack_control_node03_hostname: ctl03</td>
<td>The hostname of the third control node</td>
</tr>
<tr>
<td>OpenStack database address</td>
<td>openstack_database_address: 10.0.1.50</td>
<td>The virtual IP address of the database cluster on the control network</td>
</tr>
<tr>
<td>OpenStack database hostname</td>
<td>openstack_database_hostname: dbs</td>
<td>The hostname of the VIP database cluster</td>
</tr>
<tr>
<td>OpenStack database node01 address</td>
<td>openstack_database_node01_address: 10.0.1.51</td>
<td>The IP address of the first database node on the control network</td>
</tr>
<tr>
<td>OpenStack database node01 hostname</td>
<td>openstack_database_node01_hostname: dbs01</td>
<td>The hostname of the first database node</td>
</tr>
<tr>
<td>OpenStack database node02 address</td>
<td>openstack_database_node02_address: 10.0.1.52</td>
<td>The IP address of the second database node on the control network</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>OpenStack database node02 hostname</td>
<td>openstack_database_node02_hostname: dbs02</td>
<td>The hostname of the second database node</td>
</tr>
<tr>
<td>OpenStack database node03 address</td>
<td>openstack_database_node03_address: 10.0.1.53</td>
<td>The IP address of the third database node on the control network</td>
</tr>
<tr>
<td>OpenStack database node03 hostname</td>
<td>openstack_database_node03_hostname: dbs03</td>
<td>The hostname of the third database node</td>
</tr>
<tr>
<td>OpenStack message queue address</td>
<td>openstack_message_queue_address: 10.0.1.40</td>
<td>The virtual IP address of the message queue cluster on the control network</td>
</tr>
<tr>
<td>OpenStack message queue hostname</td>
<td>openstack_message_queue_hostname: msg</td>
<td>The hostname of the VIP message queue cluster</td>
</tr>
<tr>
<td>OpenStack message queue node01 address</td>
<td>openstack_message_queue_node01_address: 10.0.1.41</td>
<td>The IP address of the first message queue node on the control network</td>
</tr>
<tr>
<td>OpenStack message queue node01 hostname</td>
<td>openstack_message_queue_node01_hostname: msg01</td>
<td>The hostname of the first message queue node</td>
</tr>
<tr>
<td>OpenStack message queue node02 address</td>
<td>openstack_message_queue_node02_address: 10.0.1.42</td>
<td>The IP address of the second message queue node on the control network</td>
</tr>
<tr>
<td>OpenStack message queue node02 hostname</td>
<td>openstack_message_queue_node02_hostname: msg02</td>
<td>The hostname of the second message queue node</td>
</tr>
<tr>
<td>OpenStack message queue node03 address</td>
<td>openstack_message_queue_node03_address: 10.0.1.43</td>
<td>The IP address of the third message queue node on the control network</td>
</tr>
<tr>
<td>OpenStack message queue node03 hostname</td>
<td>openstack_message_queue_node03_hostname: msg03</td>
<td>The hostname of the third message queue node</td>
</tr>
<tr>
<td>OpenStack benchmark node01 address</td>
<td>openstack_benchmark_node01_address: 10.0.1.95</td>
<td>The IP address of a benchmark node on the control network</td>
</tr>
<tr>
<td>OpenStack benchmark node01 hostname</td>
<td>openstack_benchmark_node01_hostname: bmk01</td>
<td>The hostname of a benchmark node</td>
</tr>
</tbody>
</table>
### OpenStack proxy

<table>
<thead>
<tr>
<th>OpenStack proxy address</th>
<th>openstack_proxy_address: 10.0.1.80</th>
<th>The virtual IP address of a proxy cluster on the control network</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack proxy hostname</td>
<td>openstack_proxy_hostname: prx</td>
<td>The hostname of the VIP proxy cluster</td>
</tr>
<tr>
<td>OpenStack proxy node01 address</td>
<td>openstack_proxy_node01_address: 10.0.1.81</td>
<td>The IP address of the first proxy node on the control network</td>
</tr>
<tr>
<td>OpenStack proxy node01 hostname</td>
<td>openstack_proxy_node01_hostname: prx01</td>
<td>The hostname of the first proxy node</td>
</tr>
<tr>
<td>OpenStack proxy node02 address</td>
<td>openstack_proxy_node02_address: 10.0.1.82</td>
<td>The IP address of the second proxy node on the control network</td>
</tr>
<tr>
<td>OpenStack proxy node02 hostname</td>
<td>openstack_proxy_node02_hostname: prx02</td>
<td>The hostname of the second proxy node</td>
</tr>
<tr>
<td>OpenStack version</td>
<td>openstack_version: mitaka</td>
<td>The version of OpenStack to be deployed</td>
</tr>
</tbody>
</table>

### Kubernetes product parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calico cni image</td>
<td>artifactfactory.mirantis.com/docker-prod-local/mirantis/projectcalico/calico/cni:latest</td>
<td>The Calico image with CNI binaries</td>
</tr>
<tr>
<td>Calico enable nat</td>
<td>calico_enable_nat: 'True'</td>
<td>If selected, NAT will be enabled for Calico</td>
</tr>
<tr>
<td>Calico image</td>
<td>artifactfactory.mirantis.com/docker-prod-local/mirantis/projectcalico/calico/node:latest</td>
<td>The Calico image</td>
</tr>
<tr>
<td>Calico netmask</td>
<td>16</td>
<td>The netmask of the Calico network</td>
</tr>
<tr>
<td>Calico network</td>
<td>192.168.0.0</td>
<td>The network that is used for the Kubernetes containers</td>
</tr>
<tr>
<td>Calicoctl image</td>
<td>artifactfactory.mirantis.com/docker-prod-local/mirantis/projectcalico/calico/ctl:latest</td>
<td>The image with the calicoctl command</td>
</tr>
<tr>
<td>etcd SSL</td>
<td>etcd_ssl: 'True'</td>
<td>If selected, the SSL for etcd will be enabled</td>
</tr>
<tr>
<td>Hyperkube image</td>
<td>artifactfactory.mirantis.com/docker-prod-local/mirantis/kubernetes/hyperkube-amd64:v1.4.6-6</td>
<td>The Kubernetes image</td>
</tr>
<tr>
<td>Kubernetes virtlet enabled</td>
<td>False</td>
<td>Optional. Disabled by default. Virtlet enables Kubernetes to run virtual machines. For the enablement details, see Enable Virtlet.</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Kubernetes externaldns enabled</td>
<td>False</td>
<td>If selected, ExternalDNS will be deployed. For details, see: Deploy ExternalDNS for Kubernetes.</td>
</tr>
<tr>
<td>Kubernetes compute node01 hostname</td>
<td>cmp01</td>
<td>The hostname of the first Kubernetes compute node</td>
</tr>
<tr>
<td>Kubernetes compute node01 deploy address</td>
<td>10.0.0.101</td>
<td>The IP address of the first Kubernetes compute node</td>
</tr>
<tr>
<td>Kubernetes compute node01 single address</td>
<td>10.0.1.101</td>
<td>The IP address of the first Kubernetes compute node on the Control plane</td>
</tr>
<tr>
<td>Kubernetes compute node01 tenant address</td>
<td>10.0.2.101</td>
<td>The tenant IP address of the first Kubernetes compute node</td>
</tr>
<tr>
<td>Kubernetes compute node02 hostname</td>
<td>cmp02</td>
<td>The hostname of the second Kubernetes compute node</td>
</tr>
<tr>
<td>Kubernetes compute node02 deploy address</td>
<td>10.0.0.102</td>
<td>The IP address of the second Kubernetes compute node on the deploy network</td>
</tr>
<tr>
<td>Kubernetes compute node02 single address</td>
<td>10.0.1.102</td>
<td>The IP address of the second Kubernetes compute node on the control plane</td>
</tr>
<tr>
<td>Kubernetes control address</td>
<td>10.0.1.10</td>
<td>The Keepalived VIP of the Kubernetes control nodes</td>
</tr>
<tr>
<td>Kubernetes control node01 address</td>
<td>10.0.1.11</td>
<td>The IP address of the first Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node01 deploy address</td>
<td>10.0.0.11</td>
<td>The IP address of the first Kubernetes controller node on the deploy network</td>
</tr>
<tr>
<td>Kubernetes control node01 hostname</td>
<td>ctl01</td>
<td>The hostname of the first Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node01 tenant address</td>
<td>10.0.2.11</td>
<td>The tenant IP address of the first Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node02 address</td>
<td>10.0.1.12</td>
<td>The IP address of the second Kubernetes controller node</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>Kubernetes control node02 deploy address</td>
<td>10.0.0.12</td>
<td>The IP address of the second Kubernetes control node on the deploy network</td>
</tr>
<tr>
<td>Kubernetes control node02 hostname</td>
<td>ctl02</td>
<td>The hostname of the second Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node02 tenant address</td>
<td>10.0.2.12</td>
<td>The tenant IP address of the second Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node03 address</td>
<td>10.0.1.13</td>
<td>The IP address of the third Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node03 tenant address</td>
<td>10.0.2.13</td>
<td>The tenant IP address of the third Kubernetes controller node</td>
</tr>
<tr>
<td>Kubernetes control node03 deploy address</td>
<td>10.0.0.13</td>
<td>The IP address of the third Kubernetes control node on the deploy network</td>
</tr>
<tr>
<td>Kubernetes control node03 hostname</td>
<td>ctl03</td>
<td>The hostname of the third Kubernetes controller node</td>
</tr>
<tr>
<td>OpenContrail public ip range</td>
<td>10.151.0.0/16</td>
<td>The public floating IP pool for OpenContrail</td>
</tr>
<tr>
<td>OpenContrail private ip range</td>
<td>10.150.0.0/16</td>
<td>The range of private OpenContrail IPs used for pods</td>
</tr>
<tr>
<td>Kubernetes keepalived vip interface</td>
<td>ens4</td>
<td>The Kubernetes interface used for the Keepalived VIP</td>
</tr>
</tbody>
</table>

StackLight LMA product parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackLight LMA log address</td>
<td>stacklight_log_address: 10.167.4.60</td>
<td>The virtual IP address of the StackLight LMA logging cluster</td>
</tr>
<tr>
<td>StackLight LMA log hostname</td>
<td>stacklight_log_hostname: log</td>
<td>The hostname of the StackLight LMA logging cluster</td>
</tr>
<tr>
<td>StackLight LMA log node01 address</td>
<td>stacklight_log_node01_address: 10.167.4.61</td>
<td>The IP address of the first StackLight LMA logging node</td>
</tr>
<tr>
<td>StackLight LMA log node01 hostname</td>
<td>stacklight_log_node01_hostname: log01</td>
<td>The hostname of the first StackLight LMA logging node</td>
</tr>
<tr>
<td>StackLight LMA log node02 address</td>
<td>stacklight_log_node02_address: 10.167.4.62</td>
<td>The IP address of the second StackLight LMA logging node</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>StackLight LMA log node02 hostname</td>
<td>stacklight_log_node02_hostname: log02</td>
<td>The hostname of the second StackLight LMA logging node</td>
</tr>
<tr>
<td>StackLight LMA log node03 address</td>
<td>stacklight_log_node03_address: 10.167.4.63</td>
<td>The IP address of the third StackLight LMA logging node</td>
</tr>
<tr>
<td>StackLight LMA log node03 hostname</td>
<td>stacklight_log_node03_hostname: log03</td>
<td>The hostname of the third StackLight LMA logging node</td>
</tr>
<tr>
<td>StackLight LMA monitor address</td>
<td>stacklight_monitor_address: 10.167.4.70</td>
<td>The virtual IP address of the StackLight LMA monitoring cluster</td>
</tr>
<tr>
<td>StackLight LMA monitor hostname</td>
<td>stacklight_monitor_hostname: mon</td>
<td>The hostname of the StackLight LMA monitoring cluster</td>
</tr>
<tr>
<td>StackLight LMA monitor node01 address</td>
<td>stacklight_monitor_node01_address: 10.167.4.71</td>
<td>The IP address of the first StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA monitor node01 hostname</td>
<td>stacklight_monitor_node01_hostname: mon01</td>
<td>The hostname of the first StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA monitor node02 address</td>
<td>stacklight_monitor_node02_address: 10.167.4.72</td>
<td>The IP address of the second StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA monitor node02 hostname</td>
<td>stacklight_monitor_node02_hostname: mon02</td>
<td>The hostname of the second StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA monitor node03 address</td>
<td>stacklight_monitor_node03_address: 10.167.4.73</td>
<td>The IP address of the third StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA monitor node03 hostname</td>
<td>stacklight_monitor_node03_hostname: mon03</td>
<td>The hostname of the third StackLight LMA monitoring node</td>
</tr>
<tr>
<td>StackLight LMA telemetry address</td>
<td>stacklight_telemetry_address: 10.167.4.85</td>
<td>The virtual IP address of a StackLight LMA telemetry cluster</td>
</tr>
<tr>
<td>StackLight LMA telemetry hostname</td>
<td>stacklight_telemetry_hostname: mtr</td>
<td>The hostname of a StackLight LMA telemetry cluster</td>
</tr>
<tr>
<td>StackLight LMA telemetry node01 address</td>
<td>stacklight_telemetry_node01_address: 10.167.4.86</td>
<td>The IP address of the first StackLight LMA telemetry node</td>
</tr>
</tbody>
</table>
### StackLight LMA telemetry node parameters

| StackLight LMA telemetry node01 hostname | stacklight_telemetry_node01_hostname: mtr01 | The hostname of the first StackLight LMA telemetry node |
| StackLight LMA telemetry node02 address | stacklight_telemetry_node02_address: 10.167.4.87 | The IP address of the second StackLight LMA telemetry node |
| StackLight LMA telemetry node02 hostname | stacklight_telemetry_node02_hostname: mtr02 | The hostname of the second StackLight LMA telemetry node |
| StackLight LMA telemetry node03 address | stacklight_telemetry_node03_address: 10.167.4.88 | The IP address of the third StackLight LMA telemetry node |
| StackLight LMA telemetry node03 hostname | stacklight_telemetry_node03_hostname: mtr03 | The hostname of the third StackLight LMA telemetry node |

### Long-term storage type

| Long-term storage type | stacklight_long_term_storage_type: prometheus | The type of the long-term storage |

### OSS webhook login ID

| OSS webhook login ID | oss_webhook_login_id: 13 | The webhook login ID for alerts notifications |
| OSS webhook app ID | oss_webhook_app_id: 24 | The webhook application ID for alerts notifications |

### Ceph product parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Default JSON output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceph RGW address</td>
<td>ceph_rgw_address: 172.16.47.75</td>
<td>The IP address of the Ceph RGW storage cluster</td>
</tr>
<tr>
<td>Ceph RGW hostname</td>
<td>ceph_rgw_hostname: rgw</td>
<td>The hostname of the Ceph RGW storage cluster</td>
</tr>
<tr>
<td>Ceph MON node01 address</td>
<td>ceph_mon_node01_address: 172.16.47.66</td>
<td>The IP address of the first Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph MON node01 hostname</td>
<td>ceph_mon_node01_hostname: cmn01</td>
<td>The hostname of the first Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph MON node02 address</td>
<td>ceph_mon_node02_address: 172.16.47.67</td>
<td>The IP address of the second Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph MON node02 hostname</td>
<td>ceph_mon_node02_hostname: cmn02</td>
<td>The hostname of the second Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph MON node03 address</td>
<td>ceph_mon_node03_address: 172.16.47.68</td>
<td>The IP address of the third Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph MON node03 hostname</td>
<td>ceph_mon_node03_hostname: cmn03</td>
<td>The hostname of the third Ceph MON storage node</td>
</tr>
<tr>
<td>Ceph RGW node01 address</td>
<td>ceph_rgw_node01_address: 172.16.47.76</td>
<td>The IP address of the first Ceph RGW node</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Ceph RGW node01 hostname</td>
<td>ceph_rgw_node01_hostname: rgw01</td>
<td>The hostname of the first Ceph RGW storage node</td>
</tr>
<tr>
<td>Ceph RGW node02 address</td>
<td>ceph_rgw_node02_address: 172.16.47.77</td>
<td>The IP address of the second Ceph RGW storage node</td>
</tr>
<tr>
<td>Ceph RGW node02 hostname</td>
<td>ceph_rgw_node02_hostname: rgw02</td>
<td>The hostname of the second Ceph RGW storage node</td>
</tr>
<tr>
<td>Ceph RGW node03 address</td>
<td>ceph_rgw_node03_address: 172.16.47.78</td>
<td>The IP address of the third Ceph RGW storage node</td>
</tr>
<tr>
<td>Ceph RGW node03 hostname</td>
<td>ceph_rgw_node03_hostname: rgw03</td>
<td>The hostname of the third Ceph RGW storage node</td>
</tr>
<tr>
<td>Ceph OSD count</td>
<td>ceph_osd_count: 10</td>
<td>The number of OSDs</td>
</tr>
<tr>
<td>Ceph OSD rack01 hostname</td>
<td>ceph_osd_rack01_hostname: osd</td>
<td>The OSD rack01 hostname</td>
</tr>
<tr>
<td>Ceph OSD rack01 single subnet</td>
<td>ceph_osd_rack01_single_subnet: 172.16.47</td>
<td>The control plane network prefix for Ceph OSDs</td>
</tr>
<tr>
<td>Ceph OSD rack01 back-end subnet</td>
<td>ceph_osd_rack01_backend_subnet: 172.16.48</td>
<td>The deploy network prefix for Ceph OSDs</td>
</tr>
<tr>
<td>Ceph public network</td>
<td>ceph_public_network: 172.16.47.0/24</td>
<td>The IP address of Ceph public network with the network mask</td>
</tr>
<tr>
<td>Ceph cluster network</td>
<td>ceph_cluster_network: 172.16.48.70/24</td>
<td>The IP address of Ceph cluster network with the network mask</td>
</tr>
<tr>
<td>Ceph OSD block DB size</td>
<td>ceph_osd_block_db_size: 20</td>
<td>The Ceph OSD block DB size in GB</td>
</tr>
<tr>
<td>Ceph OSD data disks</td>
<td>ceph_osd_data_disks: /dev/vdd,/dev/vde</td>
<td>The list of OSD data disks</td>
</tr>
<tr>
<td>Ceph OSD journal or block DB disks</td>
<td>ceph_osd_journal_or_block_db_disks: /dev/vdb,/dev/vdc</td>
<td>The list of journal or block disks</td>
</tr>
</tbody>
</table>

**Publish the deployment model to a project repository**

If you selected the option to receive the generated deployment model to your email address and customized it as required, you need to apply the model to the project repository.

To publish the metadata model, push the changes to the project Git repository:

```bash
git add *
git commit -m "Initial commit"
```
Create a deployment metadata model manually

You can create a deployment metadata model manually by populating the Cookiecutter template with the required information and generating the model.

For simplicity, perform all the procedures described in this section on the same machine and in the same directory where you have configured your Git repository.

Before performing this task, you need to have a networking design prepared for your environment, as well as understand traffic flow in OpenStack. For more information, see MCP Reference Architecture.

For the purpose of example, the following network configuration is used:

Example of network design with OpenContrail

<table>
<thead>
<tr>
<th>Network</th>
<th>IP range</th>
<th>Gateway</th>
<th>VLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management network</td>
<td>172.17.17.192/26</td>
<td>172.17.17.193</td>
<td>130</td>
</tr>
<tr>
<td>Control network</td>
<td>172.17.18.0/26</td>
<td>N/A</td>
<td>131</td>
</tr>
<tr>
<td>Data network</td>
<td>172.17.18.128/26</td>
<td>172.17.18.129</td>
<td>133</td>
</tr>
<tr>
<td>Proxy network</td>
<td>172.17.18.64/26</td>
<td>172.17.18.65</td>
<td>132</td>
</tr>
<tr>
<td>Tenant network</td>
<td>172.17.18.192/26</td>
<td>172.17.18.193</td>
<td>134</td>
</tr>
<tr>
<td>Salt Master</td>
<td>172.17.18.5/26</td>
<td>172.17.17.197/26</td>
<td>N/A</td>
</tr>
</tbody>
</table>

This Cookiecutter template is used as an example throughout this section.

Define the Salt Master node

When you deploy your first MCP cluster, you need to define your Salt Master node.

For the purpose of this example, the following bash profile variables are used:

```bash
export RECLASS_REPO="/Users/crh/MCP-DEV/mcpdoc"
export ENV_NAME="mcpdoc"
export ENV_DOMAIN="mirantis.local"
export SALT_MASTER_NAME="cfg01"
```
Define the Salt Master node:

1. Log in to the computer on which you configured the Git repository.
2. Using the variables from your bash profile, create a $SALT_MASTER_NAME.$ENV_DOMAIN.yml file in the nodes/ directory with the Salt Master node definition:

   ```yaml
   classes:
   - cluster.$ENV_NAME.infra.config
   parameters:
     _param:
     linux_system_codename: xenial
     reclass_data_revision: master
     linux:
     system:
     name: $SALT_MASTER_NAME
     domain: $ENV_DOMAIN
   ```

3. Add the changes to a new commit:

   ```bash
   git add -A
   ```

4. Commit your changes:

   ```bash
   git commit -m "your_message"
   ```

5. Push your changes:

   ```bash
   git push
   ```

Download the Cookiecutter templates

Use the Cookiecutter templates to generate infrastructure models for your future MCP cluster deployments. Cookiecutter is a command-line utility that creates projects from cookiecutters, that are project templates.

The MCP template repository contains a number of infrastructure models for CI/CD, infrastructure nodes, Kubernetes, OpenContrail, StackLight LMA, and OpenStack.
To access the template repository, you need to have the corresponding privileges. Contact Mirantis Support for further details.

To download the Cookiecutter templates:

1. Install the latest cookiecutter:

   ```bash
   pip install cookiecutter
   ```

2. By default, Cookiecutter installs Jinja 2.9.5. To avoid incompatibility issues, you need to downgrade the version to 2.8.0:

   1. Verify the version of Jinja2 installed on your machine:

      ```bash
      pip freeze cookiecutter
      ```

   2. If the installed version of Jinja2 is lower than 2.8.0, reinstall it to fit the requirement Jinja>=2.8.0.

3. Clone the template repository to your working directory:

   ```bash
   git clone https://github.com/Mirantis/mk2x-cookiecutter-reclass-model.git
   ```

4. Create a symbolic link:

   ```bash
   mkdir $RECLASS_REPO/.cookiecutters
   ln -sv $RECLASS_REPO/mk2x-cookiecutter-reclass-model/cluster_product/* $RECLASS_REPO/.cookiecutters/
   ```

Now, you can generate the required metadata model for your MCP cluster deployment.

See also

Generate an OpenStack environment metadata model

Generate an OpenStack environment metadata model

This section describes how to generate the OpenStack environment model using the cluster_product Cookiecutter template. You need to modify the cookiecutter.json files in the following directories under the .cookiecutter directory:

- **cicd** - cluster name, IP address for the CI/CD control nodes.
- **infra** - cluster name, cluster domain name, URL to the Git repository for the cluster, networking information, such as netmasks, gateway, and so on for the infrastructure nodes.
• opencontrail - cluster name, IP addresses and host names for the OpenContrail nodes, as well as router information. An important parameter that you need to set is the interface mask opencontrail_compute_iface_mask.

• openstack - cluster name, IP addresses, host names, and interface names for different OpenStack nodes, as well as bonding type according to your network design. You must also update the cluster name parameter to be identical in all files. For gateway_primary_first_nic, gateway_primary_second_nic, compute_primary_first_nic, compute_primary_second_nic, specify virtual interface addresses.

• stacklight - cluster name, IP addresses and host names for StackLight LMA nodes.

To generate a metadata model for your OpenStack environment:

1. Log in to the compute on which you configured your Cookiecutter templates.
2. Generate the metadata model:

   1. Create symbolic links for all cookiecutter directories:

   ```
   for i in `ls .cookiecutters`; do ln -sf .cookiecutters/$i/cookiecutter.json cookiecutter.$i.json; done
   ```

   2. Configure infrastructure specifications in all cookiecutter.json files. See: Deployment parameters.

   3. Generate or regenerate the environment metadata model:

   ```
   for i in cicd infra openstack opencontrail stacklight; do 
   cookiecutter .cookiecutters/$i --output-dir ./classes/cluster --no-input -f; done
   ```

The command creates directories and files on your machine. Example:
3. Add your changes to a new commit.
4. Commit and push.

See also
- Cookiecutter documentation
- Deployment parameters

## Deployment parameters
This section lists all parameters that can be modified for generated environments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_name</td>
<td>deployment_name</td>
<td>Name of the cluster, used as cluster/&lt;ENV_NAME&gt;/ in a directory structure</td>
</tr>
<tr>
<td>cluster_domain</td>
<td>deploy-name.local</td>
<td>Domain name part of FQDN of cluster in the cluster</td>
</tr>
<tr>
<td>public_host</td>
<td>public-name</td>
<td>Name or IP of public endpoint of the deployment</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>reclass_repository</td>
<td><a href="https://github.com/Mirantis/mk-lab-salt-model.git">https://github.com/Mirantis/mk-lab-salt-model.git</a></td>
<td>URL to reclass metadata repository</td>
</tr>
<tr>
<td>control_network_netmask</td>
<td>255.255.255.0</td>
<td>IP mask of control network</td>
</tr>
<tr>
<td>control_network_gateway</td>
<td>10.167.4.1</td>
<td>IP gateway address of control network</td>
</tr>
<tr>
<td>dns_server01</td>
<td>8.8.8.8</td>
<td>IP address of dns01 server</td>
</tr>
<tr>
<td>dns_server02</td>
<td>8.8.4.4</td>
<td>IP address of dns02 server</td>
</tr>
<tr>
<td>salt_master_ip</td>
<td>10.167.4.90</td>
<td>IP address of Salt Master on control network</td>
</tr>
<tr>
<td>salt_master_management_ip</td>
<td>10.167.5.90</td>
<td>IP address of Salt Master on management network</td>
</tr>
<tr>
<td>salt_master_hostname</td>
<td>cfg01</td>
<td>Hostname of Salt Master</td>
</tr>
<tr>
<td>kvm_vip_ip</td>
<td>10.167.4.240</td>
<td>VIP address of KVM cluster</td>
</tr>
<tr>
<td>kvm01_control_ip</td>
<td>10.167.4.241</td>
<td>IP address of a KVM node01 on control network</td>
</tr>
<tr>
<td>kvm02_control_ip</td>
<td>10.167.4.242</td>
<td>IP address of a KVM node02 on control network</td>
</tr>
<tr>
<td>kvm03_control_ip</td>
<td>10.167.4.243</td>
<td>IP address of a KVM node03 on control network</td>
</tr>
<tr>
<td>kvm01_deploy_ip</td>
<td>10.167.5.241</td>
<td>IP address of KVM node01 on management network</td>
</tr>
<tr>
<td>kvm02_deploy_ip</td>
<td>10.167.5.242</td>
<td>IP address of KVM node02 on management network</td>
</tr>
<tr>
<td>kvm03_deploy_ip</td>
<td>10.167.5.243</td>
<td>IP address of KVM node03 on management network</td>
</tr>
<tr>
<td>kvm01_name</td>
<td>kvm01</td>
<td>Hostname of a KVM node01</td>
</tr>
<tr>
<td>kvm02_name</td>
<td>kvm02</td>
<td>Hostname of a KVM node02</td>
</tr>
<tr>
<td>kvm03_name</td>
<td>kvm03</td>
<td>Hostname of a KVM node03</td>
</tr>
<tr>
<td>openstack_proxy_address</td>
<td>10.167.4.80</td>
<td>VIP address of proxy cluster on control network</td>
</tr>
<tr>
<td>openstack_proxy_node01_address</td>
<td>10.167.4.81</td>
<td>IP address of a proxy node01 on control network</td>
</tr>
<tr>
<td>openstack_proxy_node02_address</td>
<td>10.167.4.82</td>
<td>IP address of a proxy node02 on control network</td>
</tr>
<tr>
<td>openstack_proxy_hostname</td>
<td>prx</td>
<td>Hostname of VIP proxy cluster</td>
</tr>
<tr>
<td>Variable</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>openstack_proxy_node01_hostname</td>
<td>prx01</td>
<td>Hostname of a proxy node01</td>
</tr>
<tr>
<td>openstack_proxy_node02_hostname</td>
<td>prx02</td>
<td>Hostname of a proxy node02</td>
</tr>
<tr>
<td>openstack_control_address</td>
<td>10.167.4.10</td>
<td>VIP address of control cluster on control network</td>
</tr>
<tr>
<td>openstack_control_node01_address</td>
<td>10.167.4.11</td>
<td>IP address of a control node01 on control network</td>
</tr>
<tr>
<td>openstack_control_node02_address</td>
<td>10.167.4.12</td>
<td>IP address of a control node02 on control network</td>
</tr>
<tr>
<td>openstack_control_node03_address</td>
<td>10.167.4.13</td>
<td>IP address of a control node03 on control network</td>
</tr>
<tr>
<td>openstack_control_hostname</td>
<td>ctl</td>
<td>Hostname of VIP control cluster</td>
</tr>
<tr>
<td>openstack_control_node01_hostname</td>
<td>ctl01</td>
<td>Hostname of a control node01</td>
</tr>
<tr>
<td>openstack_control_node02_hostname</td>
<td>ctl02</td>
<td>Hostname of a control node02</td>
</tr>
<tr>
<td>openstack_control_node03_hostname</td>
<td>ctl03</td>
<td>Hostname of a control node03</td>
</tr>
<tr>
<td>openstack_database_address</td>
<td>10.167.4.50</td>
<td>VIP address of database cluster on control network</td>
</tr>
<tr>
<td>openstack_database_node01_address</td>
<td>10.167.4.51</td>
<td>IP address of a database node01 on control network</td>
</tr>
<tr>
<td>openstack_database_node02_address</td>
<td>10.167.4.52</td>
<td>IP address of a database node02 on control network</td>
</tr>
<tr>
<td>openstack_database_node03_address</td>
<td>10.167.4.53</td>
<td>IP address of a database node03 on control network</td>
</tr>
<tr>
<td>openstack_database_hostname</td>
<td>dbs</td>
<td>Hostname of VIP database cluster</td>
</tr>
<tr>
<td>openstack_database_node01_hostname</td>
<td>dbs01</td>
<td>Hostname of a database node01</td>
</tr>
<tr>
<td>openstack_database_node02_hostname</td>
<td>dbs02</td>
<td>Hostname of a database node02</td>
</tr>
<tr>
<td>openstack_database_node03_hostname</td>
<td>dbs03</td>
<td>Hostname of a database node03</td>
</tr>
<tr>
<td>openstack_message_queue_address</td>
<td>10.167.4.40</td>
<td>VIP address of message queue cluster on control network</td>
</tr>
<tr>
<td>openstack_message_queue_node01_address</td>
<td>10.167.4.41</td>
<td>IP address of a message queue node01 on control network</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>openstack_message_queue_node02_address</td>
<td>10.167.4.42</td>
<td>IP address of a message queue node02 on control network</td>
</tr>
<tr>
<td>openstack_message_queue_node03_address</td>
<td>10.167.4.43</td>
<td>IP address of a message queue node03 on control network</td>
</tr>
<tr>
<td>openstack_message_queue_hostname</td>
<td>msg</td>
<td>Hostname of VIP message queue cluster</td>
</tr>
<tr>
<td>openstack_message_queue_node01_hostname</td>
<td>msg01</td>
<td>Hostname of a message queue node01</td>
</tr>
<tr>
<td>openstack_message_queue_node02_hostname</td>
<td>msg02</td>
<td>Hostname of a message queue node02</td>
</tr>
<tr>
<td>openstack_message_queue_node03_hostname</td>
<td>msg03</td>
<td>Hostname of a message queue node03</td>
</tr>
<tr>
<td>openstack_gateway_node01_address</td>
<td>10.167.4.224</td>
<td>IP address of gateway node01</td>
</tr>
<tr>
<td>openstack_gateway_node02_address</td>
<td>10.167.4.225</td>
<td>IP address of gateway node02</td>
</tr>
<tr>
<td>openstack_gateway_node01_tenant_address</td>
<td>192.168.50.6</td>
<td>IP tenant address of gateway node01</td>
</tr>
<tr>
<td>openstack_gateway_node02_tenant_address</td>
<td>192.168.50.7</td>
<td>IP tenant address of gateway node02</td>
</tr>
<tr>
<td>openstack_gateway_node01_hostname</td>
<td>gtw01</td>
<td>Hostname of gateway node01</td>
</tr>
<tr>
<td>openstack_gateway_node02_hostname</td>
<td>gtw02</td>
<td>Hostname of gateway node02</td>
</tr>
<tr>
<td>stacklight_log_address</td>
<td>10.167.4.60</td>
<td>VIP address of StackLight LMA logging cluster</td>
</tr>
<tr>
<td>stacklight_log_node01_address</td>
<td>10.167.4.61</td>
<td>IP address of StackLight LMA logging node01</td>
</tr>
<tr>
<td>stacklight_log_node02_address</td>
<td>10.167.4.62</td>
<td>IP address of StackLight LMA logging node02</td>
</tr>
<tr>
<td>Variable</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>stacklight_log_node03_address</td>
<td>10.167.4.63</td>
<td>IP address of StackLight LMA logging node03</td>
</tr>
<tr>
<td>stacklight_log_hostname</td>
<td>log</td>
<td>Hostname of StackLight LMA logging cluster</td>
</tr>
<tr>
<td>stacklight_log_node01_hostname</td>
<td>log01</td>
<td>Hostname of StackLight LMA logging node01</td>
</tr>
<tr>
<td>stacklight_log_node02_hostname</td>
<td>log02</td>
<td>Hostname of StackLight LMA logging node02</td>
</tr>
<tr>
<td>stacklight_log_node03_hostname</td>
<td>log03</td>
<td>Hostname of StackLight LMA logging node03</td>
</tr>
<tr>
<td>stacklight_monitor_address</td>
<td>10.167.4.70</td>
<td>VIP address of StackLight LMA monitoring cluster</td>
</tr>
<tr>
<td>stacklight_monitor_node01_address</td>
<td>10.167.4.71</td>
<td>IP address of StackLight LMA monitoring node01</td>
</tr>
<tr>
<td>stacklight_monitor_node02_address</td>
<td>10.167.4.72</td>
<td>IP address of StackLight LMA monitoring node02</td>
</tr>
<tr>
<td>stacklight_monitor_node03_address</td>
<td>10.167.4.73</td>
<td>IP address of StackLight LMA monitoring node03</td>
</tr>
<tr>
<td>stacklight_monitor_hostname</td>
<td>mon</td>
<td>Hostname of StackLight LMA monitoring cluster</td>
</tr>
<tr>
<td>stacklight_monitor_node01_hostname</td>
<td>mon01</td>
<td>Hostname of StackLight LMA monitoring node01</td>
</tr>
<tr>
<td>stacklight_monitor_node02_hostname</td>
<td>mon02</td>
<td>Hostname of StackLight LMA monitoring node02</td>
</tr>
<tr>
<td>stacklight_monitor_node03_hostname</td>
<td>mon03</td>
<td>Hostname of StackLight LMA monitoring node03</td>
</tr>
<tr>
<td>stacklight_telemetry_address</td>
<td>10.167.4.85</td>
<td>VIP address of StackLight LMA telemetry cluster</td>
</tr>
<tr>
<td>stacklight_telemetry_node01_address</td>
<td>10.167.4.86</td>
<td>IP address of StackLight LMA telemetry node01</td>
</tr>
<tr>
<td>stacklight_telemetry_node02_address</td>
<td>10.167.4.87</td>
<td>IP address of StackLight LMA telemetry node02</td>
</tr>
<tr>
<td>stacklight_telemetry_node03_address</td>
<td>10.167.4.88</td>
<td>IP address of StackLight LMA telemetry node03</td>
</tr>
<tr>
<td>stacklight_telemetry_hostname</td>
<td>mtr</td>
<td>Hostname of StackLight LMA telemetry cluster</td>
</tr>
<tr>
<td>stacklight_telemetry_node01_hostname</td>
<td>mtr01</td>
<td>Hostname of StackLight LMA telemetry node01</td>
</tr>
<tr>
<td>stacklight_telemetry_node02_hostname</td>
<td>mtr02</td>
<td>Hostname of StackLight LMA telemetry node02</td>
</tr>
<tr>
<td>Stacklight Telemetry Node03 Hostname</td>
<td>mtr03</td>
<td>Hostname of StackLight LMA telemetry node03</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>OpenStack Compute Node01 Single Address</td>
<td>10.167.2.101</td>
<td>IP address of a compute node01 on a dataplane network</td>
</tr>
<tr>
<td>OpenStack Compute Node02 Single Address</td>
<td>10.167.2.102</td>
<td>IP address of a compute node02 on a dataplane network</td>
</tr>
<tr>
<td>OpenStack Compute Node03 Single Address</td>
<td>10.167.2.103</td>
<td>IP address of a compute node03 on a dataplane network</td>
</tr>
<tr>
<td>OpenStack Compute Node01 Control Address</td>
<td>10.167.4.101</td>
<td>IP address of a compute node01 on a control network</td>
</tr>
<tr>
<td>OpenStack Compute Node02 Control Address</td>
<td>10.167.4.102</td>
<td>IP address of a compute node02 on a control network</td>
</tr>
<tr>
<td>OpenStack Compute Node03 Control Address</td>
<td>10.167.4.103</td>
<td>IP address of a compute node03 on a control network</td>
</tr>
<tr>
<td>OpenStack Compute Node01 Tenant Address</td>
<td>10.167.6.101</td>
<td>IP tenant address of a compute node01</td>
</tr>
<tr>
<td>OpenStack Compute Node02 Tenant Address</td>
<td>10.167.6.102</td>
<td>IP tenant address of a compute node02</td>
</tr>
<tr>
<td>OpenStack Compute Node03 Tenant Address</td>
<td>10.167.6.103</td>
<td>IP tenant address of a compute node03</td>
</tr>
<tr>
<td>OpenStack Compute Node01 Hostname</td>
<td>cmp001</td>
<td>Hostname of a compute node01</td>
</tr>
<tr>
<td>OpenStack Compute Node02 Hostname</td>
<td>cmp002</td>
<td>Hostname of a compute node02</td>
</tr>
<tr>
<td>OpenStack Compute Node03 Hostname</td>
<td>cmp003</td>
<td>Hostname of a compute node03</td>
</tr>
<tr>
<td>OpenStack Compute Node04 Hostname</td>
<td>cmp004</td>
<td>Hostname of a compute node04</td>
</tr>
<tr>
<td>OpenStack Compute Node05 Hostname</td>
<td>cmp005</td>
<td>Hostname of a compute node05</td>
</tr>
<tr>
<td>Ceph RGW Address</td>
<td>172.16.47.75</td>
<td>The IP address of the Ceph RGW storage cluster</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ceph_rgw_hostname</td>
<td>rgw</td>
<td>The hostname of the Ceph RGW storage cluster</td>
</tr>
<tr>
<td>ceph_mon_node01_address</td>
<td>172.16.47.66</td>
<td>The IP address of the first Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_mon_node02_address</td>
<td>172.16.47.67</td>
<td>The IP address of the second Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_mon_node03_address</td>
<td>172.16.47.68</td>
<td>The IP address of the third Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_mon_node01_hostname</td>
<td>cmn01</td>
<td>The hostname of the first Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_mon_node02_hostname</td>
<td>cmn02</td>
<td>The hostname of the second Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_mon_node03_hostname</td>
<td>cmn03</td>
<td>The hostname of the third Ceph MON storage node</td>
</tr>
<tr>
<td>ceph_rgw_node01_address</td>
<td>172.16.47.76</td>
<td>The IP address of the first Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_rgw_node02_address</td>
<td>172.16.47.77</td>
<td>The IP address of the second Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_rgw_node03_address</td>
<td>172.16.47.78</td>
<td>The IP address of the third Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_rgw_node01_hostname</td>
<td>rgw01</td>
<td>The hostname of the first Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_rgw_node02_hostname</td>
<td>rgw02</td>
<td>The hostname of the second Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_rgw_node03_hostname</td>
<td>rgw03</td>
<td>The hostname of the third Ceph RGW storage node</td>
</tr>
<tr>
<td>ceph_osd_count</td>
<td>10</td>
<td>The number of OSDs</td>
</tr>
<tr>
<td>ceph_osd_rack01_hostname</td>
<td>osd</td>
<td>The OSD rack01 hostname</td>
</tr>
<tr>
<td>ceph_osd_rack01_single_subnet</td>
<td>172.16.47.47</td>
<td>The control plane network prefix for Ceph OSDs</td>
</tr>
<tr>
<td>ceph_osd_rack01_backend_subnet</td>
<td>172.16.48.70</td>
<td>The deploy network prefix for Ceph OSDs</td>
</tr>
<tr>
<td>ceph_public_network</td>
<td>172.16.47.0/24</td>
<td>The IP address of Ceph public network with the network mask</td>
</tr>
<tr>
<td>ceph_cluster_network</td>
<td>172.16.48.70/24</td>
<td>The IP address of Ceph cluster network with the network mask</td>
</tr>
<tr>
<td>ceph_osd_block_db_size</td>
<td>20</td>
<td>The Ceph OSD block DB size in GB</td>
</tr>
<tr>
<td>ceph_osd_data_disks</td>
<td>/dev/vdd,/dev/vde</td>
<td>The list of OSD data disks</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ceph_osd_journal_or_block_db_disks</td>
<td>/dev/vdb,/dev/vdc</td>
<td>The list of journal or block disks</td>
</tr>
</tbody>
</table>
Deploy MCP DriveTrain

To reduce the deployment time and eliminate possible human errors, Mirantis recommends that you use the automated approach to the MCP DriveTrain deployment.

Caution!

The execution of the CLI commands used in the MCP Deployment Guide requires root privileges. Therefore, unless explicitly stated otherwise, run the commands as a root user or use sudo.

Automated deployment of MCP DriveTrain

The automated deployment of MCP DriveTrain bases on the bootstrap automation of the Salt Master and MAAS nodes. On a Reclass model creation, you receive the configuration drives on the email, which you specified during the model generation.

Depending on the deployment type, you receive the following configuration drives:

- For an online and offline deployment, the configuration drive for the cfg01 VM that is used in cloud-init to set up a virtual machine with the Salt Master node, MAAS node, Jenkins server, and local Git server installed on it.
- For an offline deployment, the configuration drive for the APT VM that is used in cloud-init to set up a virtual machine with all required repositories mirrors.

The high-level workflow of the automated MCP DriveTrain deployment

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manually deploy and configure the Foundation node.</td>
</tr>
<tr>
<td>2</td>
<td>Create the deployment model using the Model Designer web UI.</td>
</tr>
<tr>
<td>3</td>
<td>Obtain the pre-built ISO configuration drive(s) containing a system submodule on you email. If required, customize and regenerate the configuration drives.</td>
</tr>
<tr>
<td>4</td>
<td>Bootstrap the APT node. For an offline deployment only.</td>
</tr>
<tr>
<td>5</td>
<td>Bootstrap the Salt Master node and MAAS node.</td>
</tr>
<tr>
<td>6</td>
<td>Deploy the remaining bare metal servers through MAAS.</td>
</tr>
<tr>
<td>7</td>
<td>Deploy MCP DriveTrain using Jenkins.</td>
</tr>
</tbody>
</table>

Prerequisites for the automated deployment

Before you proceed with the actual deployment, verify that you have performed the following steps:

1. Deploy the Foundation physical node.
2. Configure bridges on the Foundation node:
   - br-mgm for the management network
   - br-ctl for the control network

   **Note**
   If the IPMI network is not reachable from the management or control network, add the br-ipmi bridge for the IPMI network or any other network that is routed to the IPMI network.

3. Depending on your case, proceed with one of the following options:
   - If you perform an initial MCP deployment:
     1. Create a metadata model using the Model Designer UI as described in Create a deployment metadata model using the Model Designer UI.
     
        **Note**
        In case of an offline deployment select the Offline deployment and Local repositories options under the Repositories section on the Infrastructure parameters tab.

     2. If required, customize the obtained configuration drives and regenerate them as described in Generate configuration drives manually.

     - If you are using an already existing model that does not have configuration drives, or you want to generate updated configuration drives, proceed with Generate configuration drives manually.

   4. Depending on your case, proceed with one of the following options:
      - If you perform the offline deployment or online deployment with local mirrors, proceed to Deploy the APT node.
      - If you perform an online deployment, proceed to Deploy the Salt Master node.

   **See also**
   - Create a bridge for the Foundation node
   - Configure network on the Foundation node

**Deploy the APT node**
MCP enables you to deploy the whole MCP cluster without access to the Internet. On creating the metadata model, along with the configuration drive for the cfg01 VM, you will obtain a preconfigured QCOW2 image that will contain packages, Docker images, operating system images, Git repositories, and other software required specifically for the offline deployment.

This section describes how to deploy the apt01 VM using the prebuilt configuration drive.

### Warning
Perform the procedure below only in case of an offline deployment or when using a local mirror from the prebuilt image.

To deploy the APT node:

1. Log in to the Foundation node.

   **Note**
   Root privileges are required for following steps. Execute the commands as a root user or use sudo.

2. In the `/var/lib/libvirt/images/` directory, create an `apt01/` subdirectory where the offline mirror image will be stored:

   **Note**
   You can create and use a different subdirectory in `/var/lib/libvirt/images/`. If that is the case, verify that you specify the correct directory for the VM_*DISK variables described in next steps.

   ```bash
   mkdir -p /var/lib/libvirt/images/apt01/
   ```


4. Save the image on the Foundation node as `/var/lib/libvirt/images/apt01/system.qcow2`.

5. Copy the configuration ISO drive for the APT VM provided with the metadata model for the offline image to, for example, `/var/lib/libvirt/images/apt01/`.
If you are using an already existing model that does not have configuration drives, or you want to generate updated configuration drives, proceed with Generate configuration drives manually.

```bash
cp /path/to/prepared-drive/apt01-config.iso /var/lib/libvirt/images/apt01/apt01-config.iso
```

6. Select from the following options to deploy the APT node:

1. Download the shell script from GitHub:

   ```bash
   export MCP_VERSION="master"
   https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/${MCP_VERSION}/predefine-vm/define-vm.sh
   ```

2. Make the script executable, export the required variables:

   ```bash
   chmod +x define-vm.sh
   export VM_NAME="apt01.[CLUSTER_DOMAIN]"
   export VM_SOURCE_DISK="/var/lib/libvirt/images/apt01/system.qcow2"
   export VM_CONFIG_DISK="/var/lib/libvirt/images/apt01/apt01-config.iso"
   ```

   The CLUSTER_DOMAIN value is the cluster domain name used for the model. See Basic deployment parameters for details.

   Note

   You may add other optional variables that have default values and change them depending on your deployment configuration. These variables include:

   - VM_MGM_BRIDGE_NAME="br-mgm"
   - VM_MEM_KB="8388608"
   - VM_CPUS="4"

   The br-mgm and br-ctl values are the names of the Linux bridges. See Create a bridge for the Foundation node for details. Custom names can be passed to a VM definition using the VM_MGM_BRIDGE_NAME and VM_CTL_BRIDGE_NAME variables accordingly.

3. Run the shell script:

   ```bash
   ./define-vm.sh
   ```
7. Start the apt01 VM:

```bash
virsh start apt01.[CLUSTER_DOMAIN]
```

8. If you plan to deploy OpenContrail from a local mirror, extract the Ubuntu Trusty mirror archive to the apt01 VM:

1. Connect to the apt01 VM's console.

```bash
virsh console apt01.[CLUSTER_DOMAIN]
```

2. Add your SSH public key for the root user:

```bash
echo '[YOUR_SSH_PUBKEY]' > .ssh/authorized_keys
```

3. Disconnect from the apt01 VM's console.

4. Download the Ubuntu Trusty mirror’s tar.gz file that corresponds to your MCP version:

```bash
```

5. Copy the Ubuntu Trusty mirror tar.gz archive to the apt01 VM using scp:

```bash
scp /var/lib/libvirt/images/apt01/ubuntu-trusty-mirror-[MCP_VERSION].tar.gz root@:/srv/
```

6. Connect to the apt01 VM's console:

```bash
virsh console apt01.[CLUSTER_DOMAIN]
```

7. Extract and decompress the archive:

```bash
tar -xzf /srv/ubuntu-trusty-mirror-[MCP_VERSION].tar.gz --directory /srv/aptly/public/ubuntu
```

8. Disconnect from the apt01 VM's console.

### Deploy the Salt Master node

This section describes how to set up a virtual machine with the Salt Master node, MAAS node, Jenkins server, and local Git server installed on it. The procedure is applicable to both online and offline MCP deployments.

To deploy the Salt Master node:

1. Log in to the Foundation node.
Note
Root privileges are required for following steps. Execute the commands as a root user or use sudo.

2. In case of the offline deployment, replace the content of the /etc/apt/sources.list file with the following lines:

```bash
deb [arch=amd64] http://<<local_mirror_url>>/ubuntu xenial-security main universe restricted
deb [arch=amd64] http://<<local_mirror_url>>/ubuntu xenial-updates main universe restricted
deb [arch=amd64] http://<<local_mirror_url>>/ubuntu xenial main universe restricted
```

3. Create a directory for the VM system disk:

```bash
mkdir -p /var/lib/libvirt/images/cfg01/
```

4. Download the day01 image for the cfg01 node:

```
wget http://images.mirantis.com/cfg01-day01.qcow2 -O /var/lib/libvirt/images/cfg01/system.qcow2
```

5. Copy the configuration ISO drive for the cfg01 VM provided with the metadata model for the offline image to, for example, /var/lib/libvirt/images/cfg01/cfg01-config.iso

```bash
Note
You can create and use a different subdirectory in /var/lib/libvirt/images/. If that is the case, verify that you specify the correct directory for the VM_*DISK variables described in next steps.
```

```bash
cp /path/to/prepared-drive/cfg01-config.iso /var/lib/libvirt/images/cfg01/cfg01-config.iso
```

6. Create the Salt Master VM domain definition using the example script:

1. Download the shell script from GitHub:
export MCP_VERSION="master"
wget https://github.com/Mirantis/mcp-common-scripts/blob/${MCP_VERSION}/predefine-vm/define-vm.sh

2. Make the script executable and export the required variables:

   chmod 0755 define-vm.sh
   export VM_NAME="cfg01.[CLUSTER_DOMAIN]"
   export VM_SOURCE_DISK="/var/lib/libvirt/images/cfg01/system.qcow2"
   export VM_CONFIG_DISK="/var/lib/libvirt/images/cfg01/cfg01-config.iso"
   export VM_CTL_BRIDGE_NAME="br-ctl"

The CLUSTER_DOMAIN value is the cluster domain name used for the model. See Basic deployment parameters for details.

Note

You may add other optional variables that have default values and change them depending on your deployment configuration. These variables include:

- VM_MGM_BRIDGE_NAME="br-mgm"
- VM_MEM_KB="8388608"
- VM_CPUS="4"

The br-mgm and br-ctl values are the names of the Linux bridges. See Create a bridge for the Foundation node for details. Custom names can be passed to a VM definition using the VM_MGM_BRIDGE_NAME and VM_CTL_BRIDGE_NAME variables accordingly.

3. Run the shell script:

   ./define-vm.sh

7. Start the Salt Master node VM and log in into it:

   virsh start cfg01.[CLUSTER_DOMAIN]
   virsh console cfg01.[CLUSTER_DOMAIN]

8. If you use local repositories, verify that mk-pipelines are present in /home/repo/mk and pipeline-library is present in /home/repo/mcp-ci after cloud-init finishes. If not, fix the connection to local repositories and run the /var/lib/cloud/instance/scripts/part-001 script.

9. Verify that the following states are successfully applied during the execution of cloud-init:
Otherwise, fix the pillar and re-apply the above states.

10 In case of using kvm01 as the Foundation node, perform the following steps on it:

1. Depending on the deployment type, proceed with one of the options below:

   • For an online deployment, add the following Debian repository to /etc/apt/sources.list.d/mcp_salt.list:

     ```
     ```

   • For an offline deployment:

     1. In /etc/apt/sources.list.d/mcp_salt.list, add the following Debian repository:

     ```
     deb [arch=amd64] http://<<local_mirror_url>>/ubuntu-xenial stable salt
     ```

     2. Install the salt-minion package.

     3. Modify /etc/salt/minion.d/minion.conf:

     ```
     id: <kvm01_FQDN>
     master: <Salt_Master_IP_or_FQDN>
     ```

     4. Restart the salt-minion service:

     ```
     service salt-minion restart
     ```

     5. Check the output of salt-key command on the Salt Master node to verify that the minion ID of kvm01 is present.

**Configure MAAS for bare metal provisioning**

Before you proceed with the provisioning of the remaining bare metal nodes, you need to configure MAAS properly.

To configure MAAS for bare metal provisioning:

1. Log in to the MAAS web UI through http://<infra_config_deploy_address>:5240/MAAS with the following credentials:

   • Username: mirantis
   • Password: r00tme

2. Go to the Subnets tab.

3. Select the fabric that is under the deploy network.

4. In the VLANs on this fabric area, click the VLAN under the VLAN column where the deploy network subnet is.
5. In the Take action drop-down menu, select Provide DHCP.
6. Adjust the IP range as required.

   **Note**
   The number of IP addresses should not be less than the number of the planned VCP nodes.

7. Click Provide DHCP to submit.
8. If you use local package mirrors:

   **Note**
   The following steps are required only to specify the local Ubuntu package repositories that are secured by a custom GPG key and used mainly for the offline mirror images prior the MCP version 2017.12.

   1. Go to Settings > Package repositories.
   2. Click Actions > Edit on the Ubuntu archive repository.
   3. Specify the GPG key of the repository in the Key field. The key can be obtained from the aptly_gpg_public_key parameter in the cluster level Reclass model.
   4. Click Save.

**Provision physical servers**

Provision the physical servers as described in Provision physical nodes using MAAS.

**Deploy physical servers**

**Caution!**

To avoid the lack of memory for the network driver and ensure its proper operation, specify the minimum reserved kernel memory in your Reclass model on the cluster level for a particular hardware node. For example, use /cluster/<cluster_name>/openstack/compute/init.yml for the OpenStack compute nodes and /cluster/<cluster_name>/infra/kvm.yml for the KVM nodes.

```yaml
linux:
  system:
    kernel:
```
To deploy physical servers:

1. Log in to the Salt Master node.

2. Verify that the cfg01 key has been added to Salt and your host FQDN is shown properly in the Accepted Keys field in the output of the following command:

   ```
salt-key
   ``

3. Verify that all pillars and Salt data are refreshed:

   ```
salt "*" saltutil.refresh_pillar
salt "*" saltutil.sync_all
   ``

4. Verify that the Reclass model is configured correctly. The following command output should show top states for all nodes:

   ```
reclass-salt --top
   ``

5. To verify that the rebooting of the nodes, which will be performed further, is successful, create the trigger file:

   ```
salt -C 'I@salt:control or I@nova:compute or I@neutron:gateway' \
cmd.run "touch /run/is_rebooted"
   ``

6. To prepare physical nodes for VCP deployment, apply the basic Salt states for setting up network interfaces and SSH access. Nodes will be rebooted.

   **Warning**
   If you use kvm01 as a Foundation node, the execution of the commands below will also reboot the Salt Master node.
Caution!

All hardware nodes must be rebooted after executing the commands below. If the nodes do not reboot for a long time, execute the below commands again or reboot the nodes manually.

Verify that you have a possibility to log in to nodes through IPMI in case of emergency.

1. For KVM nodes:

```bash
salt --async -C 'I@salt:control' cmd.run 'salt-call state.sls \ linux.system.user,openssh,linux.network;reboot'
```

2. For compute nodes:

```bash
salt --async -C 'I@nova:compute' cmd.run 'salt-call state.sls \ linux.system.user,openssh,linux.network;reboot'
```

3. For gateway nodes, execute the following command only for the deployments with OVS setup with physical gateway nodes:

```bash
salt --async -C 'I@neutron:gateway' cmd.run 'salt-call state.sls \ linux.system.user,openssh,linux.network;reboot'
```

The targeted KVM, compute, and gateway nodes will stop responding after a couple of minutes. Wait until all of the nodes reboot.

7. Verify that the targeted nodes are up and running:

```bash
salt -C 'I@salt:control or I@nova:compute or I@neutron:gateway' \ test.ping
```

8. Check the previously created trigger file to verify that the targeted nodes are actually rebooted:

```bash
salt -C 'I@salt:control or I@nova:compute or I@neutron:gateway' \ cmd.run 'if [ -f "/run/is_rebooted" ];then echo "Has not been rebooted!";else echo "Rebooted";fi'
```

All nodes should be in the Rebooted state.

9. Verify that the hardware nodes have the required network configuration. For example, verify the output of the `ip a` command:

```bash
salt -C 'I@salt:control or I@nova:compute or I@neutron:gateway' \ cmd.run "ip a"
```
Deploy MCP Drivetrain

To deploy MCP DriveTrain:

1. Deploy a customer-specific CI/CD using Jenkins as part of, for example, an OpenStack cloud environment deployment:
   
   1. Log in to the Jenkins web UI available at salt_master_management_address:8081 with the following credentials:
      
      • Username: admin
      • Password: r00tme
   
   2. Use the Deploy - OpenStack pipeline to deploy cicd cluster nodes as described in Deploy an OpenStack environment. Start with Step 7 in case of the online deployment and with Step 8 in case of the offline deployment.

2. Once the cloud environment is deployed, verify that the cicd cluster is up and running.
3. Disable the Jenkins service on the Salt Master node and start using Jenkins on cicd nodes.

See also

• Enable a watchdog

Manual deployment of MCP DriveTrain

Install a base infrastructure

Base infrastructure of your MCP cluster includes the Salt Master node and the virtualized control plane (VCP). This section describes how to install the Salt Master node, configure the MAAS service, and deploy physical nodes that will host the VCP of your deployment.

Get the virtual machines images

You must prepare virtual machine images before starting the installation to later use them in virtual machines provisioning.

To fetch source images:

1. Log in to the Foundation node console.
2. Download the images:
   
   • `ubuntu-16-04-x64-mcp{MCP_VERSION}.qcow2` for the Virtual Control Plane, DriveTrain, and StackLight nodes (virtual machines).
   
   • `ubuntu-14-04-x64-mcp{MCP_VERSION}.qcow2` for the OpenContrail controller and analytics nodes (only if you plan to install OpenContrail).

For example:
export MCP_VERSION="2018.1"
wget http://images.mirantis.com/ubuntu-16-04-x64-mcp${MCP_VERSION}.qcow2
wget http://images.mirantis.com/ubuntu-14-04-x64-mcp${MCP_VERSION}.qcow2

3. If required, rename the source disk:

mv <src>.qcow2 <src>.src.qcow2

For example:

mv ubuntu-14-*.qcow2 ubuntu-14-04-x64.src.qcow2
mv ubuntu-16-*.qcow2 ubuntu-16-04-x64.src.qcow2

4. Proceed to Prepare images for the Foundation node.

See also
The source code for images building packer-templates repository

Prepare images for the Foundation node
Before you proceed with the Salt Master and MaaS nodes installation, prepare the virtual machine image for the Foundation node.

To prepare images for the Foundation node:

1. Create new sparsely allocated images for MAAS and Salt Master nodes:

qemu-img create -f qcow2 -o preallocation=off mas01.qcow2 120G
qemu-img create -f qcow2 -o preallocation=off cfg01.qcow2 120G

2. Install libguestfs-tools to enable resizing of the image:

sudo apt-get install libguestfs-tools

3. Use the virt-resize command to expand the images and file systems:

sudo virt-resize --expand /dev/vda1 ubuntu-16-04-x64.src.qcow2 mas01.qcow2
sudo virt-resize --expand /dev/vda1 ubuntu-16-04-x64.src.qcow2 cfg01.qcow2

4. Move the images to /var/lib/libvirt/images:

sudo mv cfg01.qcow2 mas01.qcow2 ubuntu-14-04-x64.src.qcow2 ubuntu-16-04-x64.src.qcow2 /var/lib/libvirt/images

5. Proceed to Install the Salt Master node.
Install the Salt Master node

The Salt Master node acts as a central control point for the clients which are called the Salt minion nodes. The minions, in their turn, connect back to the Salt Master node. Before you start the installation process, bootstrap and configure the Salt Master node.

Create a bridge for the Foundation node

You must create a bridge that is connected to the physical interface on the Management network and named br-mgm. This bridge will provide PXE services for all nodes in the MCP cluster. Ensure that the bridge is added to /etc/network/interfaces.

To create a bridge on the Foundation node:

1. Log in to the Foundation node through IPMI.
2. Create a PXE bridge to provision network on the foundation node:
   
   ```sh
   brctl addbr br-mgm
   ```

3. Add a bridge definition for br-mgm in /etc/network/interfaces on the Management network.
   
   Example:

   ```sh
   auto br-mgm
   iface br-mgm inet static
   address 172.17.17.200
   netmask 255.255.255.192
   bridge_ports bond0
   ```

4. Restart networking from the IPMI console.

See also

Prepare images for the Foundation node

Install the Salt Master node

You install the Salt Master node using libvirt CLI.

Install the Salt Master node:

1. Log in to the Foundation node.
2. Create an SSH key pair using the ssh-keygen command.
3. Prepare for a configuration drive image creation:

   ```sh
   export MCP_VERSION=master
   sudo apt-get install -y mkisofs curl cpu-checker qemu-kvm uuid-runtime
   wget -O create-config-drive \\
   https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/blob/$\{MCP_VERSION\}/config-drive/create_config_drive.sh
   chmod +x create-config-drive
   ```
4. Create a user data script:

```
vim vm-config.sh
```

5. Add the following content to the script:

```bash
#!/bin/bash

cat << EOF > /etc/network/interfaces
auto lo
iface lo inet loopback
auto ens3
iface ens3 inet static
  address <salt_master_management_address>
  netmask <deploy_network_subnet>
  gateway <deploy_network_gateway>
  dns-nameservers <dns_server01> <dns_server02>
EOF

erm /etc/network/interfaces.d/*
ifdown ens3 || true
ifup ens3 || true
```

**Note**
For details about the variables used in the code snippet above, refer to Salt Master for 
[salt_master_management_address](#) and to Networking deployment parameters for all 
networking parameters.

6. Create the configuration drive for the Salt Master node and save it as 
`/var/lib/libvirt/images/cfg01-config.iso`:

```
sudo ./create-config-drive.sh -k .ssh/id_rsa.pub -u vm-config.sh -h cfg01 /var/lib/libvirt/images/cfg01-config.iso
```

**Note**
If you are using an already existing model that does not have configuration drives, or 
you want to generate updated configuration drives, proceed with Generate 
configuration drives manually.

7. Boot the Salt Master virtual machine:

1. Download the shell script from GitHub:
2. Make the script executable and export the required variables:

```bash
chmod 0755 define-vm.sh
export VM_NAME="cfg01.[CLUSTER_DOMAIN]"
export VM_SOURCE_DISK="/var/lib/libvirt/images/cfg01/system.qcow2"
export VM_CONFIG_DISK="/var/lib/libvirt/images/cfg01/cfg01-config.iso"
```

The CLUSTER_DOMAIN value is the cluster domain name used for the model. See Basic deployment parameters for details.

Note

You may add other optional variables that have default values and change them depending on your deployment configuration. These variables include:

- VM_MGM_BRIDGE_NAME="br-mgm"
- VM_MEM_KB="8388608"
- VM_CPUS="4"

The br-mgm and br-ctl values are the names of the Linux bridges. See Create a bridge for the Foundation node for details. Custom names can be passed to a VM definition using the VM_MGM_BRIDGE_NAME and VM_CTL_BRIDGE_NAME variables accordingly.

3. Create and run the virtual machine:

```bash
./define-vm.sh
```

Bootstrap the Salt Master node

To bootstrap the Salt Master node:

1. Log in to the Salt Master node console:

    ```bash
    virsh console cfg01
    ```

2. Install the required packages:

    ```bash
    sudo apt-get update
    sudo apt-get install git curl
    ```
3. Download the deploy scripts to the /srv/salt/scripts directory:

```
git clone https://github.com/salt-formulas/salt-formulas-scripts /srv/salt/scripts
```

4. Clone the Git repository with the Reclass model to the /srv/salt/reclass directory:

```
git clone <model-repository> /srv/salt/reclass
```

5. Fetch the classes/system submodule:

```
test ! -e .gitmodules || git submodule update --init --recursive
```

6. If not yet done, initialize submodule in the classes/system directory:

```
git submodule add https://github.com/Mirantis/reclass-system-salt-model \
/srv/salt/reclass/classes/system/ 
git add classes/system && git commit
```

7. Set the following environment variables, where EXTRA_FORMULAS contains the list of formulas required on the Salt Master node. For example:

```
export FORMULAS_SOURCE=pkg
export EXTRA_FORMULAS="linux salt reclass maas memcached openssh ntp sphinx \ 
grafana libvirt rsyslog glusterfs postfix xtrabackup freeipa prometheus telegraf \ 
elasticsearch kibana rundeck devops-portal rsync docker keepalived aptly jenkins \ 
gerrit artifactory influxdb horizon nginx collectd heka mysql backupninja"
export HOSTNAME=cfg01
export DOMAIN=infra.ci.local
```

8. Run the bootstrap.sh script from the /srv/salt/scripts directory:

```
cd /srv/salt/scripts
./bootstrap.sh
```

9. Verify that the cfg01 key has been added to Salt and your host FQDN is shown properly in the Accepted Keys field:

```
salt-key
```

Example of output:

```
Accepted Keys: cfg01.infra.ci.local
Denied Keys:
Unaccepted Keys:
Rejected Keys:
```
10 Verify Reclass by running the verify.sh script from the /srv/salt/reclass directory:

```bash
cd /srv/salt/reclass
HOSTNAME=cfg01 DOMAIN=infra.ci.local ./verify.sh
```

Configure network on the Foundation node

The Salt Master node must have a control network interface that relies on a physical network interface configured on the foundation KVM node. Therefore, you must reconfigure network configuration to create the required bonds, bridges, and so on.

To configure network on the foundation node:

1. Register the foundation node on the Salt Master node as described in add-salt-minions.
2. Before applying the linux.network state, perform a dry run to verify that the network connection between the foundation node and the Salt Master node will not be lost:

```bash
salt-call state.sls linux.system,linux.network test=true
```
3. Run the network configuration state:

```bash
salt-call state.sls linux.system,linux.network
```
4. Bring the bonds up by rebooting the foundation node.
5. Verify that the foundation node bridges are up by checking the output of the ip a show command:

```bash
ip a show br-ctl
```

Example of system response:

```
8: br-ctl: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
link/ether 00:1b:21:93:c7:c8 brd ff:ff:ff:ff:ff:ff
inet 172.17.45.241/24 brd 172.17.45.255 scope global br-ctl
  valid_lft forever preferred_lft forever
inet6 fe80::21b:21ff:fe93:c7c8/64 scope link
  valid_lft forever preferred_lft forever
```

Configure the Salt Master node

The general workflow of the Salt Master node configuration is as follows:

1. Attach review management and PXE networking
2. Attach control plane interfaces
3. Apply base states:
   1. Run the salt.master state.
   2. Run the salt.client state.
Use the Virtual Machine Manager CLI
On the KVM node that hosts the Salt Master VM, run:

```bash
virsh attach-interface --domain cfg01.ci.local --type bridge --source br-ctl --model virtio --config --live
```

Use the Virtual Machine Manager UI

1. Customize the cfg01 Salt Master VM:
   1. After clicking Finish, the Hardware window opens.
   2. Add Network for control services:
      1. Click Add Hardware.
      2. Select network:
         - Network source: Specify shared device name
         - Bridge name: br-ctl
         - MAC address: leave default entry
         - Device: virtio
   2. Verify other settings. For example, you may want to modify the default Display from Spice to VNC for your environment.
   3. Remove undesired and irrelevant items. For example, Sound controller, USB redirectors, and so on.
   4. Click Begin Installation.
   5. Configure the operating system:
      1. Log in to the cfg01 console using the default image user credentials:
         - Username: ubuntu
         - Password: ubuntu
      2. Configure the hostname:
         1. Change the hostname to cfg01:
            ```
            hostname cfg01
            ```
         2. Add the new hostname to /etc/hostname and /etc/hosts.
         3. Update /etc/hostname to contain short hostname
         4. In /etc/hosts, include the short and FQDN of the host on the localhost line.
         5. In /etc/interfaces, define the networking interfaces. For example:
            ```
            source /etc/network/interfaces.d/*
            ```

©2020, Mirantis Inc.
3. To use this node to access the Git repository, obtain the `cfg01` node’s SSH public key and add it to the Git repository server.

**Apply the base states**

```bash
salt-call state.apply salt,reclass,ntp --state-output=changes -lerror
```

Later, once you provision the rest of the infrastructure, apply all states:

```bash
salt-call state.apply --state-output=changes -lerror
```

**Add minions manually**

To manually install a Salt minion:

- `/etc/salt/minion` (id: fqdn; master: ip (on mgmt/pxe network))
- Using MaaS / salt.control
- `salt-key`

**Verify the Salt infrastructure**

Before you proceed with the deployment, validate the Reclass model and node pillars.

To perform verification on the Salt Master node:

1. To verify Salt Master pillar:

   ```bash
   source /srv/salt/scripts/bootstrap.sh
   verify_salt_master
   ```

2. To verify minions pillars, source the `salt-master-init.sh` script and execute the `verify_salt_minion` function:
To perform verification of the model on Salt minion nodes:
Verify that the Salt minion nodes are responding and have the same version as the Salt Master node, that is currently 2016.3.x:

```
salt-call --version
salt '*' test.version
```

Set up a bare-metal provisioner

Bare-metal provisioning (MAAS) is used to discover and install minimum viable product (MVP) components, such as bootstrap, storage, networking, and others, as well as to load Salt minions.

Create the MAAS virtual server

The Metal-As-A-Service (MAAS) server is leveraged in MCP installations to handle physical node lifecycle management. This section guides you through the creation and configuration steps for the Maas server.

The MAAS node is responsible for bare-metal provisioning allowing for rapid and reliable installation of the operating system, basic hardware configurations, and recognition of physical servers by network and system management software. You can deploy the Maas node on the same physical server you run other components of the Virtualized Control Plane on.

A MAAS node can be created either manually or automatically through the cloud controller called Salt Virt in the same way as other control nodes.

To create the MAAS virtual server:

1. Log in to the Salt Master cfg01 node.
2. To create only the mas01 node that will provide DHCP for all other VMs, temporarily comment out all lines related to salt.control.cluster except system.salt.control.cluster.infra_maas_single in /srv/salt/reclass/classes/cluster/cz-bud-mirantis-net/infra/kvm.yml. For example:

```
#- system.salt.control.cluster.opencontrail_analytics_cluster
#- system.salt.control.cluster.opencontrail_control_cluster
#- system.salt.control.cluster.openstack_control_cluster
#- system.salt.control.cluster.openstack_proxy_cluster
#- system.salt.control.cluster.openstack_database_cluster
#- system.salt.control.cluster.openstack_message_queue_cluster
#- system.salt.control.cluster.openstack_telemetry_cluster
#- system.salt.control.cluster.stacklight_server_cluster
#- system.salt.control.cluster.stacklight_log_cluster
#- system.salt.control.cluster.stacklight_telemetry_cluster
- system.salt.control.cluster.infra_maas_single
```
3. Apply the salt.control state on the Foundation node to start the MAAS virtual server.

```
salt "kvm01*" state.sls salt.control
```

5. Log in to the mas01 node console.

   ```bash
   virsh console $(virsh list --name | grep ^mas01)
   ```

6. Configure the temporary mas01 networking. It will be statically set by Salt later.

   ```
   ip a a <ip address>/24 dev ens2
   ip r a default via <gateway>
   ```

7. Restart the Salt minion node:

   ```
   service salt-minion restart
   ```

8. Add the MAAS ppa:

   ```
   sudo add-apt-repository ppa:maas/stable
   ```

9. Proceed to Configure the MAAS service

Configure the MAAS service

After you create the MAAS virtual server as described in Create the MAAS virtual server, you can configure the virtual machine settings. Besides configuring authentication, you need to set DHCP on the networks on which compute nodes will reside. Compute nodes will PXE boot from the specified networks, and the MaaS node will be told how to provision these PXE booted nodes.

To configure the MAAS service:

1. Run Salt states for MAAS:

   ```
   salt-call state.apply linux.system,linux,salt,openssh,ntp
   salt-call state.apply linux.network.interface
   salt-call state.apply maas.cluster,maas.region
   ```

2. Ignore the following error. It is caused by the initial setup.

   ```
   ID: maas_config
   Function: module.run
   Name: maas.process_maas_config
   Result: False
   Comment: Module function maas.process_maas_config threw an exception. Exception: {'updated': [], 'errors':
   {'commissioning_distro_series': "'commissioning_distro_series': 'xenial' is not a valid commissioning_distro_series.
   It should be one of: ['--']."}, 'default_osystem':
   {'default_osystem': 'xenial' is not a valid osystem.
   ```
3. The `salt-formula-maas` package cannot set an autogenerated PostgreSQL password. Therefore, check the system response on the `maas.region` state application for the password and update the `maas_db_password` under `reclass/classes/cluster/<name>/infra/init.yml` accordingly.

Example of system response:

```
----------
  ID: /etc/maas/regiond.conf
Function: file.managed
  Result: True
  Comment: File /etc/maas/regiond.conf updated
  Started: 12:10:17.516313
  Duration: 36.266 ms
  Changes:
       diff:
       ---
       +++
       @@ -5,6 +5,6 @@
       database_host: localhost
       database_name: maasdb
       -database_pass: LdfWijusoUuM
       +database_pass: OMcpBlb07tm2
       database_user: maas
       maas_url: http://172.17.44.91:5240/MAAS
----------
```

4. Before logging to the MAAS web UI, obtain your login credentials by typing the following commands on the MAAS node:

```
salt-call pillar.get maas:region:admin:username
salt-call pillar.get maas:region:admin:password
```

5. Log in to the the MAAS web UI using the following URL: `http://<ip address>/MAAS/#/intro.`
6. Go to Subnets to verify your subnet configuration. By default, MAAS sets the subnet on a network interface under the fabric-0 network. If your deploy network has other name, delete the subnet on fabric-0.

7. Rerun the maas.region state to finalize the configuration:

```
salt-call state.sls maas.region
```

8. On the Networks tab, select deploy_network that is used for PXE/DHCP network in MAAS. For example, fabric-0.

9. In the Take action drop-down menu, select Provide DHCP.

10. Enter the required Dynamic range and Gateway IP.

11. Click Provide DHCP to save the entry. This returns you to the deploy_network network page.

12. Add the SSH key to the MAAS web UI:

   1. In the upper right corner, click the Mirantis username, select Account.
   2. Click Add SSH key.
   3. Paste the generated key into the Public Key field.
   4. Click Add Key.

13. Reboot the VM.

14. In the MAAS web UI, select Nodes and controller. Verify that the node’s status is green.

Alternatively, you can use MAAS CLI.

1. Set the environment variables to be used further. For example:

```
export PROFILE=admin
export API_KEY_FILE=/root/API_KEY_FILE
export MAAS_URL=http://172.17.17.111:5240/MAAS
```

2. Log in to MAAS and create the configuration files:

```
maas-region apikey --username=$PROFILE > $API_KEY_FILE
maas login $PROFILE $MAAS_URL - < $API_KEY_FILE
```

Example of system output:

```
You are now logged in to the MAAS server at
http://172.17.17.111:5240/MAAS/api/2.0/ with the profile name ‘admin’.

For help with the available commands, try:
```

©2020, Mirantis Inc.
maas admin --help

3. Add your lab domain. For example, mk.slab.local:

```bash
maas admin domain update mk.slab.local
```

4. Update the DNS forwarder, if needed:

```bash
maas $PROFILE maas set-config name=upstream_dns value=8.8.8.8
```

Example of system output:

```
Success.
Machine-readable output follows:
OK
```

5. Select all kernels for 64-bit Trusty from boot source with an id of 1:

```bash
maas $PROFILE boot-source-selections create 1 \
  os="ubuntu" release="trusty" arches="amd64" \
  subarches="*" labels="*"
```

6. To import newly selected images:

```bash
maas $PROFILE boot-resources import
```

7. Mark the intro as completed:

```bash
maas $PROFILE maas set-config name=completed_intro value=true
```

8. Create the mirantis user to work with MAAS.

```bash
root@mas01:~# maas-cli createadmin
Username: mirantis
Password:
Again:
Email: mirantis@axlab.local
Import SSH keys [] (lp:user-id or gh:user-id):

root@mas01:~# maas-region apikey --username=mirantis > /root/API_KEY_FILE_mir
```

9. Create a login script for the mirantis user.
Note
The similar script can be created for the admin user with the appropriate API_KEY_FILE and PROFILE parameters.

```bash
cat > maas-login-mir.sh << EOF
#!/bin/sh

# Change these 3 values as required
export PROFILE=mirantis
export API_KEY_FILE=/root/API_KEY_FILE_mir
export MAAS_URL=http://172.17.17.111:5240/MAAS
export API_SERVER=172.17.17.111

MAAS_URL=http://$API_SERVER/MAAS/api/2.0

maas login $PROFILE $MAAS_URL - < $API_KEY_FILE
EOF
```

10 Log in to MAAS:

```
source maas-login-mir.sh
```

11 Import the SSH key. Replace the $SSH_KEY value with the actual public key. For example:

```
export SSH_KEY="ssh-rsa AAAAB3NzaC1yc2EAA root@maas"
maas $PROFILE sshkeys create "key=$SSH_KEY"
```

12 Verify that the DHCP and control subnets are present in MAAS:

```
maas $PROFILE subnets read
```

13 To determine the fabric ID for the PXE network, use the following command:

```
FABRIC_ID=$(maas $PROFILE subnet read $SUBNET_CIDR \ | grep "fabric_id"|cut -d ' ' -f 10 | cut -d "" -f 2|cut -d ',' -f 1)
```

For example, for the PXE network 192.168.201.0/24:

```
FABRIC_ID=$(maas $PROFILE subnet read 192.168.201.0/24 \ | grep "fabric_id"|cut -d ' ' -f 10 | cut -d "" -f 2|cut -d ',' -f 1)
```

14 To determine the primary rack controller name, use the following command:


15 Reserve dynamic range using following command:

```
maas $PROFILE ipranges create type=dynamic \
    start_ip=$IP_DYNAMIC_RANGE_LOW end_ip=$IP_DYNAMIC_RANGE_HIGH \
    comment='This is a reserved dynamic range'
```

For example:

```
maas $PROFILE ipranges create type=dynamic \
    start_ip=192.168.201.11 end_ip=192.168.201.254 \
    comment='This is a reserved dynamic range'
```

16 Reserve the MAAS VM IP as a static entry in the DHCP range:

```
maas $PROFILE ipranges create type=reserved start_ip=192.168.201.9 end_ip=192.168.201.9 \
    comment='This is a reserved range'
```

17 Enable DHCP using the following command:

```
maas $PROFILE vlan update $FABRIC_ID $VLAN_TAG dhcp_on=True \
    primary_rack=$PRIMARY_RACK_CONTROLLER
```

For example:

```
maas $PROFILE vlan update $FABRIC_ID 0 dhcp_on=True \
    primary_rack=$PRIMARY_RACK_CONTROLLER
```

18 Verify the service status:

```
maas $PROFILE nodes read |grep status -A 1 -B 1
```

Provision physical nodes using MAAS

Physical nodes host the Virtualized Control Plane (VCP) of your Mirantis Cloud Platform deployment.

This section describes how to provision the physical nodes using the MAAS service that you have deployed on the Foundation node as described in Set up a bare-metal provisioner. The servers that you must provision include kvm02 and kvm03 KVM nodes along with the compute cmp01 KVM node.

You can provision physical nodes automatically or manually:
An automated provisioning requires you to define IPMI and MAC addresses in your Reclass model. After you enforce all servers, the Salt Master node commissions and provisions them automatically.

A manual provisioning enables commissioning nodes through the MAAS web UI. Before you proceed with the physical nodes provisioning, you may want to customize the commissioning script, for example, to set custom NIC names. For details, see: Add custom commissioning scripts.

Warning
Before you proceed with the physical nodes provisioning, verify that BIOS settings enable PXE booting from NICs on each physical server.

Automatically commission and provision the physical nodes
This section describes how to define physical nodes in a Reclass model to automatically commission and then provision the nodes through Salt.

Automatically commission the physical nodes
You must define all IPMI credentials in your Reclass model to access physical servers for automated commissioning. Once you define the nodes, Salt enforces them into MAAS and starts commissioning.

To automatically commission physical nodes:
1. Define all physical nodes under classes/cluster/<cluster>/infra/maas.yml using the following structure.

   For example, to define the kvm02 node:

   ```yaml
   maas:
     region:
       machines:
         kvm02:
           interface:
             mac: 00:25:90:eb:92:4a
           power_parameters:
             power_address: kvm02.ipmi.net
             power_password: password
             power_type: ipmi
             power_user: ipmi_user
   ```

   Note
   To get MAC addresses from IPMI, you can use the ipmi tool. Usage example for Supermicro:
2. (Optional) Define the IP address on the first (PXE) interface. By default, it is assigned automatically and can be used as is.
For example, to define the kvm02 node:

```
maas:
  region:
    machines:
      kvm02:
        interface:
          mac: 00:25:90:eb:92:4a
          mode: "static"
          ip: "2.2.3.15"
          subnet: "subnet1"
          gateway: "2.2.3.2"
```

3. (Optional) Define a custom disk layout or partitioning per server in MAAS. For more information and examples on how to define it in the model, see: Add a custom disk layout per node in the MCP model.

4. (Optional) Modify the commissioning process as required. For more information and examples, see: Add custom commissioning scripts.

5. Once you have defined all physical servers in your Reclass model, enforce the nodes:
Caution!

For an offline deployment, remove the deb-src repositories from commissioning before enforcing the nodes, since these repositories are not present on the reduced offline apt image node. To remove these repositories, you can enforce MAAS to rebuild sources.list. For example:

```
export PROFILE="mirantis"
export API_KEY=$(cat /var/lib/maas/.maas_credentials)
maas login $(PROFILE) http://localhost:5240/MAAS/api/2.0/ $(API_KEY)
REPO_ID=$(maas $PROFILE package-repositories read | jq '.[] | select(.name=="main_archive") | .id ')
maas $PROFILE package-repository update $(REPO_ID) disabled_components=multiverse
maas $PROFILE package-repository update $(REPO_ID) "disabled_pockets=backports"
```

The default PROFILE variable is mirantis. You can find your deployment-specific value for this parameter in parameters:maas:region:admin:username of your Reclass model.

For details on building a custom list of repositories, see: MAAS GitHub project.

```
salt-call maas.process_machines
```

All nodes are automatically commissioned.

6. Verify the status of servers either through the MAAS web UI or using the salt call command:

```
salt-call maas.machines_status
```

The successfully commissioned servers appear in the ready status.

7. Enforce the interfaces configuration defined in the model for servers:

```
salt-call state.sls maas.machines.assign_ip
```

8. To protect any static IP assignment defined, for example, in the model, configure a reserved IP range in MAAS on the management subnet.

9. (Optional) Enforce the disk custom configuration defined in the model for servers:

```
salt-call state.sls maas.machines.storage
```

10 Verify that all servers have correct NIC names and configurations.

11 Proceed to Provision the automatically commissioned physical nodes.
Provision the automatically commissioned physical nodes

Once you successfully commission your physical nodes, you can start the provisioning.

To provision the automatically commissioned physical nodes through MAAS:

1. Log in to the MAAS node console.
2. Type the salt-call command:

   ```
salt-call maas.deploy_machines
   ```

3. Check the status of the nodes:

   ```
salt-call maas.machines_status
local:  
    Summary:  
        Deploying: 1
   ```

4. When all servers have been provisioned, perform the verification of the servers’ automatic registration by running the salt-key command on the Salt Master node. All nodes should be registered. For example:

   ```
salt-key
   Accepted Keys:
   cfg01.bud.mirantis.net
   cmp001.bud.mirantis.net
   cmp002.bud.mirantis.net
   kvm02.bud.mirantis.net
   kvm03.bud.mirantis.net
   ```

Manually commission and provision the physical nodes

This section describes how to discover, commission, and provision the physical nodes using the MAAS web UI.

Manually discover and commission the physical nodes

You can discover and commission your physical nodes manually using the MAAS web UI.

To discover and commission physical nodes manually:

1. Power on a physical node.
2. In the MAAS UI, verify that the server has been discovered.
3. On the Nodes tab, rename the discovered host accordingly. Click Save after each renaming.
4. In the Settings tab, configure the Commissioning release and the Default Minimum Kernel Version to Ubuntu 16.04 TLS “Xenial Xerus” and Xenial (hwe-16.04), respectively.

   Note
   The above step ensures that the NIC naming convention uses the predictable schemas, for example, enp130s0f0 rather than eth0.

5. In the Deploy area, configure the Default operating system used for deployment and Default OS release used for deployment to Ubuntu and Ubuntu 16.04 LTS “Xenial Xerus”, respectively.

6. Leave the remaining parameters as defaults.

7. (Optional) Modify the commissioning process as required. For more information and examples, see: Add custom commissioning scripts.

8. Commission the node:
   1. From the Take Action drop-down list, select Commission.
   2. Define a storage schema for each node.
   3. On the Nodes tab, click the required node link from the list.
   4. Scroll down to the Available disks and partitions section.
   5. Select two SSDs using check marks in the left column.
   6. Click the radio button to make one of the disks the boot target.
   7. Click Create RAID to create an MD raid1 volume.
   8. In RAID type, select RAID 1.
   10 Set / as Mount point.
   11 Click Create RAID.

   The Used disks and partitions section should now look as follows:

<table>
<thead>
<tr>
<th>Used disks and partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>raid</td>
</tr>
<tr>
<td>raid</td>
</tr>
<tr>
<td>raid part1</td>
</tr>
<tr>
<td>raid</td>
</tr>
</tbody>
</table>

9. Repeat the above steps for each physical node.

10 Proceed to Manually provision the physical nodes.
Manually provision the physical nodes

Start the manual provisioning of the physical nodes with the control plane kvm02 and kvm03 physical nodes, and then proceed with the compute cmp01 node deployment.

To manually provision the physical nodes through MAAS:

1. Verify that the boot order in the physical nodes’ BIOS is set in the following order:
   1. PXE
   2. The physical disk that was chosen as the boot target in the Maas UI.
2. Log in to the MAAS web UI.
3. Click on a node.
4. Click the Take Action drop-down menu and select Deploy.
5. In the Choose your image area, verify that Ubuntu 16.04 LTS “Xenial Xerus” with the Xenial(hwe-16.04) kernel is selected.
6. Click Go to deploy the node.
7. Repeat the above steps for each node.

Now, your physical nodes are provisioned and you can proceed with configuring and deploying the OpenStack controllers nodes on them. If you plan to install CI/CD infrastructure, proceed with Set up physical servers for a CI/CD deployment.

See also
• Configure PXE booting over UEFI

Deploy VCP

The virtualized control plane (VCP) is hosted by KVM nodes deployed by MaaS. The VCP run OpenStack services, database (MySQL), message queue (RabbitMQ), Contrail, and support services, such as monitoring, log aggregation, and a time-series metric database. VMs can be added to or removed from the VCP allowing for easy scaling of your environment.

After the KVM nodes are deployed, Salt is used to configure Linux networking, appropriate repositories, host name, and so on by running the linux Salt state against these nodes. The libvirt packages configuration, in its turn, is managed by running the libvirt Salt state.

Prepare KVM nodes to run OpenStack controller nodes

To prepare physical nodes to run OpenStack controller nodes:

1. On the Salt Master node, prepare the node operating system by running the Salt linux state:

   ```bash
   salt-call state.sls linux -l info
   ```
Warning
Some formulae may not correctly deploy on the first run of this command. This could be due to a race condition in running the deployment of nodes and services in parallel while some services are dependent on others. Repeat the command execution. If an immediate subsequent run of the command fails again, reboot the affected physical node and re-run the command.

2. Prepare physical nodes operating system to run the controller node:
   1. Verify the salt-common and salt-minion versions
   2. If necessary, Install the correct versions of salt-common and salt-minion.
   3. Proceed to Create and provision the control plane VMs.

Verify the salt-common and salt-minion versions
To verify the version deployed with the state:
   1. Log in to the physical node console.
   2. To verify the salt-common version, run:

```
apt-cache policy salt-common
```

3. To verify the salt-minion version, run:

```
apt-cache policy salt-minion
```

The output for the commands above must show the 2016.3.x version. If you have different versions installed, proceed with Install the correct versions of salt-common and salt-minion.

Install the correct versions of salt-common and salt-minion
This section describes the workaround for salt.virt to properly inject minion.conf.
To manually install the required version of salt-common and salt-minion:
   1. Log in to the physical node console
   2. Change the version to 2016.3.8 in /etc/apt/sources.list.d/salt.list:

```
deb [arch=amd64] http://repo.saltstack.com/apt/ubuntu/14.04/amd64/2016.3 trusty main
```

3. Sync the packages index files:

```
apt-get update
```

4. Verify the versions:
apt-cache policy salt-common
apt-cache policy salt-minion

5. If the wrong versions are installed, remove them:

apt-get remove salt-minion
apt-get remove salt-common

6. Install the required versions of salt-common and salt-minion:

apt-get install salt-common=2016.3.8
apt-get install salt-minion=2016.3.8

7. Restart the salt-minion service to ensure connectivity with the Salt Master node:

service salt-minion stop && service salt-minion start

8. Verify that the required version is installed:

apt-cache policy salt-common
apt-cache policy salt-minion

9. Repeat the procedure on each physical node.

Create and provision the control plane VMs

The control plane VMs are created on each node by running the salt state. This state leverages the salt virt module along with some customizations defined in a Mirantis formula called salt-formula-salt. Similarly to how MaaS manages bare metal, the salt virt module creates VMs based on profiles that are defined in the metadata and mounts the virtual disk to add the appropriate parameters to the minion configuration file.

After the salt state successfully runs against a KVM node where metadata specifies the VMs placement, these VMs will be started and automatically added to the Salt Master node.

To create control plane VMs:

1. Log in to the KVM nodes that do not host the Salt Master node and MaaS node. The correct physical node names used in the installation described in this guide to perform the next step are kvm02 and kvm03.

     Warning
     Otherwise, on running the command in the step below, you will delete the cfg Salt Master and mas MaaS nodes!

2. Verify whether virtual machines are not yet present:
virsh list --name --all | grep -Ev '^(mas|cfg|apt)' | xargs -n 1 virsh destroy
virsh list --name --all | grep -Ev '^(mas|cfg|apt)' | xargs -n 1 virsh undefine

3. Log in to the Salt Master node console.

4. Verify that the Salt Minion nodes are synchronized by running the following command on the Salt Master node:

```
salt '*' saltutil.sync_all
```

5. Perform the initial Salt configuration:

```
salt 'kvm*' state.sls salt.minion
```

6. Set up the network interfaces and the SSH access:

```
salt -C 'I@salt:control' cmd.run 'salt-call state.sls 
        linux.system.user,openssh,linux.network;reboot'
```

Warning

This will also reboot the Salt Master and MAAS nodes because they are running on top of kvm01.

7. Log in back to the Salt Master node console.

8. Run the libvirt state:

```
salt 'kvm*' state.sls libvirt
```

9. Add system.salt.control.cluster.openstack_gateway_single to infra/kvm.yml to enable a gateway VM for your OpenStack environment.

10. Run salt.control to create virtual machines. This command also inserts minion.conf files from KVM hosts:

```
salt 'kvm*' state.sls salt.control
```

11. Verify that all your Salt Minion nodes are registered on the Salt Master node. This may take a few minutes.

```
salt-key
...
mon03.bud.mirantis.net
msg01.bud.mirantis.net
```
Deploy DriveTrain

The automated deployment of the MCP components is performed through CI/CD that is a part of MCP DriveTrain along with SaltStack and Reclass. CI/CD, in its turn, includes Jenkins, Gerrit, and MCP Registry components. This section explains how to deploy a CI/CD infrastructure.

MCP CI/CD components

The core components of the MCP CI/CD infrastructure include:

- Jenkins: CI server
- Gerrit: gate changes
- Aptly: Debian repository management
- Docker registry

The CI server roles depend on the specific functions it performs within the CI/CD infrastructure and include:

**CI controller role**

CI/CD controller is the leader of Docker Swarm

The CI/CD controllers run the following services:

- Keepalived and HAProxy
- GlusterFS client
- Docker Swarm mode:
  - Jenkins master
  - Gerrit
  - Aptly (API and public)
  - Docker registry

**CI worker role**
CI workers are responsible for running a wide range of pipelines such as basic test performance or update appliance. An example of CI workers are Jenkins slaves.

Set up physical servers for a CI/CD deployment

Before you proceed with the CI/CD deployment, you must set up the KVM physical nodes, as described in Deploy VCP, right after deploying the physical nodes with MAAS.

To set up the physical nodes for CI/CD:

1. Enable virtual IP:
   
   ```shell
   salt -C '@salt:control' state.sls keepalived
   ```

2. Deploy the GlusterFS cluster:
   
   ```shell
   salt -C '@glusterfs:server' state.sls glusterfs.server.service
   salt -C '@glusterfs:server and *01*' state.sls glusterfs.server.setup
   ```

3. Once the CI nodes are up and running, proceed to Deploy CI/CD.

Deploy CI/CD

Before you start deploying the CI/CD infrastructure, verify that your physical servers are running Ubuntu 16.04 (Xenial) and have Internet access.

To deploy the CI/CD infrastructure:

1. Perform the setup of the physical servers as described in Set up physical servers for a CI/CD deployment.
2. Log in to the Salt Master node.
3. Perform the initial configuration:
   
   ```shell
   salt 'ci*' cmd.run 'salt-call state.sls salt.minion'
   salt 'ci*' state.sls salt.minion,linux,openssh,ntp
   ```

4. Mount Gluster volumes from the KVM nodes:
   
   ```shell
   salt -C '@glusterfs:client and @docker:host' state.sls glusterfs.client
   ```

5. Configure virtual IP and HAProxy balancing:
   
   ```shell
   salt -C '@haproxy:proxy and @docker:host' state.sls haproxy,keepalived
   ```

6. Install Docker:
   
   ```shell
   salt -C '@docker:host' state.sls docker.host
   ```

7. Initial Docker swarm leader:
8. Update the Salt mine to enable other swarm nodes to connect to leader:

```
salt -C '@docker:swarm' state.sls salt
salt -C '@docker:swarm' mine.flush
salt -C '@docker:swarm' mine.update
```

9. Synchronize modules and states:

```
salt -C '@docker:swarm' saltutil.sync_all
```

10. Complete the Docker swarm deployment:

```
salt -C '@docker:swarm' state.sls docker.swarm
```

11. Verify that all nodes are in the cluster:

```
salt -C '@docker:swarm:role:master' cmd.run 'docker node ls'
```

12. Apply the aptly.publisher state:

```
salt -C '@aptly:publisher' state.sls aptly.publisher
```

13. Start the CI/CD containers, for example, MySQL, Aptly, Jenkins, Gerrit, and others:

```
salt -C '@docker:swarm:role:master' state.sls docker.client
```

14. Configure the Aptly service:

```
salt -C '@aptly:server' state.sls aptly
```

15. Configure the OpenLDAP service for Jenkins and Gerrit:

```
salt -C '@openldap:client' state.sls openldap
```

16. Configure the Gerrit service, create users, projects, and so on:

```
salt -C '@gerrit:client' state.sls gerrit
```

Note

If the command execution fails in the first run, re-run it.
17 Configure the Jenkins service, create users, add pipelines, and so on:

```
salt -C 'I@jenkins:client' state.sls jenkins
```

**Note**

If the command execution fails in the first run, re-run it.

Now, you are able to access all CI/CD services using the VIP address and view the HAProxy stats on port 9600 and Docker visualizer on port 8084.

**Deploy CI/CD using Heat templates**

This section explains how to deploy an environment with CI/CD installed using Heat templates.

**Note**

This section is targeted at Mirantis QA engineers only.

**Note**

For production environments, CI/CD should be deployed on a per-customer basis.

For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

After the deployment, you can proceed with the manual MCP cluster deployment described in Deploy an MCP cluster manually.

**To deploy CI/CD using Heat templates:**

1. Verify you complete the Deploy CI/CD procedure.
2. Log in to the Jenkins web UI.
3. Go to the Deploy tab from the top navigation bar.
4. Expand the Heat category.
5. Select the Build with Parameters option from the drop-down menu next to one of the jobs in this category.
6. Specify the following parameters:
   - `HEAT_STACK_DELETE = false`
• SSH_PUBLIC_KEY = <PUBLIC_SSH_RSA_KEY>

7. If required, change the default values for other parameters.

8. Click Build.

By default, after the successful deployment of the pipeline, the DriveTrain will be available at http://172.16.10.254:8800

See also

• View the deployment details
• Deploy an MCP cluster manually
• Remove CI/CD installed using Heat templates

Remove CI/CD installed using Heat templates

You may need to remove an earlier deployed CI/CD. To delete a Heat stack using Jenkins, run the deploy-heat-cleanup Deploy pipeline.

Note

This section is targeted at Mirantis QA engineers only.

Note

For production environments, CI/CD should be deployed on a per-customer basis.

For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

To execute the CI/CD deploy pipeline:

1. Log in to the Jenkins web UI.
2. Go to the Deploy tab from the top navigation bar.
3. Expand the Heat category.
4. Select the Build with Parameters option from the drop-down menu next to the Deploy heat-clean-up job in this category.
5. Define HEAT_STACK_NAME = <HEAT_STACK_NAME>
Note
To obtain the Heat stack name, search for the heat stack-create entry in the output of the job that deployed this specific environment.

6. Click Build.

See also
Deploy CI/CD using Heat templates
Deploy an MCP cluster using DriveTrain

After you have installed the MCP CI/CD infrastructure as described in Deploy DriveTrain, you can reach the Jenkins web UI through the Jenkins master IP address. This section contains procedures explaining how to deploy OpenStack environments and Kubernetes clusters using CI/CD pipelines.

Note

For production environments, CI/CD should be deployed on a per-customer basis.

For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

Deploy an OpenStack environment

This section explains how to configure and launch the OpenStack environment deployment pipeline. This job is run by Jenkins through the Salt API on the functioning Salt Master node and deployed hardware servers to set up your MCP OpenStack environment.

Run this Jenkins pipeline after you configure the basic infrastructure as described in Install a base infrastructure. Also, verify that you have successfully applied the linux and salt states to all physical and virtual nodes for them not to be disconnected during network and Salt Minion setup.

Note

For production environments, CI/CD should be deployed on a per-customer basis.

For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

To automatically deploy an OpenStack environment:

1. Log in to the Salt Master node.
2. Set up network interfaces and the SSH access on all compute nodes:

   ```bash
   salt -C 'I@ nova:compute' cmd.run 'salt-call state.sls \n   linux.system.user,openssh,linux.network;reboot'
   ```

3. If you run OVS, run the same command on physical gateway nodes as well:

   ```bash
   salt -C 'I@ neutron:gateway' cmd.run 'salt-call state.sls \n   linux.system.user,openssh,linux.network;reboot'
   ```
4. Verify that all nodes are ready for deployment:

```
salt '*' state.sls linux,ntp,openssh,salt.minion
```

Caution!

If any of these states fails, fix the issue provided in the output and re-apply the state before you proceed to the next step. Otherwise, the Jenkins pipeline will fail.

5. In a web browser, open http://<ip address>:8081 to access the Jenkins web UI.

Note

The IP address is defined in the classes/cluster/<cluster_name>/cicd/init.yml file of the Reclass model under the cicd_control_address parameter variable.

6. Log in to the Jenkins web UI as admin.

Note

The password for the admin user is defined in the classes/cluster/<cluster_name>/cicd/control/init.yml file of the Reclass model under the openldap_admin_password parameter variable.

7. In the global view, verify that the git-mirror-downstream-mk-pipelines and git-mirror-downstream-pipeline-library pipelines have successfully mirrored all content.

8. Find the Deploy - OpenStack job in the global view.

9. Select the Build with Parameters option from the drop-down menu of the Deploy - OpenStack job.

10 Specify the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description and values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK_ON_ERROR</td>
<td>If checked, Jenkins will ask either to stop a pipeline or continue execution in case of Salt state fails on any task</td>
</tr>
<tr>
<td><strong>STACK_INSTALL</strong></td>
<td>Specifies the components you need to install. The available values include:</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>• core</td>
<td>• kvm</td>
</tr>
<tr>
<td>• cicd</td>
<td>• openstack</td>
</tr>
<tr>
<td>• ovs or contrail</td>
<td>depending on the network plugin.</td>
</tr>
</tbody>
</table>

**Note**

OpenContrail does not support tenant renaming due to architecture limitations.

• ceph
• stacklight
• oss

**Note**

For the details regarding StackLight LMA (stacklight) with the DevOps Portal (oss) deployment, see Deploy StackLight LMA with the DevOps Portal.

<table>
<thead>
<tr>
<th><strong>SALT_MASTER_CREDENTIALS</strong></th>
<th>Specifies credentials to Salt API stored in Jenkins, included by default. See View credentials details used in Jenkins pipelines for details.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>SALT_MASTER_URL</strong></th>
<th>Specifies the reachable IP address of the Salt Master node and port on which Salt API listens. For example, <a href="http://172.18.170.28:6969">http://172.18.170.28:6969</a>. To find out on which port Salt API listens:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Log in to the Salt Master node.</td>
</tr>
<tr>
<td></td>
<td>2. Search for the port in the /etc/salt/master.d/_api.conf file.</td>
</tr>
<tr>
<td></td>
<td>3. Verify that the Salt Master node is listening on that port:</td>
</tr>
<tr>
<td></td>
<td>netstat -tunelp</td>
</tr>
</tbody>
</table>

| **STACK_TYPE** | Specifies the environment type. Use physical for a bare metal deployment                                                                                                            |
11 Click Build.

See also
- View the deployment details
- Enable a watchdog

**Deploy a multi-site OpenStack environment**

MCP DriveTrain enables you to deploy several OpenStack environments at the same time.

**Note**

For production environments, CI/CD should be deployed on a per-customer basis.

For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

To deploy a multi-site OpenStack environment, repeat the Deploy an OpenStack environment procedure as many times as you need specifying different values for the SALT_MASTER_URL parameter.

See also
- View the deployment details

**Deploy a Kubernetes cluster**

The MCP Containers as a Service architecture enables you to easily deploy a Kubernetes cluster on top of AWS or OpenStack cloud providers with Calico or OpenContrail plugins set for Kubernetes networking.

This section explains how to configure and launch the Kubernetes cluster deployment pipelines using DriveTrain. The available deployment pipelines include:

`deploy-{{stack_template}}-{{stack_type}}`

where:
- stack_template is k8s_ha_calico or k8s_ha_contrail
• stack_type is aws or heat

Note
The OpenContrail for the Kubernetes on AWS deployment is not supported yet.

You may also configure and use either of the above-mentioned pipelines to deploy Kubernetes on bare metal without using a cloud provider for deployment.

If you choose heat as the deployment type, you can enable the OpenStack cloud provider functionality that allows leveraging Cinder volumes and Neutron LBaaS to enhance the functionality of the Kubernetes cluster.

You can also deploy ExternalDNS to set up a DNS management server in order to control DNS records dynamically through Kubernetes resources and make Kubernetes resources discoverable through public DNS servers.

Depending on the required type of installation, proceed with the corresponding section listed below.

Note
For production environments, CI/CD should be deployed on a per-customer basis.
For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

Prerequisites
Before you proceed with an automated deployment of a Kubernetes cluster, follow the steps below:

1. If you have swap enabled on the ctl and cmp nodes, modify your Kubernetes deployment model as described in Add swap configuration to a Kubernetes deployment model.
2. Deploy DriveTrain as described in Deploy MCP DriveTrain.

Now, proceed to deploying Kubernetes depending on your use case.

Deploy a Kubernetes cluster on AWS
This section provides the steps to deploy a Kubernetes cluster on AWS with Calico as a Kubernetes networking plugin.

To automatically provision the Kubernetes cluster on AWS:

1. Verify that you have completed the steps described in Prerequisites.
2. Log in to the Jenkins web UI as Administrator.
Note

The password for the Administrator is defined in the classes/cluster/<CLUSTER_NAME>/cicd/control/init.yml file of the Reclass model under the openldap_admin_password parameter variable.

3. Find the Pipeline Deploy - k8s_calico aws job in the global view.

4. Select the Build with Parameters option from the drop-down menu of the Pipeline Deploy - k8s_calico aws job.

5. Configure the deployment by setting the following parameters as required:

### Deploy - k8s_calico AWS parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASK_ON_ER_ROR</strong></td>
<td>False</td>
<td>If True, Jenkins will stop on any failure and ask either you want to cancel the pipeline or process execution and ignore the error.</td>
</tr>
<tr>
<td><strong>AWS_API_CREDENTIALS</strong></td>
<td>aws-credentials</td>
<td>The AWS credentials of the Jenkins user name and password format. The user name is the AWS Secret Key ID, the password is the AWS Secret Access Key. You need to obtain credentials from Amazon IAM.</td>
</tr>
<tr>
<td><strong>AWS_SSH_KEY</strong></td>
<td>N/A</td>
<td>The ID of a keypair in AWS. This option is required. However, the keypair is used only for initial bootstrap, and SSH keys from the Reclass metadata model will be added as well.</td>
</tr>
<tr>
<td><strong>AWS_S_TAC_R_KEY</strong></td>
<td>us-west-2</td>
<td>The AWS Region. Currently, the us-west-2 is the only supported region. To add more regions to the AWS CloudFormation template, set the AMI of Ubuntu images in this region in the AWSRegion2Ami mappings configuration.</td>
</tr>
<tr>
<td>SAL_T_MASTER_CREDENTIALS</td>
<td>N/A</td>
<td>The credentials for logging in to the Salt API. See View credentials details used in Jenkins pipelines for details.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STACK_CLÉANUP JOB</td>
<td>deploy-stack-cleanup</td>
<td>Optional, the name of the job to clean the stack.</td>
</tr>
<tr>
<td>STACK.Compute_COUNT</td>
<td>2</td>
<td>The number of Kubernetes compute nodes.</td>
</tr>
<tr>
<td>STACK_Delete</td>
<td>True</td>
<td>Deletes the cluster after installation. Uncheck if you need to use the cluster after deployment.</td>
</tr>
<tr>
<td>STACK.Install_All</td>
<td>core,k8s,calico</td>
<td>Components to install</td>
</tr>
<tr>
<td>STACK_Name</td>
<td>N/A</td>
<td>The AWS stack name. Will be generated if missing. Used only when STACK_REUSE is enabled.</td>
</tr>
<tr>
<td>STACK_Reuse</td>
<td>False</td>
<td>If True, Jenkins will not create a new stack trying to reuse the existing one.</td>
</tr>
<tr>
<td>STACK_Template</td>
<td>k8s_calico</td>
<td>The name of the AWS CloudFormation template in the cfg directory.</td>
</tr>
<tr>
<td>Stack_emplate_branch</td>
<td>master</td>
<td>The branch for the cluster templates.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Stack-template_credential</td>
<td>N/A</td>
<td>The credentials for logging in to the repository with the cluster templates.</td>
</tr>
<tr>
<td>Stack-template_url</td>
<td>N/A</td>
<td>The URL to the repository that contains the cluster templates. For example: ssh://jenkins-mk@gerrit.mcp.mirantis.net:29418/mk/heat-templates.</td>
</tr>
<tr>
<td>Stack_test</td>
<td>Empty</td>
<td>The names of the cluster components to test. By default, nothing is tested.</td>
</tr>
<tr>
<td>Stack_type</td>
<td>aws</td>
<td>The type of the cluster.</td>
</tr>
</tbody>
</table>

6. Click Build to launch the pipeline.

7. Click Full stage view to track the deployment process.

The Deploy - k8s_calico aws pipeline workflow

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
</table>

©2020, Mirantis Inc.  Page 99
<table>
<thead>
<tr>
<th></th>
<th>Create infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Create infrastructure</strong></td>
</tr>
<tr>
<td></td>
<td>1. Creates the CloudFormation cluster.</td>
</tr>
<tr>
<td></td>
<td>On launch, the pipeline starts a virtual environment and prepares to set up the infrastructure by starting to create the conformation template in AWS.</td>
</tr>
<tr>
<td></td>
<td>To verify that the conformation template in AWS is being created:</td>
</tr>
<tr>
<td></td>
<td>1. Log in to the AWS web UI or refresh the page if you are already logged in.</td>
</tr>
<tr>
<td></td>
<td>2. Verify that the new cluster status is <code>CREATE_IN_PROGRESS</code> indicating that the pipeline is creating the conformation template and applying this template against conformation.</td>
</tr>
<tr>
<td></td>
<td>2. Bootstraps nodes.</td>
</tr>
<tr>
<td></td>
<td>• On the Salt Master node:</td>
</tr>
<tr>
<td></td>
<td>1. Bootstraps the Salt Master node</td>
</tr>
<tr>
<td></td>
<td>2. Clones the Reclass model</td>
</tr>
<tr>
<td></td>
<td>3. Enforces the basic states</td>
</tr>
<tr>
<td></td>
<td>• On the Salt Minion nodes:</td>
</tr>
<tr>
<td></td>
<td>1. Bootstraps the nodes</td>
</tr>
<tr>
<td></td>
<td>2. Enforces the basic states</td>
</tr>
<tr>
<td></td>
<td>3. Registers the nodes to the Salt Master node</td>
</tr>
<tr>
<td></td>
<td>To verify that the VMs are created successfully:</td>
</tr>
<tr>
<td></td>
<td>1. Log in to the Jenkins web UI.</td>
</tr>
<tr>
<td></td>
<td>2. Go to the Outputs tab and copy the Salt Master node public IP address.</td>
</tr>
<tr>
<td></td>
<td>3. Log in to the Salt Master node running on AWS using the copied Salt Master node public IP address. You can see all registered nodes that include 3 controller nodes and 2 compute nodes.</td>
</tr>
<tr>
<td></td>
<td>During this stage, the pipeline also creates all required resources that include virtual private cloud (VPC), Floating IPs, security groups, gateways, scaling groups, and load balancers.</td>
</tr>
</tbody>
</table>
2. Install core infrastructure
   1. Prepares and validates the Salt Master node and Minion nodes that includes refreshing pillars, synchronization of custom modules.
   2. Applies the linux,openssh,salt.minion,ntp states to all nodes.

3. Install Kubernetes infrastructure
   1. Reads the control plane load-balancer address provided by AWS LB and applies it to the model.
   2. Generates the Kubernetes certificates.
   3. Installs the Kubernetes support packages that include Keepalived, HAProxy, Docker, and etcd.

4. Install Kubernetes control plane
   Installs Calico, sets up etcd, installs the control planes nodes.

5. Scale Kubernetes compute nodes
   1. Scales up the Kubernetes compute nodes by updating the count of nodes in the AWS scaling group as configured in the STACK_COMPUTE_COUNT parameter.
   2. Waits 60 seconds for computes to boot.
   3. Runs the states to install Kubernetes Nodes.

6. Finalize
   Finishes the Kubernetes installation by re-applying all states to ensure that everything is working correctly.

8. When the pipeline has successfully executed, log in to any Kubernetes ctl node and verify that all nodes have been registered successfully:

   kubectl get nodes

See also
- View the deployment details
- Scale up a Kubernetes cluster

Deploy a Kubernetes cluster on OpenStack
This section provides the steps to deploy a Kubernetes cluster on top of OpenStack with Calico or OpenContrail as a Kubernetes networking plugin.

If you want to add ExternalDNS or OpenStack cloud provider features to your Kubernetes cluster, refer to the Enable OpenStack cloud provider in Kubernetes and Deploy ExternalDNS for Kubernetes sections for details before you start the deployment.

To automatically deploy the Kubernetes cluster on OpenStack:
1. Verify that you have completed the steps described in Prerequisites.
2. Log in to the Jenkins web UI as Administrator.

**Note**
The password for the Administrator is defined in the classes/cluster/<CLUSTER_NAME>/cicd/control/init.yml file of the Reclasse model under the openldap_admin_password parameter variable.

3. Depending on your use case, find the k8s_ha_calico heat or k8s_ha_contrail heat job in the global view.
4. Select the Build with Parameters option from the drop-down menu of the selected job.
5. Configure the deployment by setting the following parameters as required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK_ON_ERROR</td>
<td>False</td>
<td>If True, Jenkins will stop on any failure and ask either you want to cancel the pipeline or proceed with the execution ignoring the error.</td>
</tr>
<tr>
<td>OPESTACK_API_CLIENT</td>
<td>Empty</td>
<td>The version of the OpenStack client to install and use for connections. Available values are empty meaning latest, kilo, liberty, and mitaka.</td>
</tr>
<tr>
<td>OPESTACK_API_CREDENTIALS</td>
<td>openstack-devcloud-credential</td>
<td>Jenkins credentials for the OpenStack environment.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>OPENSTACK_API_PROJECT</td>
<td>N/A</td>
<td>The Openstack project to connect to.</td>
</tr>
<tr>
<td>OPENSTACK_API_PROJECT_DOMAIN</td>
<td>default</td>
<td>The domain for the OpenStack project.</td>
</tr>
<tr>
<td>OPENSTACK_API_PROJECT_ID</td>
<td>Empty</td>
<td>The ID of the OpenStack project.</td>
</tr>
<tr>
<td>OPENSTACK_API_URL</td>
<td>N/A</td>
<td>The OpenStack API address (Keystone).</td>
</tr>
<tr>
<td>OPENSTACK_API_USER_DOMAIN</td>
<td>default</td>
<td>The OpenStack user domain.</td>
</tr>
<tr>
<td><strong>OPENSTACK_API_VERSION</strong></td>
<td>3</td>
<td>The version of the OpenStack API to use. Available values are 2 and 3.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SALT_MASTERCREDENTIALS</strong></td>
<td>N/A</td>
<td>The credentials for logging in to the Salt API. See View credentials details used in Jenkins pipelines for details.</td>
</tr>
<tr>
<td><strong>STACK_CLEANUPJOB</strong></td>
<td>deploy-stack-cleanup</td>
<td>Optional, the name of the job to clean the stack.</td>
</tr>
<tr>
<td><strong>STACK_COMPUTECOUNT</strong></td>
<td>2</td>
<td>The number of Kubernetes compute nodes.</td>
</tr>
<tr>
<td><strong>STACK_DELETE</strong></td>
<td>True</td>
<td>Deletes the cluster after installation. Uncheck if you need to use the cluster after deployment.</td>
</tr>
</tbody>
</table>
| **STACK_INSTALL** | • core,k8s,calico for a deployment with Calico  
• core,k8s,contrail for a deployment with OpenContrail | Components to install |
<p>| <strong>STACK_NAME</strong> | N/A | The Heat stack name. Will be generated if missing. Used only when STACK_REUSE is enabled. |</p>
<table>
<thead>
<tr>
<th><strong>STACK_REUSE</strong></th>
<th>False</th>
<th>If True, Jenkins will not create a new stack trying to reuse the existing one.</th>
</tr>
</thead>
</table>
| **STACK_TEMPLATE** | • k8s_calico for the deployments with Calico  
• k8s_ha_contrail for the deployments with OpenContrail | The name of the Heat template in the cfg directory. |
| **STACKTEMPLATE_BRANCH** | master | The branch for the cluster templates. |
| **STACK_TEMPLATE_CREDENTIALS** | N/A | The credentials for logging in to the repository with the cluster templates. |
| **STACK_TEMPLATE_URL** | N/A | The URL to the repository that contains the cluster templates. For example: ssh://jenkins-mk@gerrit.mcp.mirantis.net:29418/mk/heat-templates. |
| **STACK_TEST** | Empty | The names of the cluster components to test. By default, nothing is tested. |
| **STACK_TYPE** | heat | The type of the cluster. |

6. Click Build to launch the pipeline.
7. Click Full stage view to track the deployment process.

- The The Deploy - k8s_ha_calico heat pipeline workflow table contains the stages details for the deployment with Calico as a Kubernetes networking plugin:
# The Deploy - k8s_ha_calico heat pipeline workflow

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create infrastructure</td>
<td>Creates a Heat stack, waits for it to appear, and starts reading output from Heat, for example, the Salt Master node IP.</td>
</tr>
<tr>
<td>2</td>
<td>Install core infrastructure</td>
<td>1. Prepares and validates the Salt Master node and Salt Minion nodes. For example, refreshes pillars and synchronizes custom modules.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Applies the linux, openssh, salt.minion, ntp states to all nodes.</td>
</tr>
<tr>
<td>3</td>
<td>Install Kubernetes infrastructure</td>
<td>1. Reads the control plane load-balancer address and applies it to the model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Generates the Kubernetes certificates.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Installs the Kubernetes support packages that include Keepalived, HAProxy, Docker, and etc.</td>
</tr>
<tr>
<td>4</td>
<td>Install Kubernetes control plane</td>
<td>Installs Calico, sets up etcd, installs the control planes nodes.</td>
</tr>
<tr>
<td>5</td>
<td>Scale Kubernetes compute nodes</td>
<td>1. Scales up the Kubernetes compute nodes by updating the count of nodes in the AWS scaling group as configured in the STACK_COMPUTE_COUNT parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Waits 60 seconds for computes to boot.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Runs the states to install Kubernetes Nodes.</td>
</tr>
</tbody>
</table>

- The The Deploy - k8s_ha_contrail heat pipeline workflow table contains the stages details for the deployment with OpenContrail as the Kubernetes networking plugin:

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create infrastructure</td>
<td>Creates a Heat stack, waits for it to appear, and starts reading output from Heat, for example, the Salt Master node IP.</td>
</tr>
<tr>
<td>2</td>
<td>Install OpenContrail for Kubernetes</td>
<td>1. Installs the OpenContrail infrastructure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Configures OpenContrail to be used by Kubernetes.</td>
</tr>
</tbody>
</table>

8. When the pipeline has successfully executed, log in to any Kubernetes `ctl` node and verify that all nodes have been registered successfully:

```
kubectl get nodes
```
While deploying MCP Kubernetes cluster using the Heat deploy job, you can also enable the OpenStack cloud provider functionality that allows you leveraging Cinder volumes and Neutron LBaaS.

Enable OpenStack cloud provider in Kubernetes

If you deploy Kubernetes on top of OpenStack using Heat templates as described in Deploy a Kubernetes cluster on OpenStack, you can enable the OpenStack cloud provider functionality, which is built into Kubernetes itself. The OpenStack cloud provider allows a deployer to leverage Cinder volumes and Neutron LBaaS to enhance the functionality of the Kubernetes cluster.

Important

The OpenStack cloud provider functionality is only available when you deploy a Kubernetes cluster on top of OpenStack as described in Deploy a Kubernetes cluster on OpenStack.

The two main functions provided by the OpenStack cloud provider are PersistentVolume for pods and LoadBalancer for services.

Considerations when using OpenStack cloud provider

The OpenStack cloud provider for Kubernetes has several requirements in OpenStack, which are outlined in the OpenStack cloud provider Overview section. In addition to component requirements, there are operational requirements:

- Instance names must have a proper DNS label, consisting of letters, numbers, and dashes, ending with an alphanumeric character. Underscores and other symbols are invalid.
- All Kubernetes nodes must be Nova instances in the same project/tenant. Bare metal hosts or OpenStack instances from another tenant cannot be joined to the cluster with the OpenStack cloud provider.
- All Kubernetes nodes must be on the same Neutron subnet.

In addition to operational requirements, the OpenStack cloud provider introduces a significant security concern. As a result, a non-privileged user should be created in the project/tenant where the instances reside specifically for this purpose. The reason behind this is that every single Kubernetes node (both controller and compute nodes) needs to contain the entire credentials in cleartext in the /etc/kubernetes/cloud-config.conf file. These credentials are put into pillar as well, so this is also a security vector to be aware of.

Enable OpenStack cloud provider

In order to enable the OpenStack cloud provider, you must configure it in the Salt model.

To enable the OpenStack cloud provider:

1. Open your Git project repository.
2. In `classes/cluster/<cluster_name>/kubernetes/control.yml`, add the following text to pillar data, replacing the credentials to reflect your OpenStack environment:

```yaml
kubernetes:
  common:
    cloudprovider:
      enabled: True
      provider: openstack
      params:
        auth_url: https://openstack.mydomain:5000/v3
        username: nova
        password: nova
        region: RegionOne
        tenant_name: mytenant
        domain_name: default
        subnet_id: 72407854-aca6-4cf1-b873-e9affb09484b
        lb_version: v2
```

**Note**

The `subnet_id` parameter must be the UUID of the subnet that is attached to the first network of any host in your cluster. Do not use the network ID.

3. Commit and push the changes to the project Git repository.

4. Select from the following options:
   - If you are performing the initial deployment of your MCP cluster, proceed with the further cluster configuration as required.
   - If you are making changes to an existing MCP cluster, re-run the Salt configuration on the Salt Master node to apply changes:
     1. Log in to the Salt Master node.
     2. Update your Salt formulas and the system level of your repository:
        1. Change the directory to `/srv/salt/reclass`.
        2. Run the `git pull origin master` command.
        3. Run the `salt-call state.sls salt.master` command.
        4. Run the `salt-call state.sls reclass` command.
     3. In the Jenkins web UI, re-run the Kubernetes deployment pipeline that was used for your MCP cluster deployment.
     After you enable the OpenStack cloud provider, proceed to Verify OpenStack cloud provider after deployment.

Verify OpenStack cloud provider after deployment
After you enable the OpenStack cloud provider on your Kubernetes cluster as described in Enable OpenStack cloud provider, verify that it has been successfully deployed using the procedure below.

To verify OpenStack cloud provider:

1. Log in to the Kubernetes Master node.
2. Create a file called cinder-storage.yaml with the following content:

   ```yaml
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     name: gold
   provisioner: kubernetes.io/cinder
   parameters:
     type: fast
     availability: nova
   ``

3. Run the following command:

   `kubectl apply -f cinder-storage.yaml`

4. Create a file called claim1.yaml with the following content:

   ```yaml
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: claim1
   spec:
     accessModes:
     - ReadWriteMany
     resources:
       requests:
         storage: 1Gi
   ``

5. Run the following command:

   `kubectl apply -f claim1.yaml`

6. Create a file called cinder-test-rc.yaml with the following content:

   ```yaml
   apiVersion: v1
   kind: ReplicationController
   metadata:
     name: server
   spec:
     replicas: 1
   ```
spec:
  containers:
  - name: server
    image: nginx
    volumeMounts:
    - mountPath: /var/lib/www/html
      name: cinderpvc
    volumes:
    - name: cinderpvc
      persistentVolumeClaim:
        claimName: claim1

7. Run the following command:

    kubectl apply -f cinder-test-rc.yaml

8. Verify that the volume was created:

    cinder list

9. Verify that Neutron LBaaS can create LoadBalancer objects:

    1. Create the nginx-rs.yml file with the following content:

        apiVersion: extensions/v1beta1
        kind: ReplicaSet
        metadata:
          name: nginx
        spec:
          replicas: 4
        template:
          metadata:
            labels:
              app: nginx
          spec:
            containers:
            - name: nginx
              image: nginx:1.10
              resources:
                requests:
                  cpu: 100m
                  memory: 100Mi

        2. Run the following commands:

            kubectl create -f nginx-rs.yml
            kubectl expose rs nginx --port 80 --type=LoadBalancer
10 Verify that the service has an external IP:

```
kubectl get services -owide
```

Example of system response:

<table>
<thead>
<tr>
<th>NAME</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
<th>SELECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubernetes</td>
<td>10.254.0.1</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>40m</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>nginx</td>
<td>10.254.18.214</td>
<td>192.168.10.96,172.17.48.159</td>
<td>80:31710/TCP</td>
<td>1m</td>
<td>app=nginx</td>
</tr>
</tbody>
</table>

11 Verify that the LoadBalancer was created in OpenStack:

```
neutron lbaas-loadbalancer-list
```

In the output, the vip_address should match the first external IP for the service created.

See also

Troubleshoot OpenStack cloud provider

Troubleshoot OpenStack cloud provider

The table in this section includes solutions for issues related to OpenStack cloud provider operations after deployment.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinder volume cannot be mounted</td>
<td>1. Verify that the --cloud-provider and --cloud-config options are enabled on the Kubernetes node.</td>
</tr>
<tr>
<td></td>
<td>2. Check logs for the pod that failed and kubelet logs. There may be permission issues or incorrect zone specified in the StorageClass object.</td>
</tr>
<tr>
<td>Cinder volume does not create</td>
<td>1. Verify that your user has privileges to create Cinder volumes:</td>
</tr>
<tr>
<td></td>
<td>1. Source the openrc file of your environment. For details, see: Create OpenStack client environment scripts.</td>
</tr>
<tr>
<td></td>
<td>2. Run the openstack volume create test --size 1.</td>
</tr>
<tr>
<td></td>
<td>2. Check logs for kube-controller-manager on each Kubernetes Master node.</td>
</tr>
</tbody>
</table>
### The kubelet agent does not register with apiserver

1. Verify that the instance name does not contain invalid characters. An instance name must be a RFC-953 compliant, which states that a DNS name must consist of characters drawn from the alphabet (A-Z), digits (0-9), minus sign (-), and period (.). It is best to destroy and recreate the instance because the configdrive metadata located in /dev/vdb cannot be updated automatically even after renaming an instance.

2. Verify that your cloud credentials are valid. The kubelet agent will not start if the credentials are wrong.

### Heat stack cannot be deleted because of LoadBalancer services

1. Delete all service resources before deleting the Heat stack using the kubectl delete svc --all command.

2. If the stack was already deleted and is now in the DELETE_FAILED state, purge all LBaaS objects visible to your OpenStack user with the following commands:

   ```bash
   for pool in `neutron lbaas-pool-list -c id -f value`; do
     while read member; do
       neutron lbaas-member-delete $member $pool
     done
   done
   for listener in `neutron lbaas-listener-list -c id -f value`; do
     neutron lbaas-listener-delete $listener
   done
   for lb in `neutron lbaas-loadbalancer-list -c id -f value`; do
     neutron lbaas-loadbalancer-delete $lb
   done
   ```

3. Delete the stack safely with the openstack stack delete STACKNAME command.

### LBaaS is stuck in Pending state

1. Verify the subnet ID used for deployment. The subnet should match the network attached to the first interface on the instances (such as net01). Use the openstack subnet list command to get a list of subnets.

2. Verify that the public net ID is correct. Use the neutron net-external-list command to find the public net.

3. Verify that neutron-lbaas is installed and configured. The neutron lbaas-loadbalancer-list command must return either 0 or some entries, but not error.
Deploy a Kubernetes cluster on bare metal

This section provides the steps to deploy a Kubernetes cluster on bare metal nodes configured using MAAS with Calico or OpenContrail as a Kubernetes networking plugin.

To automatically deploy a Kubernetes cluster on bare metal nodes:

1. Verify that you have completed the steps described in Prerequisites.
2. Log in to the Jenkins web UI as Administrator.

   Note
   The password for the Administrator is defined in the classes/cluster/<CLUSTER_NAME>/cicd/control/init.yml file of the Reclass model under the openldap_admin_password parameter variable.

3. Depending on your use case, find the k8s_ha_calico heat or k8s_ha_contrail heat job in the global view.
4. Select the Build with Parameters option from the drop-down menu of the selected job.
5. Configure the deployment by setting the following parameters as required:

   Deployment parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK_ON_ERROR</td>
<td>False</td>
<td>If True, Jenkins will stop on any failure and ask either you want to cancel the pipeline or proceed with the execution ignoring the error.</td>
</tr>
</tbody>
</table>
The Jenkins ID of credentials for logging in to the Salt API. For example, salt-credentials. See View credentials details used in Jenkins pipelines for details.

The URL to access the Salt Master node.

Components to install.

The names of the cluster components to test. By default, nothing is tested.

The type of the cluster.

6. Click Build to launch the pipeline.

7. Click Full stage view to track the deployment process.

The following table contains the stages details for the deployment with Calico or OpenContrail as a Kubernetes networking plugin:

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create infrastructure</td>
<td>Creates a base infrastructure using MAAS.</td>
</tr>
</tbody>
</table>
| 2  | Install core infrastructure | 1. Prepares and validates the Salt Master node and Salt Minion nodes. For example, refreshes pillars and synchronizes custom modules.  
|    |                         | 2. Applies the linux,openssh,salt.minion,ntp states to all nodes.      |
Install Kubernetes infrastructure

1. Reads the control plane load-balancer address and applies it to the model.
2. Generates the Kubernetes certificates.
3. Installs the Kubernetes support packages that include Keepalived, HAProxy, Docker, and etcd.

Install the Kubernetes control plane and networking plugins

- For the Calico deployments:
  1. Installs Calico.
  2. Sets up etcd.
  3. Installs the control plane nodes.
- For the OpenContrail deployments:
  1. Installs the OpenContrail infrastructure.
  2. Configures OpenContrail to be used by Kubernetes.
  3. Installs the control plane nodes.

When the pipeline has successfully executed, log in to any Kubernetes ctl node and verify that all nodes have been registered successfully:

```
kubectl get nodes
```

See also

View the deployment details

Deploy ExternalDNS for Kubernetes

ExternalDNS deployed on Mirantis Cloud Platform (MCP) allows you to set up a DNS management server for Kubernetes starting with version 1.7. ExternalDNS enables you to control DNS records dynamically through Kubernetes resources and make Kubernetes resources discoverable through public DNS servers. ExternalDNS synchronizes exposed Kubernetes Services and Ingresses with DNS cloud providers, such as Designate, AWS Route 53, Google CloudDNS, and CoreDNS.

ExternalDNS retrieves a list of resources from the Kubernetes API to determine the desired list of DNS records. It synchronizes the DNS service according to the current Kubernetes status.

ExternalDNS can use the following DNS back-end providers:

- **AWS Route 53** is a highly available and scalable cloud DNS web service. Amazon Route 53 is fully compliant with IPv6.
- **Google CloudDNS** is a highly available, scalable, cost-effective, and programmable DNS service running on the same infrastructure as Google.
• **OpenStack Designate** can use different DNS servers including Bind9 and PowerDNS that are supported by MCP.

• **CoreDNS** is the next generation of SkyDNS that can use etcd to accept updates to DNS entries. It functions as an on-premises open-source alternative to cloud DNS services (DNSaaS). You can deploy CoreDNS with ExternalDNS if you do not have an active DNS back-end provider yet.

This section describes how to configure and set up ExternalDNS on a new or existing MCP Kubernetes-based cluster.

Prepare a DNS back end for ExternalDNS

Depending on your DNS back-end provider, prepare your back end and the metadata model of your MCP cluster before setting up ExternalDNS. If you do not have an active DNS back-end provider yet, you can use CoreDNS that functions as an on-premises open-source alternative to cloud DNS services.

To prepare a DNS back end

Choose from the following options depending on your DNS back end:

• For AWS Route 53:
  1. Log in to your AWS Route 53 console.
  2. Navigate to the AWS Services page.
  3. In the search field, type “Route 53” to find the corresponding service page.
  4. On the Route 53 page, find the DNS management icon and click Get started now.
  5. On the DNS management page, click Create hosted zone.
  6. On the right side of the Create hosted zone window:
     1. Add `<your_mcp_domain.>.local` name.
     2. Choose the Public Hosted Zone type.
     3. Click Create.
     You will be redirected to the previous page with two records of NS and SOA type. Keep the link of this page for verification after the ExternalDNS deployment.
  7. Click Back to Hosted zones.
  8. Locate and copy the Hosted Zone ID in the corresponding column of your recently created hosted zone.
  9. Add this ID to the following template:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "route53:ChangeResourceRecordSets",
                "route53:ListResourceRecordSets",
                "route53:GetHostedZone"
            ],
            "Resource": [
        ```
"arn:aws:route53:::hostedzone/<YOUR_ZONE_ID>" ]
},
{
  "Effect" : "Allow",
  "Action" : [
    "route53:GetChange"
  ],
  "Resource" : [
    "arn:aws:route53:::change/*"
  ]
},
{
  "Effect" : "Allow",
  "Action" : [
    "route53:ListHostedZones"
  ],
  "Resource" : [
    "*"
  ]
}
}

10 Navigate to Services > IAM > Customer Managed Policies.

11 Click Create Policy > Create your own policy.

12 Fill in the required fields:
   • Policy Name field: externaldns
   • Policy Document field: use the JSON template provided in step 9

13 Click Validate Policy.

14 Click Create Policy. You will be redirected to the policy view page.

15 Navigate to Users.

16 Click Add user:
   1. Add a user name: extenaldns.
   2. Select the Programmatic access check box.
   3. Click Next: Permissions.
   4. Select the Attach existing policy directly option.
5. Choose the Customer managed policy type in the Filter drop-down menu.
6. Select the externaldns check box.
7. Click Next: Review.
8. Click Create user.
9. Copy the Access key ID and Secret access key.

• For Google CloudDNS:
  1. Log in to your Google Cloud Platform web console.
  2. Navigate to IAM & Admin > Service accounts > Create service account.
  3. In the Create service account window, configure your new ExternalDNS service account:
     1. Add a service account name.
     2. Assign the DNS Administrator role to the account.
     3. Select the Furnish a new private key check box and the JSON key type radio button.
        The private key is automatically saved on your computer.
  4. Navigate to NETWORKING > Network services > Cloud DNS.
  5. Click CREATE ZONE to create a DNS zone that will be managed by ExternalDNS.
  6. In the Create a DNS zone window, fill in the following fields:
     • Zone name
     • DNS name that must contain your MCP domain address in the <your_mcp_domain>.local format.
  7. Click Create.
     You will be redirected to the Zone details page with two DNS names of the NS and SOA type. Keep this page for verification after the ExternalDNS deployment.

• For Designate:
  1. Log in to the Horizon web UI of your OpenStack environment with Designate.
  2. Create a project with the required admin role as well as generate the access credentials for the project.
  3. Create a hosted DNS zone in this project.

• For CoreDNS, proceed to Configure cluster model for ExternalDNS.
Now, proceed to Configure cluster model for ExternalDNS.

Configure cluster model for ExternalDNS
After you prepare your DNS back end as described in Prepare a DNS back end for ExternalDNS, prepare your cluster model as described below.

To configure the cluster model:

1. Choose from the following options:

   - If you are performing the initial deployment of your MCP Kubernetes cluster:
     1. Use the ModelDesigner UI to create the Kubernetes cluster model. For details, see: Create a deployment metadata model using the Model Designer UI.
     2. While creating the model, select the Kubernetes externaldns enabled check box in the Kubernetes product parameters section.

   - If you are making changes to an existing MCP Kubernetes cluster, proceed to the next step.

2. Open your Git project repository.

3. In classes/cluster/<cluster_name>/kubernetes/control.yml:

   1. If you are performing the initial deployment of your MCP Kubernetes cluster, configure the provider parameter in the snippet below depending on your DNS provider: coredns|aws|google|designate. If you are making changes to an existing cluster, add and configure the snippet below. For example:

```
parameters:
  kubernetes:
    common:
      addons:
        externaldns:
          enabled: True
          namespace: kube-system
          image: mirantis/external-dns:latest
          domain: domain
          provider: coredns
```

   2. Set up the pillar data for your DNS provider to configure it as an add-on. Use the credentials generated while preparing your DNS provider.

      - For Designate:

```
parameters:
  kubernetes:
    common:
      addons:
        externaldns:
          enabled: True
          domain: company.mydomain
          provider: designate
          designate_os_options:
            OS_AUTH_URL: https://keystone_auth_endpoint:5000
```

Mirantis Cloud Platform Deployment Guide

©2020, Mirantis Inc.
OS_PROJECT_DOMAIN_NAME: default
OS_USER_DOMAIN_NAME: default
OS_PROJECT_NAME: admin
OS_USERNAME: admin
OS_PASSWORD: password
OS_REGION_NAME: RegionOne

• For AWS Route 53:

```yaml
parameters:
kubernetes:
  common:
  addons:
    externaldns:
      enabled: True
      domain: company.mydomain
      provider: aws
      aws_options:
        AWS_ACCESS_KEY_ID: XXXXXXXXXXXXXXXXXXXXXXXXX
        AWS_SECRET_ACCESS_KEY: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

• For Google CloudDNS:

```yaml
parameters:
kubernetes:
  common:
  addons:
    externaldns:
```

Note
You can export the credentials from the Google console and process them using the `cat key.json | tr -d 'n'` command.

• For CoreDNS:

```yaml
parameters:
kubernetes:
  common:
```
addons:
coredns:
   enabled: True
   namespace: kube-system
   image: coredns/coredns:latest
etcd:
   operator_image: quay.io/coreos/etcd-operator:v0.5.2
   version: 3.1.8
   base_image: quay.io/coreos/etcd

4. Commit and push the changes to the project Git repository.

5. Log in to the Salt Master node.

6. Update your Salt formulas and the system level of your repository:
   1. Change the directory to /srv/salt/reclass.
   2. Run the `git pull origin master` command.
   3. Run the `salt-call state.sls salt.master` command.
   4. Run the `salt-call state.sls reclass` command.

Now, proceed to Deploy ExternalDNS.

Deploy ExternalDNS

Before you deploy ExternalDNS, complete the steps described in Configure cluster model for ExternalDNS.

To deploy ExternalDNS

Choose from the following options:

- If you are performing the initial deployment of your MCP Kubernetes cluster, deploy a Kubernetes cluster using Heat stack as described in Deploy a Kubernetes cluster on OpenStack. The ExternalDNS will be deployed automatically by the MCP DriveTrain pipeline during the Kubernetes cluster deployment.

- If you are making changes to an existing MCP Kubernetes cluster, apply the following state:

```bash
salt --hard-crash --state-output=mixed --state-verbose=False -C @kubernetes:master state.sls kubernetes.master.kube-addons
```

Once the state is applied, the `kube-addons.sh` script applies the Kubernetes resources and they will shortly appear in the Kubernetes resources list.

Verify ExternalDNS after deployment

After you complete the steps described in Deploy ExternalDNS, verify that ExternalDNS is up and running using the procedures below depending on your DNS back end.

Verify ExternalDNS with Designate back end after deployment
After you complete the steps described in Deploy ExternalDNS, verify that ExternalDNS is successfully deployed with Designate back end using the procedure below. 

To verify ExternalDNS with Designate back end:

1. Log in to any Kubernetes Master node.
2. Source the openrc file of your OpenStack environment:

   ```
   source keystonerc
   ```

   **Note**
   If you use Keystone v3, use the `source keystonercv3` command instead.

3. Open the Designate shell using the `designate` command.
4. Create a domain:

   ```
   domain-create --name nginx.<your_mcp_domain>.local. --email <your_email>
   ```

   Example of system response:

   | Field       | Value                                |
   |-------------+-------------------------------------|
   | description | None                                |
   | created_at  | 2017-10-13T16:23:26.533547           |
   | updated_at  | None                                |
   | email       | designate@example.org               |
   | ttl         | 3600                                |
   | serial      | 1423844606                          |
   | id          | ae59d62b-d655-49a0-ab4b-ea536d845a32 |
   | name        | nginx.virtual-mcp11-k8s-calico.local. |

5. Verify that the domain was successfully created. Use the `id` parameter value from the output of the command described in the previous step. Keep this value for further verification steps.

   For example:

   ```
   record-list ae59d62b-d655-49a0-ab4b-ea536d845a32
   ```

   Example of system response:
6. Start my-nginx:

```bash
kubectl run my-nginx --image=nginx --port=80
```

Example of system response:

```
deployment "my-nginx" created
```

7. Expose my-nginx:

```bash
kubectl expose deployment my-nginx --port=80 --type=ClusterIP
```

Example of system response:

```
service "my-nginx" exposed
```

8. Annotate my-nginx:

```bash
kubectl annotate service my-nginx \"external-dns.alpha.kubernetes.io/hostname=nginx.<your_domain>.local.\"
```

Example of system response:

```
service "my-nginx" annotated
```

9. Verify that the domain was associated with the IP inside a Designate record by running the record-list [id] command. Use the id parameter value from the output of the command described in step 4. For example:

```bash
record-list ae59d62b-d655-49a0-ab4b-ea536d845a32
```

Example of system response:

```
+----+------+---------------------------------------+------------------------+
<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>name</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>NS</td>
<td>nginx.virtual-mcp11-k8s-calico.local.</td>
<td>dns01.bud.mirantis.net.</td>
</tr>
</tbody>
</table>
+----+------+---------------------------------------+------------------------+
Verify ExternalDNS with CoreDNS back end after deployment

After you complete the steps described in Deploy ExternalDNS, verify that ExternalDNS is successfully deployed with CoreDNS back end using the procedure below.

To verify ExternalDNS with CoreDNS back end:

1. Log in to any Kubernetes Master node.
2. Start my-nginx:
   
   ```
kubectl run my-nginx --image=nginx --port=80
   
   Example of system response:
   
   deployment "my-nginx" created
   ```

3. Expose my-nginx:
   
   ```
kubectl expose deployment my-nginx --port=80 --type=ClusterIP
   
   Example of system response:
   
   service "my-nginx" exposed
   ```

4. Annotate my-nginx:
   
   ```
kubectl annotate service my-nginx \ 
"external-dns.alpha.kubernetes.io/hostname=nginx.<your_domain>.local."
   
   Example of system response:
   
   service "my-nginx" annotated
   ```

5. Get the IP of DNS service:
   
   ```
kubectl get svc coredns -n kube-system | awk '{print $2}' | tail -1
   
   Example of system response:
   
   10.254.203.8
6. Choose from the following options:
   
   • If your Kubernetes networking is Calico, run the following command from any Kubernetes Master node.
   • If your Kubernetes networking is OpenContrail, run the following command from any Kubernetes pod.

   \[
   \text{nslookup nginx.<your_domain>.local. <coredns_ip>}
   \]

   Example of system response:

   \[
   \begin{align*}
   \text{Server:} & \quad 10.254.203.8 \quad \text{Address:} \quad 10.254.203.8#53 \\
   \text{Name:} & \quad \text{test.my_domain.local Address:} \quad 10.254.42.128
   \end{align*}
   \]

   Verify ExternalDNS with Google CloudDNS back end after deployment

   After you complete the steps described in Deploy ExternalDNS, verify that ExternalDNS is successfully deployed with Google CloudDNS back end using the procedure below.

   To verify ExternalDNS with Google CloudDNS back end:

   1. Log in to any Kubernetes Master node.
   2. Start my-nginx:

       \[
       \text{kubectl run my-nginx --image=nginx --port=80}
       \]

       Example of system response:

       \[
       \text{deployment "my-nginx" created}
       \]

   3. Expose my-nginx:

       \[
       \text{kubectl expose deployment my-nginx --port=80 --type=ClusterIP}
       \]

       Example of system response:

       \[
       \text{service "my-nginx" exposed}
       \]

   4. Annotate my-nginx:

       \[
       \text{kubectl annotate service my-nginx \\
       "external-dns.alpha.kubernetes.io/hostname=nginx.<your_domain>.local."
       \]

       Example of system response:

       \[
       \text{service "my-nginx" annotated}
       \]
5. Log in to your Google Cloud Platform web console.
6. Navigate to the Cloud DNS > Zone details page.
7. Verify that your DNS zone now has two more records of the A and TXT type. Both records must point to nginx.<your_domain>.local.

Verify ExternalDNS with AWS Route 53 back end after deployment

After you complete the steps described in Deploy ExternalDNS, verify that ExternalDNS is successfully deployed with AWS Route 53 back end using the procedure below.

To verify ExternalDNS with AWS Route 53 back end:
1. Log in to any Kubernetes Master node.
2. Start my-nginx:

```
kubectl run my-nginx --image=nginx --port=80
```

Example of system response:
```
deployment "my-nginx" created
```
3. Expose my-nginx:

```
kubectl expose deployment my-nginx --port=80 --type=ClusterIP
```

Example of system response:
```
service "my-nginx" exposed
```
4. Annotate my-nginx:

```
kubectl annotate service my-nginx "external-dns.alpha.kubernetes.io/hostname=nginx.<your_domain>.local."
```

Example of system response:
```
service "my-nginx" annotated
```
5. Log in to your AWS Route 53 console.
7. Verify that your DNS zone now has two more records of the A and TXT type. Both records must point to nginx.<your_domain>.local.

**Deploy StackLight LMA with the DevOps Portal**
This section explains how to deploy StackLight LMA with the DevOps Portal (OSS) using Jenkins. Before you proceed with the deployment, verify that your cluster level model contains configuration to deploy StackLight LMA as well as OSS. More specifically, check whether you enabled StackLight LMA v2 and OSS as described in Services deployment parameters, and specified all the required parameters for these MCP components as described in StackLight LMA product parameters and OSS parameters.

Note
For production environments, CI/CD should be deployed on a per-customer basis.
For testing purposes, you can use the central Jenkins lab that is available for Mirantis employees only. To be able to configure and execute Jenkins pipelines using the lab, you need to log in to the Jenkins web UI with your Launchpad credentials.

To deploy StackLight LMA with the DevOps Portal:

1. In a web browser, open http://<ip_address>:8081 to access the Jenkins web UI.

   Note
   The IP address is defined in the classes/cluster/<cluster_name>/cicd/init.yml file of the Reclass model under the cicd_control_address parameter variable.

2. Log in to the Jenkins web UI as admin.

   Note
   The password for the admin user is defined in the classes/cluster/<cluster_name>/cicd/control/init.yml file of the Reclass model under the openldap_admin_password parameter variable.

3. Find the Deploy - OpenStack job in the global view.
4. Select the Build with Parameters option from the drop-down menu of the Deploy - OpenStack job.
5. For the STACK_INSTALL parameter, specify the stacklight and oss values.

   Warning
   If you enabled Stacklight LMA and OSS in the Reclass model, you should specify both stacklight and oss to deploy them together. Otherwise, the Runbooks Automation service (Rundeck) will not start due to Salt and Rundeck behavior.
6. Click Build.

7. Once the cluster is deployed, you can access the DevOps Portal at the the IP address specified in the stacklight_monitor_address parameter on port 8800.

See also
- Deploy an OpenStack environment
- View the deployment details

Scale up a Kubernetes cluster

MCP DriveTrain enables you to automatically scale up the number of Kubernetes compute nodes in your Kubernetes cluster deployed on top of AWS or OpenStack.

Note
Currently, only scaling up is supported; scaling down is not available.

To scale up the Kubernetes cluster on AWS:

1. Log in to the Jenkins web UI as Administrator.

Note
The password for the Administrator is defined in the classes/cluster/<CLUSTER_NAME>/cicd/control/init.yml file of the Reclass model under the openldap_admin_password parameter variable.

2. Find the deployment pipeline that you used to successfully deploy your Kubernetes cluster as described in the Deploy a Kubernetes cluster on AWS or Deploy a Kubernetes cluster on OpenStack procedure. You will reuse the existing deployment to scale it up.

3. Select the Build with Parameters option from the drop-down menu of the pipeline.
4. Reconfigure the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK_COMPUTE_COUNT</td>
<td>The number of Kubernetes compute nodes to be deployed by the pipeline.</td>
</tr>
<tr>
<td></td>
<td>Configure as required for your use case.</td>
</tr>
<tr>
<td>STACK_NAME</td>
<td>The Heat stack name to reuse.</td>
</tr>
<tr>
<td>STACK_REUSE</td>
<td>Check the box to reuse the existing Kubernetes deployment that requires</td>
</tr>
<tr>
<td></td>
<td>scaling up.</td>
</tr>
</tbody>
</table>

Deploy - k8s_calico AWS parameters to scale up the cluster

5. Click Build to launch the pipeline.
As a result of the deployment pipeline execution, your existing Kubernetes cluster will be scaled up during the Scale Kubernetes compute nodes stage as configured. The preceding stages of the workflow will be executed as well to ensure proper configuration. However, it will take significantly shorter time to execute these stages, as most of operations has been performed already during the initial deployment of the cluster.

See also
- Deploy a Kubernetes cluster on AWS
- Deploy a Kubernetes cluster on OpenStack
- The Deploy - k8s_calico aws pipeline workflow
- The Deploy - k8s_ha_calico heat pipeline workflow
- View the deployment details

View credentials details used in Jenkins pipelines
MCP uses the Jenkins Credentials Plugin that enables users to store credentials in Jenkins globally. Each Jenkins pipeline can operate only the credential ID defined in the pipeline’s parameters and does not share any security data.

To view the detailed information about all available credentials in the Jenkins UI:

1. Log in to your Jenkins master located at http://<jenkins_master_ip_address>:8081.

Note
The Jenkins master IP address is defined in the classes/cluster/<cluster_name>/cicd/init.yml file of the Reclass model under the cicd_control_address parameter variable.
2. Navigate to the Credentials page from the left navigation menu.

All credentials listed on the Credentials page are defined in the Reclass model. For example, on the system level in the ../../system/jenkins/client/credential/gerrit.yml file.

Examples of users definitions in the Reclass model:

- With the RSA key definition:

```yaml
jenkins:
  client:
    credential:
      gerrit:
        username: ${_param:gerrit_admin_user}
        key: ${_param:gerrit_admin_private_key}
```

- With the open password:

```yaml
jenkins:
  client:
    credential:
      salt:
        username: salt
        password: ${_param:salt_api_password}
```

**View the deployment details**

Once you have enforced a pipeline in CI/CD, you can monitor the progress of its execution on the job progress bar that appears on your screen. Moreover, Jenkins enables you to analyze the details of the deployments process.

To view the deployment details:

1. Log in to the Jenkins web UI.
2. Under Build History on the left, click the number of the build you are interested in.
3. Go to Console Output from the navigation menu to view the the deployment progress.
4. When the deployment succeeds, verify the deployment result in Horizon.

**Note**

The IP address for Horizon is defined in the classes/cluster/<name>/openstack/init.yml file of the Reclass model under the openstack_proxy_address parameter variable.

To troubleshoot an OpenStack deployment:

1. Log in to the Jenkins web UI.
2. Under Build History on the left, click the number of the build you are interested in.
3. Verify Full log to determine the cause of the error.
4. Rerun the deployment with the failed component only. For example, if StackLight LMA fails, run the deployment with only StackLight selected for deployment. Use steps 6-10 of the Deploy an OpenStack environment instruction.
Deploy an MCP cluster manually

Deploy an OpenStack environment manually

This section explains how to manually configure and install software required by your MCP OpenStack environment, such as support services, OpenStack services, and others.

Prepare VMs to install OpenStack

This section instructs you on how to prepare the virtual machines for the OpenStack services installation.

To prepare VMs for a manual installation of an OpenStack environment:

1. Log in to the Salt Master node.
2. Verify that the Salt Minion nodes are synchronized:
   
   ```
   salt '*' saltutil.sync_all
   ```
3. Configure basic operating system settings on all nodes:
   
   ```
   salt '*' state.sls salt.minion,linux,ntp,openssh
   ```

Enable TLS support

To assure the confidentiality and integrity of network traffic inside your OpenStack deployment, you should use cryptographic protective measures, such as the Transport Layer Security (TLS) protocol.

By default, only the traffic that is transmitted over public networks is encrypted. If you have specific security requirements, you may want to configure internal communications to connect through encrypted channels. This section explains how to enable the TLS support for your MCP cluster.

Encrypt internal API HTTP transport with TLS

This section explains how to encrypt the internal OpenStack API HTTP with TLS.

To encrypt the internal API HTTP transport with TLS:

1. Verify that the Keystone, Nova Placement, Cinder, Barbican, Gnocchi, Panko, and Manila API services, whose formulas support using Web Server Gateway Interface (WSGI) templates from Apache, are running under Apache by adding the following classes to your deployment model:

   • In openstack/control.yml:

   ```
   classes:
   ...
   - system.apache.server.site.barbican
   - system.apache.server.site.cinder
   ```
In openstack/telemetry.yml:

```
classes:
  ...
  - system.apache.server.site.gnocchi
  - system.apache.server.site.panko
```

2. Add SSL configuration for each WSGI template by specifying the following parameters:

In openstack/control.yml:

```
parameters:
  _param:
    ...
    apache_proxy_ssl:
      enabled: true
      engine: salt
      authority: "${_param:salt_minion_ca_authority}"
      key_file: "/etc/ssl/private/internal_proxy.key"
      cert_file: "/etc/ssl/certs/internal_proxy.crt"
      chain_file: "/etc/ssl/certs/internal_proxy-with-chain.crt"

  apache_cinder_ssl: ${_param:apache_proxy_ssl}
  apache_keystone_ssl: ${_param:apache_proxy_ssl}
  apache_barbican_ssl: ${_param:apache_proxy_ssl}
  apache_manila_ssl: ${_param:apache_proxy_ssl}
  apache_nova_placement: ${_param:apache_proxy_ssl}
```

In openstack/telemetry.yml:

```
parameters:
  _param:
    ...
    apache_gnocchi_api_address: ${_param:single_address}
    apache_panko_api_address: ${_param:single_address}
    apache_gnocchi_ssl: ${_param:nginx_proxy_ssl}
    apache_panko_ssl: ${_param:nginx_proxy_ssl}
```

3. For services that are still running under Eventlet, configure TLS termination proxy. Such services include Nova, Neutron, Ironic, Glance, Heat, Aodh, and Designate.

Depending on your use case, configure proxy on top of either Apache or NGINX by defining the following classes and parameters:
• In openstack/control.yml:

  • To configure proxy on Apache:

```
classes:
  ...
- system.apache.server.proxy.openstack.designate
- system.apache.server.proxy.openstack.glance
- system.apache.server.proxy.openstack.heat
- system.apache.server.proxy.openstack.ironic
- system.apache.server.proxy.openstack.neutron
- system.apache.server.proxy.openstack.nova

parameters:
  _param:
    ...
    # Configure proxy to redirect request to localhost:
    apache_proxy_openstack_api_address: '${_param:cluster_local_host}'
    apache_proxy_openstack_designate_host: 127.0.0.1
    apache_proxy_openstack_glance_host: 127.0.0.1
    apache_proxy_openstack_heat_host: 127.0.0.1
    apache_proxy_openstack_ironic_host: 127.0.0.1
    apache_proxy_openstack_neutron_host: 127.0.0.1
    apache_proxy_openstack_nova_host: 127.0.0.1
```

• To configure proxy on NGINX:

```
classes:
  ...
- system.nginx.server.single
- system.nginx.server.proxy.openstack_api
- system.nginx.server.proxy.openstack.designate
- system.nginx.server.proxy.openstack.ironic
- system.nginx.server.proxy.openstack.placement

# Delete proxy sites that are running under Apache:
  _param:
    ...
  nginx:
    server:
      site:
        nginx_proxy_openstack_api_keystone: enabled: false
        nginx_proxy_openstack_api_keystone_private: enabled: false
        ...

  # Configure proxy to redirect request to localhost
  _param:
```

©2020, Mirantis Inc.
nginx_proxy_openstack_api_address: ${_param:cluster_local_address}
nginx_proxy_openstack_cinder_host: 127.0.0.1
nginx_proxy_openstack_designate_host: 127.0.0.1
nginx_proxy_openstack_glance_host: 127.0.0.1
nginx_proxy_openstack_heat_host: 127.0.0.1
nginx_proxy_openstack_ironic_host: 127.0.0.1
nginx_proxy_openstack_neutron_host: 127.0.0.1
nginx_proxy_openstack_nova_host: 127.0.0.1

# Add nginx SSL settings:

    _param:
    ...

    nginx_proxy_ssl:
        enabled: true
        engine: salt
        authority: "${_param:salt_minion_ca_authority}"
        key_file: "/etc/ssl/private/internal_proxy.key"
        cert_file: "/etc/ssl/certs/internal_proxy.crt"
        chain_file: "/etc/ssl/certs/internal_proxy-with-chain.crt"

• In openstack/telemetry.yml:

    classes:
        ...
        - system.nginx.server.proxy.openstack_aodh

    parameters:
        _param:
        ...
        nginx_proxy_openstack_aodh_host: 127.0.0.1

4. Edit the openstack/init.yml file:

    1. Add the following parameters to the cluster model:

        parameters:
            _param:
            ...
            cluster_public_protocol: https
            cluster_internal_protocol: https
            aodh_service_protocol: ${_param:cluster_internal_protocol}
            barbican_service_protocol: ${_param:cluster_internal_protocol}
            cinder_service_protocol: ${_param:cluster_internal_protocol}
            designate_service_protocol: ${_param:cluster_internal_protocol}
            glance_service_protocol: ${_param:cluster_internal_protocol}
            gnocchi_service_protocol: ${_param:cluster_internal_protocol}
            heat_service_protocol: ${_param:cluster_internal_protocol}
            ironic_service_protocol: ${_param:cluster_internal_protocol}
2. Depending on your use case, define the following parameters for the OpenStack services to verify that the services running behind TLS proxy are binded to the localhost:

- In openstack/control.yml:

<table>
<thead>
<tr>
<th>OpenStack service</th>
<th>Required configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barbican</td>
<td>bind:</td>
</tr>
<tr>
<td></td>
<td>address: 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>identity:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
<tr>
<td>Cinder</td>
<td>identity:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
<tr>
<td></td>
<td>osapi:</td>
</tr>
<tr>
<td></td>
<td>host: 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>glance:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
<tr>
<td>Designate</td>
<td>identity:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
<tr>
<td></td>
<td>bind:</td>
</tr>
<tr>
<td></td>
<td>api:</td>
</tr>
<tr>
<td></td>
<td>address: 127.0.0.1</td>
</tr>
<tr>
<td>Glance</td>
<td>bind:</td>
</tr>
<tr>
<td></td>
<td>address: 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>identity:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
<tr>
<td></td>
<td>registry:</td>
</tr>
<tr>
<td></td>
<td>protocol: https</td>
</tr>
</tbody>
</table>
### Heat

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>api:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>api_cfn:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>api_cloudwatch:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>identity:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>protocol:</strong></td>
<td>https</td>
</tr>
</tbody>
</table>

### Horizon

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>identity:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>encryption:</strong></td>
<td>ssl</td>
</tr>
</tbody>
</table>

### Ironic

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ironic:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>api:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>127.0.0.1</td>
</tr>
</tbody>
</table>

### Neutron

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>identity:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>protocol:</strong></td>
<td>https</td>
</tr>
</tbody>
</table>

### Nova

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>controller:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>private_address:</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>identity:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>protocol:</strong></td>
<td>https</td>
</tr>
<tr>
<td><strong>network:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>protocol:</strong></td>
<td>https</td>
</tr>
<tr>
<td><strong>glance:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>protocol:</strong></td>
<td>https</td>
</tr>
<tr>
<td><strong>metadata:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>address:</strong></td>
<td>${_param:nova_service_host}</td>
</tr>
</tbody>
</table>

### Panko

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>panko:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>server:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>bind:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>host:</strong></td>
<td>127.0.0.1</td>
</tr>
</tbody>
</table>

- In openstack/telemetry.yml:
parameters:
  _param:
  ...
  aodh
  server:
    bind:
      host: 127.0.0.1
    identity:
      protocol: http
  gnocchi:
    server:
      identity:
        protocol: http
  panko:
    server:
      identity:
        protocol: https

5. Apply the model changes to your deployment:

```shell
salt -C 'I@haproxy' state.apply haproxy
salt -C 'I@apache' state.apply apache
salt 'ctl0*' state.apply kesytone,nova,neutron,heat,glance,cinder,designate,manila,ironic
salt 'mdb0*' state.apply aodh,ceilometer,panko,gnocchi
```

Enable TLS for RabbitMQ and MySQL back ends
Using TLS protects the communications within your cloud environment from tampering and eavesdropping. This section explains how to configure the OpenStack databases back ends to require TLS.

Caution!
TLS for MySQL is supported starting from the Pike OpenStack release.

To encrypt RabbitMQ and MySQL communications:

1. Add the following classes to the cluster model of the nodes where the server is located:
   • For the RabbitMQ server:

```
classes:
  ### Enable tls, contains paths to certs/keys
  - service.rabbitmq.server.ssl
```
Enable TLS for client-server communications
This section explains how to encrypt the communication paths between the OpenStack services and the message queue service (RabbitMQ) as well as the MySQL database.

To enable TLS for client-server communications:

1. For each of the OpenStack services, enable the TLS protocol usage for messaging and database communications by changing the cluster model as shown in the examples below:

   • For a controller node:

      • The database server configuration example:

```yaml
classes:
- system.salt.minion.cert.mysql.server
```

2. Verify that each node trusts the CA certificates that come from the Salt Master node:

```yaml
_param:
salt_minion_ca_host: $cfg01.${_param:cluster_domain}
salt:
  minion:
    trusted_ca_minions:
      - $cfg01.${_param:cluster_domain}
```

3. Deploy RabbitMQ and MySQL as described in Install support services.

4. Apply the changes by executing the salt.minion state:

```bash
salt -I salt:minion:enabled state.apply salt.minion
```

See also

• [Database transport security](#) in the OpenStack Security Guide
• [Messaging security](#) in the OpenStack Security Guide

### Definition of cert/key
- system.salt.minion.cert.rabbitmq_server

• For the MySQL server (Galera cluster):

```yaml
classes:
  ### Enable tls, contains paths to certs/keys
  - service.galera.ssl
  ### Definition of cert/key
  - system.salt.minion.cert.mysql.server
```

Enable TLS for client-server communications
This section explains how to encrypt the communication paths between the OpenStack services and the message queue service (RabbitMQ) as well as the MySQL database.

To enable TLS for client-server communications:

1. For each of the OpenStack services, enable the TLS protocol usage for messaging and database communications by changing the cluster model as shown in the examples below:

   • For a controller node:

   • The database server configuration example:

```yaml
classes:
- system.salt.minion.cert.mysql.server
```
- service.galera.ssl

parameters:
  barbican:
    server:
      database:
        ssl:
          enabled: True
  heat:
    server:
      database:
        ssl:
          enabled: True
  designate:
    server:
      database:
        ssl:
          enabled: True
  glance:
    server:
      database:
        ssl:
          enabled: True
  neutron:
    server:
      database:
        ssl:
          enabled: True
  nova:
    controller:
      database:
        ssl:
          enabled: True
  cinder:
    controller:
      database:
        ssl:
          enabled: True
  volume:
    database:
      ssl:
        enabled: True
  keystone:
    server:
      database:
        ssl:
          enabled: True
• The messaging server configuration example:

classes:
  - service.rabbitmq.server.ssl
  - system.salt.minion.cert.rabbitmq_server

parameters:

designate:
  server:
    message_queue:
      port: 5671
      ssl:
        enabled: True

barbican:
  server:
    message_queue:
      port: 5671
      ssl:
        enabled: True

heat:
  server:
    message_queue:
      port: 5671
      ssl:
        enabled: True

glance:
  server:
    message_queue:
      port: 5671
      ssl:
        enabled: True

neutron:
  server:
    message_queue:
      port: 5671
      ssl:
        enabled: True

nova:
  controller:
    message_queue:
      port: 5671
      ssl:
        enabled: True
cinder:
  controller:
    message_queue:
      port: 5671
    ssl:
      enabled: True
  volume:
    message_queue:
      port: 5671
    ssl:
      enabled: True

keystone:
  server:
    message_queue:
      port: 5671
    ssl:
      enabled: True

• For a compute node, the messaging server configuration example:

```yaml
parameters:
  neutron:
    compute:
      message_queue:
        port: 5671
    ssl:
      enabled: True
  nova:
    compute:
      message_queue:
        port: 5671
    ssl:
      enabled: True
```

• For a gateway node, the messaging configuration example:

```yaml
parameters:
  neutron:
    gateway:
      message_queue:
        port: 5671
    ssl:
      enabled: True
```

2. Refresh the pillar data to synchronize the model update at all nodes:
3. Proceed to Install OpenStack services.

Install support services

Your installation should include a number of support services such as RabbitMQ for messaging; HAProxy for load balancing, proxying, and HA; GlusterFS for storage; and others. This section provides the procedures to install the services and verify they are up and running.

Warning
The HAProxy state should not be deployed prior to Galera. Otherwise, the Galera deployment will fail because of the ports/IP are not available due to HAProxy is already listening on them attempting to bind to 0.0.0.0.

Therefore, verify that your deployment workflow is correct:
1. Keepalived
2. Galera
3. HAProxy

Deploy Keepalived

Keepalived is a framework that provides high availability and load balancing to Linux systems. Keepalived provides a virtual IP address that network clients use as a main entry point to access the CI/CD services distributed between nodes. Therefore, in MCP, Keepalived is used in HA (multiple-node warm-standby) configuration to keep track of services availability and manage failovers.

Warning
The HAProxy state should not be deployed prior to Galera. Otherwise, the Galera deployment will fail because of the ports/IP are not available due to HAProxy is already listening on them attempting to bind to 0.0.0.0.

Therefore, verify that your deployment workflow is correct:
1. Keepalived
2. Galera
3. HAProxy

To deploy Keepalived:
To verify the VIP address:

1. Determine the VIP address for the current environment:

```bash
salt -C 'l@keepalived:cluster' pillar.get keepalived:cluster:instance:VIP:address
```

Example of system output:

```text
ctl03.mk22-lab-basic.local:
  172.16.10.254
ctl02.mk22-lab-basic.local:
  172.16.10.254
ctl01.mk22-lab-basic.local:
  172.16.10.254
```

Note
You can also find the Keepalived VIP address in the following files of the Reclass model:

- `/usr/share/salt-formulas/reclass/service/keepalived/cluster/single.yml`, parameter `keepalived.cluster.instance.VIP.address`
- `/srv/salt/reclass/classes/cluster/<ENV_NAME>/openstack/control.yml`, parameter `cluster_vip_address`

2. Verify if the obtained VIP address is assigned to any network interface on one of the controller nodes:

```bash
salt -C 'l@keepalived:cluster' cmd.run "ip a | grep <ENV_VIP_ADDRESS>"
```

Note
Remember that multiple clusters are defined. Therefore, verify that all of them are up and running.

Deploy NTP
The Network Time Protocol (NTP) is used to properly synchronize services among your OpenStack nodes.
To deploy NTP:

```
salt '*' state.sls ntp
```

See also

Enable NTP authentication

Deploy GlusterFS

GlusterFS is a highly-scalable distributed network file system that enables you to create a reliable and redundant data storage. GlusterFS keeps all important data for Database, Artifactory, and Gerrit in shared storage on separate volumes that makes MCP CI infrastructure fully tolerant to failovers.

To deploy GlusterFS:

```
salt -C 'I@glusterfs:server' state.sls glusterfs.server.service
salt -C 'I@glusterfs:server' state.sls glusterfs.server.setup -b 1
```

To verify GlusterFS:

```
salt -C 'I@glusterfs:server' cmd.run "gluster peer status; gluster volume status" -b 1
```

Deploy RabbitMQ

RabbitMQ is an intermediary for messaging. It provides a platform to send and receive messages for applications and a safe place for messages to live until they are received. All OpenStack services depend on RabbitMQ message queues to communicate and distribute the workload across workers.

To deploy RabbitMQ:

1. Log in to the Salt Master node.
2. Apply the rabbitmq state:

```
salt -C 'I@rabbitmq:server' state.sls rabbitmq
```
3. Verify the RabbitMQ status:

```
salt -C 'I@rabbitmq:server' cmd.run "rabbitmqctl cluster_status"
```

Deploy Galera (MySQL)

Galera cluster is a synchronous multi-master database cluster based on the MySQL storage engine. Galera is an HA service that provides scalability and high system uptime.
Warning
The HAProxy state should not be deployed prior to Galera. Otherwise, the Galera deployment will fail because of the ports/IP are not available due to HAProxy is already listening on them attempting to bind to 0.0.0.0.

Therefore, verify that your deployment workflow is correct:

1. Keepalived
2. Galera
3. HAProxy

To deploy Galera:

1. Log in to the Salt Master node.
2. Apply the galera state:

```
salt -C '@galera:master' state.sls galera
salt -C '@galera:slave' state.sls galera -b 1
```
3. Verify that Galera is up and running:

```
salt -C '@galera:master' mysql.status | grep -A1 wsrep_cluster_size
salt -C '@galera:slave' mysql.status | grep -A1 wsrep_cluster_size
```

Deploy HAProxy

HAProxy is a software that provides load balancing for network connections while Keepalived is used for configuring the IP address of the VIP.

Warning
The HAProxy state should not be deployed prior to Galera. Otherwise, the Galera deployment will fail because of the ports/IP are not available due to HAProxy is already listening on them attempting to bind to 0.0.0.0.

Therefore, verify that your deployment workflow is correct:

1. Keepalived
2. Galera
3. HAProxy

To deploy HAProxy:
Deploy Memcached

Memcached is used for caching data for different OpenStack services such as Keystone, for example.

To deploy Memcached:

```bash
salt -C 'I@memcached:server' state.sls memcached
```

Deploy a DNS back end for Designate

Berkely Internet Name Domain (BIND9) and PowerDNS are the two underlying Domain Name system (DNS) servers that Designate supports out of the box. You can use either new or existing DNS server as a back end for Designate.

Deploy BIND9 for Designate

Berkely Internet Name Domain (BIND9) server can be used by Designate as its underlying back end. This section describes how to configure an existing or deploy a new BIND9 server for Designate.

Configure an existing BIND9 server for Designate

If you already have a running BIND9 server, you can configure and use it for the Designate deployment.

The example configuration below has three predeployed BIND9 servers.

To configure an existing BIND9 server for Designate:

1. Open your BIND9 server UI.
2. Verify that the BIND9 configuration files contain rdnc.key for Designate.

   The following text is an example of /etc/bind/named.conf.local on the managed BIND9 server with the IPs allowed for Designate and rdnc.key:

   ```
   key "designate" {
       algorithm hmac-sha512;
       secret "4pc+X4PDqb2q+5o72dlSm72LM1Ds9X2EYZjqq+nnmS7f/C8H+z0fLLBunoitw=";
   };
   controls {
       inet 10.0.0.3 port 953
       allow {
           172.16.10.101;
           172.16.10.102;
           172.16.10.103;
       }
   }
   ```
3. Open classes/cluster/cluster_name/openstack in your Git project repository.

4. In init.yml, add the following parameters:

```yaml
bind9_node01_address: 10.0.0.1
bind9_node02_address: 10.0.0.2
bind9_node03_address: 10.0.0.3
mysql_designate_password: password
keystone_designate_password: password
designate_service_host: ${_param:openstack_control_address}
designate_bind9_rndc_algorithm: hmac-sha512
designate_bind9_rndc_key: >
  4pc+X4PDqg9b2q+5o72dS72LM1Ds9X2EYjzqg+nmsS7F/C8H+z0flLBunoitw==
designate_domain_id: 5186883b-91fb-4891-bd49-e6769234a8fc
designate_pool_ns_records:
  - hostname: 'ns1.example.org.'
    priority: 10
designate_pool_nameservers:
  - host: ${_param:bind9_node01_address}
    port: 53
  - host: ${_param:bind9_node02_address}
    port: 53
  - host: ${_param:bind9_node03_address}
    port: 53
designate_pool_target_type: bind9
designate_pool_target_masters:
  - host: ${_param:openstack_control_node01_address}
    port: 5354
  - host: ${_param:openstack_control_node02_address}
    port: 5354
  - host: ${_param:openstack_control_node03_address}
    port: 5354
designate_pool_target_options:
  host: ${_param:bind9_node01_address}
  port: 53
  rndc_host: ${_param:bind9_node01_address}
rndc_port: 953
  rndc_key_file: /etc/designate/rndc.key
designate_version: ${_param:openstack_version}
```

5. In control.yml, modify the parameters section. Add targets according to the number of BIND9 servers that will be managed, three in our case.

Example:
6. Add your changes to a new commit.

7. Commit and push the changes.

Once done, proceed to deploy Designate as described in Deploy Designate.

Prepare a deployment model for a new BIND9 server

Before you deploy a BIND9 server as a back end for Designate, prepare your cluster deployment model as described below.

The example provided in this section describes the configuration of the deployment model with two BIND9 servers deployed on separate VMs of the infrastructure nodes.

To prepare a deployment model for a new BIND9 server:

1. Open the classes/cluster/cluster_name/openstack directory in your Git project repository.
2. Create a dns.yml file with the following parameters:
classes:
- system.linux.system.repo.mcp.extra
- system.linux.system.repo.mcp.apt_mirantis.ubuntu
- system.linux.system.repo.mcp.apt_mirantis.saltstack_2016_3
- system.bind.server.single
- cluster.cluster_name.infra

parameters:
linux:
  network:
    interface:
      ens3: ${_param:linux_single_interface}
bind:
server:
  key:
    designate:
      secret: "${_param:designate_bind9_rndc_key}"
      algorithm: "${_param:designate_bind9_rndc_algorithm}"
allow_new_zones: true
query: true
control:
  mgmt:
    enabled: true
bind:
  address: ${_param:single_address}
  port: 953
  allow:
    - ${_param:openstack_control_node01_address}
    - ${_param:openstack_control_node02_address}
    - ${_param:openstack_control_node03_address}
    - ${_param:single_address}
    - 127.0.0.1
keys:
  - designate
client:
  enabled: true
option:
default:
  server: 127.0.0.1
  port: 953
  key: designate
key:
  designate:
    secret: "${_param:designate_bind9_rndc_key}"
    algorithm: "${_param:designate_bind9_rndc_algorithm}"
3. In control.yml, modify the parameters section as follows. Add targets according to the number of the BIND9 servers that will be managed.

```yaml
 designate:
   server:
     backend:
       bind9:
         rndc_key: ${_param:designate_bind9_rndc_key}
         rndc_algorithm: ${_param:designate_bind9_rndc_algorithm}
     pools:
       default:
         description: 'test pool'
         targets:
           default:
             description: 'test target1'
           default1:
             type: ${_param:designate_pool_target_type}
             description: 'test target2'
             masters: ${_param:designate_pool_target_masters}
             options:
               host: ${_param:openstack_dns_node02_address}
               port: 53
               rndc_host: ${_param:openstack_dns_node02_address}
               rndc_port: 953
               rndc_key_file: /etc/designate/rndc.key
```

Note
In the example above, the first target that contains default parameters is defined in openstack/init.yml. The second target is defined explicitly. You can add more targets in this section as required.

4. In init.yml, modify the parameters section.

Example:

```yaml
 openstack_dns_node01_hostname: dns01
 openstack_dns_node02_hostname: dns02
 openstack_dns_node01_deploy_address: 10.0.0.8
 openstack_dns_node02_deploy_address: 10.0.0.9
```
openstack_dns_node01_address: 10.0.0.1
openstack_dns_node02_address: 10.0.0.2
mysql_designate_password: password
keystone_designate_password: password
designate_service_host: ${_param:openstack_control_address}
designate_bind9_rndc_key: >
  4pc+X4PDqb2q+5o72d5s72LM1Ds9X2EYjwgg+nmsS7F/C8H+z0fLLBunotw==
designate_bind9_rndc_algorithm: hmac-sha512
designate_domain_id: 5186883b-91fb-4891-bd49-e6769234a8fc
designate_pool_nameservers:
  - host: '${_param:openstack_dns_node01_address}'
    port: 53
  - host: '${_param:openstack_dns_node02_address}'
    port: 53
designate_pool_target_type: bind9
designate_pool_target_masters:
  - host: '${_param:openstack_control_node01_address}'
    port: 5354
  - host: '${_param:openstack_control_node02_address}'
    port: 5354
  - host: '${_param:openstack_control_node03_address}'
    port: 5354
designate_pool_target_options:
  host: '${_param:openstack_dns_node01_address}'
  port: 53
  rndc_host: '${_param:openstack_dns_node01_address}'
  rndc_port: 953
  rndc_key_file: /etc/designate/rndc.key
designate_version: ${_param:openstack_version}

linux:
  network:
    host:
      dns01:
        address: '${_param:openstack_dns_node01_address}'
        names:
          - '${_param:openstack_dns_node01_hostname}'
          - '${_param:openstack_dns_node01_hostname}.${_param:cluster_domain}'
      dns02:
        address: '${_param:openstack_dns_node02_address}'
        names:
          - '${_param:openstack_dns_node02_hostname}'
          - '${_param:openstack_dns_node02_hostname}.${_param:cluster_domain}'

5. In classes/cluster/cluster_name/infra/kvm.yml, add the following class:
6. In `classes/cluster/cluster_name/infra/config.yml`, modify the `classes` and `parameters` sections.

Example:

- In the `classes` section:

```yaml
classes:
- system.reclass.storage.system.openstack_dns_cluster
```

- In the `parameters` section, add the DNS VMs.

```yaml
reclass:
  storage:
    node:
      openstack_dns_node01:
        params:
          linux_system_codename: xenial
          deploy_address: ${_param:openstack_database_node03_deploy_address}
      openstack_dns_node02:
        params:
          linux_system_codename: xenial
          deploy_address: ${_param:openstack_dns_node01_deploy_address}
      openstack_message_queue_node01:
        params:
          linux_system_codename: xenial
```

7. Commit and push the changes.

Once done, proceed to deploy the BIND9 server service as described in Deploy a new BIND9 server for Designate.

**Deploy a new BIND9 server for Designate**

After you configure the Reclass model for a BIND9 server as the back end for Designate, proceed to deploying the BIND9 server service as described below.

To deploy a BIND9 server service:

1. Log in to the Salt Master node.

2. Configure basic operating system settings on the DNS nodes:

   ```bash
   salt -C 'I@bind:server' state.sls linux,ntp,openssh
   ```
3. Apply the following state:

```
salt -C 'I@bind:server' state.sls bind
```

Once done, proceed to deploy Designate as described in Deploy Designate.

Deploy PowerDNS for Designate

PowerDNS server can be used by Designate as its underlying back end. This section describes how to configure an existing or deploy a new PowerDNS server for Designate.

The default PowerDNS configuration for Designate uses the Designate worker role. If you need live synchronization of DNS zones between Designate and PowerDNS servers, you can configure Designate with the pool_manager role. The Designate Pool Manager keeps records consistent across the Designate database and the PowerDNS servers. For example, if a record was removed from the PowerDNS server due to a hard disk failure, this record will be automatically restored from the Designate database.

Configure an existing PowerDNS server for Designate

If you already have a running PowerDNS server, you can configure and use it for the Designate deployment.

The example configuration below has three predeployed PowerDNS servers.

To configure an existing PowerDNS server for Designate:

1. Open your PowerDNS server UI.
2. In etc/powerdns/pdns.conf, modify the following parameters:
   - allow-axfr-ips - must list the IPs of the Designate nodes, which will be located on the OpenStack API nodes
   - api-key - must coincide with the designate_pdns_api_key parameter for Designate in the Reclass model
   - webserver - must have the value yes
   - webserver-port - must coincide with the powerdns_webserver_port parameter for Designate in the Reclass model
   - api - must have the value yes to enable management through API
   - disable-axfr - must have the value no to enable the axfr zone updates from the Designate nodes

   Example:

   ```
   allow-axfr-ips=172.16.10.101,172.16.10.102,172.16.10.103,127.0.0.1
   allow-recursion=127.0.0.1
   api-key=VxK9cM1FL5Ae
   api=yes
   config-dir=/etc/powerdns
   daemon=yes
   default-soa-name=a.very.best.power.dns.server
   disable-axfr=no
   guardian=yes
   include-dir=/etc/powerdns/pdns.d
   ```
3. Open the classes/cluster/cluster_name/openstack directory in your Git project repository.

4. In init.yml, add the following parameters:

```yaml
powerdns_node01_address: 10.0.0.1
powerdns_node02_address: 10.0.0.2
powerdns_node03_address: 10.0.0.3
powerdns_webserver_password: gJ6n3gVaYP8eS
powerdns_webserver_port: 8081
mysql_designate_password: password
keystone_designate_password: password
designate_service_host: ${_param:openstack_control_address}
designate_domain_id: 5186883b-91fb-4891-bd49-e6769234a8fc
designate_pdns_api_key: VxK9cMIFL5Ae
designate_pdns_api_endpoint: >
  "http://${_param:powerdns_node01_address}:${_param:powerdns_webserver_port}"
designate_pool_ns_records:
  - hostname: 'ns1.example.org.'
    priority: 10
designate_pool_nameservers:
  - host: ${_param:powerdns_node01_address}
    port: 53
  - host: ${_param:powerdns_node02_address}
    port: 53
  - host: ${_param:powerdns_node03_address}
    port: 53
designate_pool_target_type: pdns4
designate_pool_target_masters:
  - host: ${_param:openstack_control_node01_address}
    port: 5354
  - host: ${_param:openstack_control_node02_address}
    port: 5354
  - host: ${_param:openstack_control_node03_address}
    port: 5354
```

---

```
launch=
local-address=10.0.0.1
local-port=53
master=no
setgid=pdns
setuid=pdns
slave=yes
soa-minimum-ttl=3600
socket-dir=/var/run
version-string=powerdns
webserver=yes
webserver-address=10.0.0.1
webserver-password=gJ6n3gVaYP8eS
webserver-port=8081
```
designate_pool_target_options:
  host: ${param:powerdns_node01_address}
  port: 53
  api_token: ${param:designate_pdns_api_key}
  api_endpoint: ${param:designate_pdns_api_endpoint}
  designate_version: ${param:openstack_version}

5. In control.yml, modify the parameters section. Add targets according to the number of PowerDNS servers that will be managed, three in our case.

Example:

```
designate:
  server:
    backend:
      pdns4:
        api_token: ${param:designate_pdns_api_key}
        api_endpoint: ${param:designate_pdns_api_endpoint}
    pools:
      default:
        description: 'test pool'
        targets:
          default:
            description: 'test target1'
            default1:
              type: ${param:designate_pool_target_type}
              description: 'test target2'
              masters: ${param:designate_pool_target_masters}
              options:
                host: ${param:powerdns_node02_address}
                port: 53
                api_endpoint: >
                  "http://${param:${param:powerdns_node02_address}}:
                  ${param:powerdns_webserver_port}"
                api_token: ${param:designate_pdns_api_key}
            default2:
              type: ${param:designate_pool_target_type}
              description: 'test target3'
              masters: ${param:designate_pool_target_masters}
              options:
                host: ${param:powerdns_node03_address}
                port: 53
                api_endpoint: >
                  "http://${param:powerdns_node03_address}:
                  ${param:powerdns_webserver_port}"
                api_token: ${param:designate_pdns_api_key}
```

Once done, proceed to deploy Designate as described in Deploy Designate.
Prepare a deployment model for a new PowerDNS server with the worker role

Before you deploy a PowerDNS server as a back end for Designate, prepare your deployment model with the default Designate worker role as described below.

If you need live synchronization of DNS zones between Designate and PowerDNS servers, configure Designate with the pool_manager role as described in Prepare a deployment model for a new PowerDNS server with the pool_manager role.

The examples provided in this section describe the configuration of the deployment model with two PowerDNS servers deployed on separate VMs of the infrastructure nodes.

To prepare a deployment model for a new PowerDNS server:

1. Open the classes/cluster/cluster_name/openstack directory of your Git project repository.
2. Create a dns.yml file with the following parameters:

```yaml
classes:
  - system.powerdns.server.single
  - cluster.cluster_name.infra
parameters:
  linux:
    network:
      interface:
        ens3: ${_param:linux_single_interface}
  host:
    dns01:
      address: ${_param:openstack_dns_node01_address}
      names:
        - dns01
        - dns01.${_param:cluster_domain}
    dns02:
      address: ${_param:openstack_dns_node02_address}
      names:
        - dns02
        - dns02.${_param:cluster_domain}
  powerdns:
    server:
      enabled: true
      bind:
        address: ${_param:single_address}
        port: 53
      backend:
        engine: sqlite
        dbname: pdns.sqlite3
        dbpath: /var/lib/powerdns
      api:
        enabled: true
        key: ${_param:designate_pdns_api_key}
    webserver:
      enabled: true
```
3. In init.yml, define the following parameters:

Example:

```yaml
address: ${_param:single_address}
port: ${_param:powerdns_webserver_port}
password: ${_param:powerdns_webserver_password}
axfr_ips:
  - ${_param:openstack_control_node01_address}
  - ${_param:openstack_control_node02_address}
  - ${_param:openstack_control_node03_address}
  - 127.0.0.1
```

4. In control.yml, define the following parameters in the parameters section:

Example:

```yaml
openstack_dns_node01_address: 10.0.0.1
openstack_dns_node02_address: 10.0.0.2
powerdns_webserver_password: gJ6n3gVaYP8eS
powerdns_webserver_port: 8081
mysql_designate_password: password
keystone_designate_password: password
designate_service_host: ${_param:openstack_control_address}
designate_domain_id: 5186883b-91fb-4891-bd49-e6769234a8fc
designate_pdns_api_key: VxK9cMFL5Ae
designate_pdns_api_endpoint: >
  "http://${_param:openstack_dns_node01_address}:${_param:powerdns_webserver_port}"
designate_pool_ns_records:
  - hostname: 'ns1.example.org.'
    priority: 10
designate_pool_nameservers:
  - host: ${_param:openstack_dns_node01_address}
    port: 53
designate_pool_target_type: pdns4
designate_pool_target_masters:
  - host: ${_param:openstack_control_node01_address}
    port: 5354
designate_pool_target_options:
  - host: ${_param:openstack_control_node01_address}
    port: 53
    api_token: ${_param:designate_pdns_api_key}
    api_endpoint: ${_param:designate_pdns_api_endpoint}
designate_version: ${_param:openstack_version}
designate_worker_enabled: true
```
designate:
  worker:
    enabled: ${_param:designate_worker_enabled}
  server:
    backend:
      pdns4:
        api_token: ${_param:designate_pdns_api_key}
        api_endpoint: ${_param:designate_pdns_api_endpoint}
  pools:
    default:
      description: 'test pool'
      targets:
        default:
          description: 'test target1'
        default1:
          type: ${_param:designate_pool_target_type}
          description: 'test target2'
          masters: ${_param:designate_pool_target_masters}
          options:
            host: ${_param:openstack_dns_node02_address}
            port: 53
            api_endpoint: >
            "http://${_param:openstack_dns_node02_address}:${_param:powerdns_webserver_port}"
            api_token: ${_param:designate_pdns_api_key}

5. In classes/cluster/cluster_name/infra/kvm.yml, modify the classes and parameters sections.
   Example:
   - In the classes section:

   ```yaml
   classes:
   - system.salt.control.cluster.openstack_dns_cluster
   ```
   - In the parameters section, add the DNS parameters for VMs with the required location
     of DNS VMs on kvm nodes and the planned resource usage for them.

   ```yaml
   salt:
     control:
       openstack.dns:
         cpu: 2
         ram: 2048
         disk_profile: small
         net_profile: default
     cluster:
       internal:
         node:
           dns01:
             provider: kvm01.${_param:cluster_domain}
   ```
6. In classes/cluster/cluster_name/infra/config.yml, modify the classes and parameters sections.

Example:

- In the classes section:

```yaml
classes:
  - system.reclass.storage.system.openstack_dns_cluster
```

- In the parameters section, add the DNS VMs. For example:

```yaml
reclass:
  storage:
    node:
      openstack_dns_node01:
        params:
          linux_system_codename: xenial
      openstack_dns_node02:
        params:
          linux_system_codename: xenial
```

7. Commit and push the changes.

Once done, proceed to deploy the PowerDNS server service as described in Deploy a new PowerDNS server for Designate.

Prepare a deployment model for a new PowerDNS server with the pool_manager role

If you need live synchronization of DNS zones between Designate and PowerDNS servers, you can configure Designate with the pool_manager role as described below. The Designate Pool Manager keeps records consistent across the Designate database and the PowerDNS servers. For example, if a record was removed from the PowerDNS server due to a hard disk failure, this record will be automatically restored from the Designate database.

To configure a PowerDNS server with the default Designate worker role, see Prepare a deployment model for a new PowerDNS server with the worker role.

The examples provided in this section describe the configuration of the deployment model with two PowerDNS servers deployed on separate VMs of the infrastructure nodes.

To prepare a model for a new PowerDNS server with the pool_manager role:

1. Open the classes/cluster/cluster_name/openstack directory of your Git project repository.
2. Create a dns.yml file with the following parameters:

```yaml
classes:
  - system.powerdns.server.single
```
- cluster.cluster_name.infra

```yaml
parameters:
  linux:
    network:
      interface:
        ens3: ${_param:linux_single_interface}
  host:
    dns01:
      address: ${_param:openstack_dns_node01_address}
      names:
        - dns01
        - dns01.${_param:cluster_domain}
    dns02:
      address: ${_param:openstack_dns_node02_address}
      names:
        - dns02
        - dns02.${_param:cluster_domain}
  powerdns:
    server:
    bind:
    backend:
      engine: sqlite
      dbname: pdns.sqlite3
      dbpath: /var/lib/powerdns
    api:
    overwrite_supermasters: ${_param:powerdns_supermasters}
    supermasters:
      - ${_param:powerdns_supermasters}
    webserver:
    enabled: true
    address: ${_param:single_address}
    port: ${_param:powerdns_webserver_port}
    password: ${_param:powerdns_webserver_password}
    axfr_ips:
      - ${_param:openstack_control_node01_address}
      - ${_param:openstack_control_node02_address}
      - ${_param:openstack_control_node03_address}
      - 127.0.0.1
```

3. In init.yml, define the following parameters:

Example:
**openstack_dns_node01_address**: 10.0.0.1  
**openstack_dns_node02_address**: 10.0.0.2  
**powerdns_axfr_ips**:  
  - `$(_param:openstack_control_node01_address)`  
  - `$(_param:openstack_control_node02_address)`  
  - `$(_param:openstack_control_node03_address)`  
  - 127.0.0.1  
**powerdns_supermasters**:  
  - **ip**: `$(_param:openstack_control_node01_address)`  
    - name server: ns1.example.org  
    - account: master  
  - **ip**: `$(_param:openstack_control_node02_address)`  
    - name server: ns2.example.org  
    - account: master  
  - **ip**: `$(_param:openstack_control_node03_address)`  
    - name server: ns3.example.org  
    - account: master  
**powerdns_overwrite_supermasters**: True  
**powerdns_webserver_password**: gj6n3gVaYP8eS  
**powerdns_webserver_port**: 8081  
**mysql_designate_password**: password  
**keystone_designate_password**: password  
**designate_service_host**: `$(_param:openstack_control_address)`  
**designate_domain_id**: 5186883b-91fb-4891-bd49-e6769234a8fc  
**designate_mdns_address**: 0.0.0.0  
**designate_mdns_port**: 53  
**designate_pdns_api_key**: VxK9cMIFL5Ae  
**designate_pdns_api_endpoint**: >  
  "http://${_param:openstack_dns_node01_address}:${_param:powerdns_webserver_port}"
**designate_pool_manager_enabled**: True  
**designate_pool_manager_periodic_sync_interval**: '120'  
**designate_pool_ns_records**:  
  - **host**: 'ns1.example.org,'  
    - priority: 10  
  - **host**: 'ns2.example.org,'  
    - priority: 20  
  - **host**: 'ns3.example.org,'  
    - priority: 30  
**designate_pool_nameservers**:  
  - **host**: `$(_param:openstack_dns_node01_address)`  
    - port: 53  
  - **host**: `$(_param:openstack_dns_node02_address)`  
    - port: 53  
**designate_pool_target_type**: pdns4  
**designate_pool_target_masters**:  
  - **host**: `$(_param:openstack_control_node01_address)`  
    - port: `$(_param:designate_mdns_port)`  
  - **host**: `$(_param:openstack_control_node02_address)`
4. In control.yml, define the following parameters in the parameters section:

Example:

```yaml
designate:
  pool_manager:
    enabled: ${_param:designate_pool_manager_enabled}
    periodic_sync_interval: ${_param:designate_pool_manager_periodic_sync_interval}
  server:
    backend:
      pdns4:
        api_token: ${_param:designate_pdns_api_key}
        api_endpoint: ${_param:designate_pdns_api_endpoint}
    mdns:
      address: ${_param:designate_mdns_address}
      port: ${_param:designate_mdns_port}
  pools:
    default:
      description: 'test pool'
    targets:
      default:
        description: 'test target1'
      default1:
        type: ${_param:designate_pool_target_type}
        description: 'test target2'
        masters: ${_param:designate_pool_target_masters}
        options:
          host: ${_param:openstack_dns_node02_address}
          port: 53
          api_endpoint: >
            "http://${_param:openstack_dns_node02_address}:${_param:powerdns_webserver_port}"
          api_token: ${_param:designate_pdns_api_key}
```

5. In classes/cluster/cluster_name/infra/kvm.yml, modify the classes and parameters sections.

Example:

- In the classes section:
In the parameters section, add the DNS parameters for VMs with the required location of DNS VMs on the kvm nodes and the planned resource usage for them.

```yaml
salt:
  control:
    openstack.dns:
      cpu: 2
      ram: 2048
      disk_profile: small
      net_profile: default
    cluster:
      internal:
        node:
          dns01:
            provider: kvm01.${_param:cluster_domain}
          dns02:
            provider: kvm02.${_param:cluster_domain}
```

6. In classes/cluster/cluster_name/infra/config.yml, modify the classes and parameters sections.

   Example:

   ```yaml
   classes:
     - system.reclass.storage.system.openstack_dns_cluster
   
   • In the classes section:

   ```yaml
   reclass:
     storage:
       node:
         openstack_dns_node01:
           params:
             linux_system_codename: xenial
         openstack_dns_node02:
           params:
             linux_system_codename: xenial
   ```

7. Commit and push the changes.

Once done, proceed to deploy the PowerDNS server service as described in Deploy a new PowerDNS server for Designate.

Deploy a new PowerDNS server for Designate
After you configure the Reclass model for PowerDNS server as a back end for Designate, proceed to deploying the PowerDNS server service as described below.

To deploy a PowerDNS server service:

1. Log in to the Salt Master node.
2. Configure basic operating system settings on the DNS nodes:
   
   ```
   salt -C 'I@powerdns:server' state.sls linux,ntp,openssh
   ```

3. Apply the following state:
   
   ```
   salt -C 'I@powerdns:server' state.sls powerdns
   ```

Once done, you can proceed to deploy Designate as described in Deploy Designate.

### See also

- Deploy Designate
- BIND9 documentation
- PowerDNS documentation
- Plan the Domain Name System

## Install OpenStack services

Many of the OpenStack service states make changes to the databases upon deployment. To ensure proper deployment and to prevent multiple simultaneous attempts to make these changes, deploy a service states on a single node of the environment first. Then, you can deploy the remaining nodes of this environment.

Keystone must be deployed before other services. Following the order of installation is important, because many of the services have dependencies of the others being in place.

### Deploy Keystone

To deploy Keystone:

1. Set up the Keystone service:
   
   ```
   salt -C 'I@keystone:server' state.sls keystone.server -b 1
   ```

2. Populate keystone services/tenants/admins:
   
   ```
   salt -C 'I@keystone:client' state.sls keystone.client
   salt -C 'I@keystone:server' cmd.run "./root/keystonerc; openstack service list"
   ```
Deploy Glance

The OpenStack Image service (Glance) provides a REST API for storing and managing virtual machine images and snapshots.

To deploy Glance:

1. Install Glance and verify that GlusterFS clusters exist:

   ```
salt -C 'I@glance:server' state.sls glance -b 1
salt -C 'I@glusterfs:client' state.sls glusterfs.client
   ```

2. Update Fernet tokens before doing request on the Keystone server. Otherwise, you will get the following error: No encryption keys found; run keystone-manage fernet_setup to bootstrap one:

   ```
salt -C 'I@keystone:server' state.sls keystone.server
salt -C 'I@keystone:server' cmd.run "./root/keystonerc; glance image-list"
   ```

Deploy Nova

To deploy the Nova:

1. Install Nova:

   ```
salt -C 'I@nova:controller' state.sls nova -b 1
salt -C 'I@keystone:server' cmd.run "./root/keystonerc; nova service-list"
   ```

2. On one of the controller nodes, verify that the Nova services are enabled and running:

   ```
root@cfg01:~# ssh ctl01 "source keystonerc; nova service-list"
   ```

Deploy Cinder

To deploy Cinder:

1. Install Cinder:

   ```
salt -C 'I@cinder:controller' state.sls cinder -b 1
   ```

2. On one of the controller nodes, verify that the Cinder service is enabled and running:

   ```
salt -C 'I@keystone:server' cmd.run "./root/keystonerc; cinder list"
   ```
Note
If the MCP version of your deployment is 2018.8.0, 2018.8.0-milestone1, or earlier, the Cinder Salt formula will deploy HA back ends if they exist, as a separate single HA back end due to the known issue in the formula. To resolve the issue:

1. Log in to the Salt Master node.
2. Update the HA host name of all volumes:

```
cinder-manage volume update_host \
--currenthost <single_bugged_host_from_HA>@<backend> \
--newhost <new_HA_host>@<backend>#<POOL>
```

Define POOL as described in the OpenStack official documentation. For example:

```
cinder-manage volume update_host --currenthost ctl01@ceph \
--newhost ceph@ceph#ceph
```

Deploy Neutron
To install Neutron:

```
salt -C 'I@neutron:server' state.sls neutron -b
salt -C 'I@neutron:gateway' state.sls neutron
salt -C 'I@keystone:server' cmd.run "./root/keystonerc; neutron agent-list"
```

Note
For installations with the OpenContrail setup, see Deploy OpenContrail manually.

Seealso
MCP Operations Guide: Configure Neutron OVS

Deploy Horizon
To install Horizon:
Deploy Tenant Telemetry

Tenant Telemetry collects metrics about the OpenStack resources and provides this data through the APIs. When deploying Tenant Telemetry, consider the following:

- If your OpenStack version is prior to Pike, deploy Tenant Telemetry that uses the legacy StackLight LMA back ends.
- If your OpenStack version is Pike, deploy the standalone Tenant Telemetry that uses its own back ends.

**Important**

Standalone Tenant Telemetry does not support integration with StackLight LMA.

Deploy Tenant Telemetry that uses the legacy StackLight LMA back ends

If your OpenStack version is prior to Pike, follow the procedure below to deploy Tenant Telemetry. In this case, Tenant Telemetry will use the legacy StackLight LMA back ends such as Heka and InfluxDB.

**Important**

Tenant Telemetry that uses the legacy StackLight LMA back ends is deprecated for OpenStack Pike.

To deploy Tenant Telemetry:

1. On the controller nodes, install the Ceilometer agents and Aodh:

   ```
salt -C 'I@ceilometer:server' state.sls ceilometer -b 1
   salt -C 'I@aodh:server' state.sls aodh -b 1
   ```

2. On the compute nodes, install Ceilometer:

   ```
salt -C 'I@ceilometer:agent' state.sls ceilometer
   ```

3. On the Salt Master node, install Heka for Ceilometer:

   1. Restart salt-minion to make sure that it uses the latest Jinja library:
salt '*' --async service.restart salt-minion

2. Clean the Salt mine:

salt "*#" mine.flush

3. Clean the grains files to make sure that you start from a clean state:

salt "*#" file.remove /etc/salt/grains.d/heka
salt "*#" file.remove /etc/salt/grains

4. Update the Salt mine:

salt "*#" state.sls salt.minion.grains
salt "*#" saltutil.refresh_modules
salt "*#" mine.update

5. Install ceilometer collector:

salt -C "I@heka:ceilometer_collector:enabled" state.sls heka.ceilometer_collector

6. Restart ceilometer collector:

salt -C "I@heka:ceilometer_collector:enabled" service.restart ceilometer_collector

4. Install back ends for Ceilometer as described in Install back ends for StackLight LMA.

Note
Tenant Telemetry does not have any Grafana or Kibana dashboards, therefore, you can skip the corresponding steps. The databases names and passwords for Ceilometer are defined in the system Reclass model. For details, see example.

Deploy standalone Tenant Telemetry

Important
Standalone Tenant Telemetry deployment is supported starting from the Pike OpenStack release and does not support integration with StackLight LMA. If your OpenStack version is prior to Pike, deploy Tenant Telemetry as described in Deploy Tenant Telemetry that uses the legacy StackLight LMA back ends.
Follow the procedure below to deploy standalone Tenant Telemetry. In this case, Tenant Telemetry will use its own back ends, such as Gnocchi and Panko.

To deploy standalone Tenant Telemetry:

1. Log in to the Salt Master node.
2. To use the Redis cluster as coordination back end and storage for Gnocchi, deploy Redis master:
   
   ```
   salt -C 'I@redis:cluster:role:master' state.sls redis
   ```

3. Deploy Redis on all servers:
   
   ```
   salt -C 'I@redis:server' state.sls redis
   ```

4. Deploy Gnocchi:
   
   ```
   salt -C 'I@gnocchi:server' state.sls gnocchi -b 1
   ```

5. Deploy Panko:
   
   ```
   salt -C 'I@panko:server' state.sls panko -b 1
   ```

6. Deploy Aodh and Ceilometer:
   
   ```
   salt -C 'I@ceilometer:server' state.sls ceilometer -b 1
   salt -C 'I@ceilometer:agent' state.sls ceilometer -b 1
   salt -C 'I@aodh:server' state.sls aodh -b 1
   ```

Deploy Designate

Designate supports underlying DNS servers, such as BIND9 and PowerDNS. You can use either a new or an existing DNS server as a back end for Designate. By default, Designate is deployed on three OpenStack API VMs of the VCP nodes.

Prepare a deployment model for the Designate deployment

Before you deploy Designate with a new or existing BIND9 or PowerDNS server as a back end, prepare your cluster deployment model by making corresponding changes in your Git project repository.

To prepare a deployment model for the Designate deployment:

1. Verify that you have configured and deployed a DNS server as a back end for Designate as described in Deploy a DNS back end for Designate.
2. Open the `classes/cluster/<cluster_name>/openstack/` directory in your Git project repository.
3. In `control_init.yml`, add the following parameter in the classes section:
4. In control.yml, add the following parameter in the classes section:

```yaml
classes:
- system.keystone.client.service.designate
```

5. In database.yml, add the following parameter in the classes section:

```yaml
classes:
- system.designate.server.cluster
```

6. Add your changes to a new commit.

7. Commit and push the changes.

Once done, proceed to Install Designate.

Install Designate

This section describes how to install Designate on a new or existing MCP cluster.

Before you proceed to installing Designate:

1. Configure and deploy a DNS back end for Designate as described in Deploy a DNS back end for Designate.
2. Prepare your cluster model for the Designate deployment as described in Prepare a deployment model for the Designate deployment.

To install Designate on a new MCP cluster:

1. Log in to the Salt Master node.
2. Apply the following state:

```
salt -C 'I@designate:server' state.sls designate -b 1
```

To install Designate on an already deployed MCP cluster:

1. Log in to the Salt Master node.
2. Refresh Salt pillars:

```
salt '*' saltutil.pillar_refresh
```
3. Create databases for Designate by applying the mysql state:

```
salt -C 'I@galera:master' state.sls galera
```
4. Create the HAProxy configuration for Designate:
salt -C 'I@haproxy:proxy' state.sls haproxy

5. Create endpoints for Designate in Keystone:
salt -C 'I@keystone:client' state.sls keystone.client

6. Apply the designate state:
salt -C 'I@designate:server' state.sls designate -b 1

7. Verify that the Designate services are up and running:
salt -C 'I@designate:server' cmd.run ". /root/keystonercv3; openstack dns service list"

Example of the system response extract:

<table>
<thead>
<tr>
<th>id</th>
<th>hostname</th>
<th>service_name</th>
<th>status</th>
<th>stats</th>
<th>capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>680902ef-380a-...</td>
<td>ctl02</td>
<td>worker</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e96dffc1-dab2-...</td>
<td>ctl01</td>
<td>central</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3c0d2310-d852-...</td>
<td>ctl02</td>
<td>api</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4c807478-f545-...</td>
<td>ctl02</td>
<td>mdns</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>b66305e3-a75f-...</td>
<td>ctl02</td>
<td>central</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e96dffc1-dab2-...</td>
<td>ctl01</td>
<td>central</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3c0d2310-d852-...</td>
<td>ctl02</td>
<td>mdns</td>
<td>UP</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
+-------------------+---------+-------------+-------+-------+-------------+

See also
Designate operations
Deploy Barbican

MCP enables you to integrate LBaaSv2 Barbican to OpenContrail. Barbican is an OpenStack service that provides a REST API for secured storage as well as for provisioning and managing of secrets such as passwords, encryption keys, and X.509 certificates.

Barbican requires a back end to store secret data in its database. If you have an existing Dogtag back end, deploy and configure Barbican with it as described in Deploy Barbican with the Dogtag back end. Otherwise, deploy a new Dogtag back end as described in Deploy Dogtag. For testing purposes, you can use the simple_crypto back end.

Deploy Dogtag

Dogtag is one of the Barbican plugins that represents a back end for storing symmetric keys, for example, for volume encryption, as well as passwords, and X.509 certificates.

To deploy the Dogtag back end for Barbican:

1. Open the classes/cluster/<cluster_name>/ directory of your Git project repository.
2. In openstack/control.yml, add the Dogtag class and specify the required parameters. For example:

```yaml
classes:
- system.dogtag.server.cluster

parameters:
  _param:
    dogtag_master_host: ${_param:openstack_control_node01_hostname}.${_param:cluster_domain}
    haproxy_dogtag_bind_port: 8444
    cluster_dogtag_port: 8443
    # Dogtag listens on 8443 but there is no way to bind it to a
    # Specific IP, as in this setup Dogtag is installed on ctl nodes
    # Change port on haproxy side to avoid binding conflict.
    haproxy_dogtag_bind_port: 8444
    cluster_dogtag_port: 8443
    dogtag_master_host: ctl01.${linux:system:domain}
    dogtag_pki_admin_password: workshop
    dogtag_pki_client_database_password: workshop
    dogtag_pki_client_pkcs12_password: workshop
```
3. In classes/cluster/os-ha-ovs/infra/config.yml:

1. Add the salt/master/formula/pkg/dogtag class to the classes section.
2. Specify the dogtag_cluster_role: master parameter in the openstack_control_node01 section, and the dogtag_cluster_role: slave parameter in the openstack_control_node02 and openstack_control_node03 sections. For example:

```yaml
classes:
- salt/master/formula/pkg/dogtag
...

node:
openstack_control_node01:
  classes:
  - service.galera.master.cluster
  - service.dogtag.server.cluster.master
  params:
    mysql_cluster_role: master
    linux_system_codename: xenial
    dogtag_cluster_role: master
openstack_control_node02:
  classes:
  - service.galera.slave.cluster
  - service.dogtag.server.cluster.slave
  params:
    mysql_cluster_role: slave
    linux_system_codename: xenial
    dogtag_cluster_role: slave
openstack_control_node03:
  classes:
  - service.galera.slave.cluster
  - service.dogtag.server.cluster.slave
  params:
    mysql_cluster_role: slave
    linux_system_codename: xenial
    dogtag_cluster_role: slave
```

4. Commit and push the changes to the project Git repository.
5. Log in to the Salt Master node.
6. Update your Salt formulas at the system level:
   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.
7. Apply the following states:
   ```
salt -C '@salt:master' state.apply salt.reclass
salt -C '@dogtag:server and *01*' state.apply dogtag.server
salt -C '@dogtag:server' state.apply dogtag.server
salt -C '@haproxy:proxy' state.apply haproxy
   ```
8. Proceed to Deploy Barbican with the Dogtag back end.

Note
If the dogtag:export_pem_file_path variable is defined, the system imports kra admin certificate to the defined .pem file and to the Salt Mine dogtag_admin_cert variable. After that, Barbican and other components can use kra admin certificate.

See also
Dogtag OpenStack documentation

Deploy Barbican with the Dogtag back end
You can deploy and configure Barbican to work with the private Key Recovery Agent (KRA) Dogtag back end.

Before you proceed with the deployment, make sure that you have a running Dogtag back end. If you do not have a Dogtag back end yet, deploy it as described in Deploy Dogtag.

To deploy Barbican with the Dogtag back end:
1. Open the classes/cluster/<cluster_name>/ directory of your Git project repository.
2. In infra/config.yml, add the following class:

   ```
classes:
   - system.keystone.client.service.barbican
   ```
3. In openstack/control.yml, modify the classes and parameters sections:
classes:
- system.apache.server.site.barbican
- system.galera.server.database.barbican
- system.barbican.server.cluster
- service.barbican.server.plugin.dogtag

parameters:
  _param:
  apache_barbican_api_address: ${_param:cluster_local_address}
  apache_barbican_api_host: ${_param:single_address}
  apache_barbican_ssl: ${_param:nginx_proxy_ssl}
  barbican_dogtag_nss_password: workshop
  barbican_dogtag_host: ${_param:cluster_vip_address}

  barbican:
    enabled: true
    dogtag_admin_cert:
      engine: mine
      minion: ${_param:dogtag_master_host}
    ks_notifications_enable: True
    store:
      software:
        store_plugin: dogtag_crypto
        global_default: True
    plugin:
      dogtag:
        port: ${_param:haproxy_dogtag_bind_port}

  nova:
    controller:
      barbican:
        enabled: ${_param:barbican_integration_enabled}

  cinder:
    controller:
      barbican:
        enabled: ${_param:barbican_integration_enabled}

  glance:
    server:
      barbican:
        enabled: ${_param:barbican_integration_enabled}

4. In openstack/init.yml, modify the parameters section. For example:

  parameters:
    _param:
      ...,
    barbican_service_protocol: ${_param:cluster_internal_protocol}
    barbican_service_host: ${_param:openstack_control_address}
5. In openstack/compute.yml, add the following parameters:

```yaml
parameters:
  _param:
    ...
  nova:
    compute:
      barbican:
        enabled: ${_param:barbican_integration_enabled}
    cinder:
      volume:
        barbican:
          enabled: ${_param:barbican_integration_enabled}
```

6. In openstack/proxy.yml, add the following class:

```yaml
classes:
  - system.nginx.server.proxy.openstack.barbican
```

7. Optional. In init.yml, add the following parameters if you plan to use a self-signed certificate managed by Salt:

```yaml
parameters:
  _param:
    salt:
      minion:
        trusted_ca_minions:
          - cfg01
```

8. Distribute the Dogtag KRA certificate from the Dogtag node to the Barbican nodes. Choose from the following options (engines):

- Define the KRA admin certificate manually in pillar by editing the infra/openstack/control.yml file:

```yaml
barbican:
  server:
    dogtag_admin_cert:
      engine: manual
      key: |
        <key_data>
```
• Receive the Dogtag certificate from Salt Mine. The Dogtag formula sends the KRA certificate to the dogtag_admin_cert Mine function. Add the following to infra/openstack/control.yml:

```yaml
barbican:
  server:
    dogtag_admin_cert:
      engine: mine
      minion: <dogtag_minion_node_name>
```

• If some additional steps were applied to install the KRA certificate and these steps are out of scope of the Barbican formula, the formula has the noop engine to perform no operations. If the noop engine is defined in infra/openstack/control.yml, the Barbican formula does nothing to install the KRA admin certificate.

```yaml
barbican:
  server:
    dogtag_admin_cert:
      engine: noop
```

In this case, manually populate the Dogtag KRA certificate in `/etc/barbican/kra_admin_cert.pem` on the Barbican nodes.

9. Commit and push the changes to the project Git repository.

10. Log in to the Salt Master node.

11. Update your Salt formulas at the system level:
   1. Change the directory to `/srv/salt/reclass`.
   2. Run the `git pull origin master` command.
   3. Run the `salt-call state.sls salt.master` command.

12. Apply the following state if you configured your model to use a self-signed certificate managed by Salt, as described in step 6:

   ```
salt -C '@salt:minion' state.apply salt.minion
```

13. Apply the following states:

   ```
salt -C '@keystone:client' state.apply keystone.client
salt -C '@galera:master' state.apply galera.server
salt -C '@nginx:server' state.apply nginx
salt -C '@barbican:server' state.apply barbican.server
```

14. If you have async workers enabled, restart the Barbican worker service:

   ```
salt -C '@barbican:server' service.restart barbican-worker
```
15 Restart the Barbican API server:

```
salt -C 'I@barbican:server' service.restart apache2
```

16 Verify that Barbican works correctly. For example:

```
openstack secret store --name mysecret --payload j4=]d21
```

Deploy Barbican with the simple_crypto back end

**Warning**
The deployment of Barbican with the simple_crypto back end described in this section is intended for testing and evaluation purposes only. For production deployments, use the Dogtag back end. For details, see: Deploy Dogtag.

You can configure and deploy Barbican with the simple_crypto back end.

To deploy Barbican with the simple_crypto back end:

1. Open the classes/cluster/<cluster_name>/ directory of your Git project repository.
2. In infra/config.yml, add the following classes:

   ```yaml
   classes:
   - system.keystone.client.service.barbican
   - system.mysql.client.database.barbican
   ```

3. In infra/openstack/control.yml, modify the classes and parameters sections:

   ```yaml
   classes:
   - system.galera.server.database.barbican
   - system.barbican.server.cluster
   - service.barbican.server.plugin.simple_crypto

   parameters:
   _param:
   barbican:
   server:
   store:
   software:
   crypto_plugin: simple_crypto
   store_plugin: store_crypto
   global_default: True
   ```

4. In openstack/init.yml, modify the parameters section. For example:
parameters:
  _param:
  ...  
  barbican_version: ${_param:openstack_version}
  barbican_service_host: ${_param:openstack_control_address}
  mysql_barbican_password: password123
  keystone_barbican_password: password123
  barbican_simple_crypto_kek: “base64 encoded 32 bytes as secret key”

5. In openstack/proxy.yml, add the following class:

    classes:
    - system.nginx.server.proxy.openstack.barbican

6. Optional. In init.yml, add the following parameters if you plan to use a self-signed certificate managed by Salt:

    parameters:
      _param:
      salt:
        minion:
          trusted_ca_minions:
            - cfg01

7. Commit and push the changes to the project Git repository.

8. Log in to the Salt Master node.

9. Update your Salt formulas at the system level:

   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.

10 Apply the following state if you configured your model to use a self-signed certificate managed by Salt, as described in step 6:

    salt -C 'I@salt:minion' state.apply salt.minion

11 Apply the following states:

    salt -C 'I@keystone:client' state.apply keystone.client
    salt -C 'I@galera:master' state.apply galera.server
    salt -C 'I@nginx:server' state.apply nginx
    salt -C 'I@haproxy:proxy' state.apply haproxy.proxy
    salt -C 'I@barbican:server' state.apply barbican.server
Deploy Ironic

While virtualization provides outstanding benefits in server management, cost efficiency, and resource consolidation, some cloud environments with particularly high I/O rate may require physical servers as opposed to virtual.

MCP supports bare-metal provisioning for OpenStack environments using the OpenStack Bare Metal service (Ironic). Ironic enables system administrators to provision physical machines in the same fashion as they provision virtual machines.

By default, MCP does not deploy Ironic, therefore, to use this functionality, you need to make changes to your Reclass model manually prior to deploying an OpenStack environment.

Limitations

When you plan on using the OpenStack Bare Metal provisioning service (Ironic), consider the following limitations:

**Network segmentation for bare-metal nodes is not supported.**

The OpenStack Bare Metal provisioning service is deployed with flat network, where each bare-metal node can access one another. Due to security implications of such setup, using the OpenStack Bare Metal service with MCP is limited to single tenant or trusted tenants deployments.

**Only iSCSI deploy drivers are enabled.**

The OpenStack Bare Metal service is deployed with only iSCSI deploy drivers enabled which may pose performance limitations for deploying multiple nodes concurrently. You can enable agent-based Ironic drivers manually after deployment if the deployed cloud has a working Swift-compatible object-store service with support for temporary URLs, with OpenStack Image service (Glance) configured to use the object store service to store images. For more information on how to configure the OpenStack Image service for temporary URLs, see OpenStack documentation.

**OpenContrail networking is not supported.**

Although it may be possible to enable the OpenStack Bare Metal service with OpenContrail for a static, flat network provisioning use case, such configuration has not been tested by Mirantis and therefore, not supported.

Modify the deployment model

To use the OpenStack Bare Metal service, you need to modify your Reclass model before deploying a new OpenStack environment. You can also deploy the OpenStack Bare Metal service in the existing OpenStack environment by updating the Salt states.

As bare-metal configurations vary, this section provides examples of deployment model modifications. You may need to tailor them for your specific use case. The examples describe:

See also

- Integrate Barbican to OpenContrail LBaaSv2
- Barbican OpenStack documentation
• OpenStack Bare Metal API service running on the OpenStack Controller node
• A single-node Bare Metal service for ironic-conductor and other services per the baremetal role residing on the bmt01 node

To modify the deployment model:

1. Create a deployment model as described in Create a deployment metadata model using the Model Designer UI.
2. In the top Reclass ./init.yml file, add:

```yaml
parameters:
  _param:
    openstack_baremetal_node01_address: 172.16.10.110
    openstack_baremetal_address: 192.168.90.10
    openstack_baremetal_node01_baremetal_address: 192.168.90.11
    openstack_baremetal_neutron_subnet_cidr: 192.168.90.0/24
    openstack_baremetal_neutron_subnet_allocation_start: 192.168.90.100
    openstack_baremetal_neutron_subnet_allocation_end: 192.168.90.150
    openstack_baremetal_node01_hostname: bmt01
```

**Note**

The openstack_baremetal_neutron_subnet_ parameters must match your baremetal network settings. The baremetal nodes must connected to the network before the deployment. During the deployment, MCP automatically registers this network in the OpenStack Networking service.

3. Modify the ./infra/config.yml:

```yaml
classes:
- system.salt.master.formula.pkg.baremetal
- system.keystone.client.service.ironic
- system.reclass.storage.system.openstack_baremetal_single

parameters:
reclass:
  storage:
    class_mapping:
      expression: <<node_hostname>>__startswith__bmt
      node_class:
        value_template:
          - cluster.<<node_cluster>>.openstack.baremetal
    cluster_param:
      openstack_baremetal_node01_address:
        value_template: <<node_control_ip>>
```

©2020, Mirantis Inc.
node:
  openstack_baremetal_node01:
    params:
      single_baremetal_address: ${_param:openstack_baremetal_node01_baremetal_address}
      keepalived_openstack_baremetal_vip.priority: 100
      ironic_api_type: 'deploy'
      tenant_address: 10.1.0.110
      external_address: 10.16.0.110

4. Modify the OpenStack nodes:

• ./openstack/init.yml:

  parameters:
    _param:
      ironic_version: ${_param:openstack_version}
      ironic_api_type: 'public'
      ironic_service_host: ${_param:cluster_vip_address}
      cluster_baremetal_local_address: ${_param:cluster_local_address}
      mysql_baremetal_password: workshop
      keystone_ironic_password: workshop

  linux:
    network:
      host:
        bmt01:
          address: ${_param:openstack_baremetal_node01_address}
          names:
            - bmt01
            - bmt01.$(_param:cluster_domain)

• ./openstack/control.yml:

  classes:
  - system.haproxy.proxy.listen.openstack.ironic
  - system.galera.server.database.ironic
  - service.ironic.client
  - system.ironic.api.cluster
  - cluster.virtual-mcp11-ovs-ironic

• ./openstack/baremetal.yml:

  classes:
  - system.linux.system.repo.mcp.openstack
  - system.linux.system.repo.mcp.extra
  - system.linux.system.repo.saltstack.xenial
  - system.ironic.api.cluster # deploy only api (heartbeat and lookup endpoints are open)
  - system.ironic.conductor.cluster
  - system.ironic.tftpd_hpa
  - system.nova.compute_ironic.cluster
- system.apache.server.single
- system.apache.server.site.ironic
- system.keystone.client.core
- system.neutron.client.service.ironic
- cluster.virtual-mcp11-ovs-ironic

**parameters:**

```yaml
  _param:
    primary_interface: ens4
    baremetal_interface: ens5
    linux_system_codename: xenial
    interface_mtu: 1450
    cluster_vip_address: ${_param:openstack_control_address}
    cluster_baremetal_vip_address: ${_param:single_baremetal_address}
    cluster_baremetal_local_address: ${_param:single_baremetal_address}
    linux_system_codename: xenial

linux:
  network:
    concat_iface_files:
      - src: '/etc/network/interfaces.d/50-cloud-init.cfg'
        dst: '/etc/network/interfaces'
    bridge: openvswitch
  interface:
    dhcp_int:
      enabled: true
      name: ens3
      proto: dhcp
      type: eth
      mtu: ${_param:interface_mtu}
    primary_interface:
      enabled: true
      name: ${_param:primary_interface}
      proto: static
      address: ${_param:single_address}
      netmask: 255.255.255.0
      mtu: ${_param:interface_mtu}
      type: eth
    baremetal_interface:
      enabled: true
      name: ${_param:baremetal_interface}
      mtu: ${_param:interface_mtu}
      proto: static
      address: ${_param:cluster_baremetal_local_address}
      netmask: 255.255.255.0
      type: eth
      mtu: ${_param:interface_mtu}
```

5. Proceed to Install the Bare Metal service components.

Install the Bare Metal service components
After you have configured the deployment model as described in Modify the deployment model, install the Bare Metal service components, including Ironic API, Ironic Conductor, Ironic Client, and others. Use the procedure below for both new or existing clusters.

To install the Bare Metal service components:

1. Install Ironic API:
   
   ```
   salt -C 'I@ironic:api' state.sls ironic.api -b 1
   ```

2. Install Ironic Conductor:
   
   ```
   salt -C 'I@ironic:conductor' state.sls ironic.conductor
   ```

3. Install Ironic Client:
   
   ```
   salt -C 'I@ironic:client' state.sls ironic.client
   ```

4. Install software required by Ironic, such as Apache and TFTP server:
   
   ```
   salt -C 'I@ironic:conductor' state.sls apache
   salt -C 'I@tftpd_hpa:server' state.sls tftpd_hpa
   ```

5. Install nova-compute with ironic virt-driver:
   
   ```
   salt -C 'I@nova:compute' state.sls nova.compute
   salt -C 'I@nova:compute' cmd.run 'systemctl restart nova-compute'
   ```

6. Log in to an OpenStack Controller node.

7. Verify that the Ironic services are enabled and running:
   
   ```
   salt -C 'I@ironic:client' cmd.run 'source keystonec; ironic driver-list'
   ```

Deploy Manila

Manila, also known as the OpenStack Shared File Systems service, provides coordinated access to shared or distributed file systems that a compute instance can consume.

By default, MCP does not deploy Manila. Therefore, to use this functionality, you need to make changes to your Reclass model manually prior to deploying an OpenStack environment.

Modify the deployment model

To use the Manila service, you need to modify your Reclass model before deploying a new OpenStack environment. You can also deploy Manila in an existing OpenStack environment by updating the Salt states.

The manila-share service may use different back ends. This section provides examples of deployment model modifications for the LVM back end. You may need to tailor these examples
depending on the needs of your deployment. Basically, the examples provided in this section describe the following configuration:

- OpenStack Manila API and Scheduler services run on an OpenStack controller node
- The manila-share service and other services per share role reside on the share01 node

To modify the deployment model:

1. Create a deployment model as described in Create a deployment metadata model using the Model Designer UI.
2. Modify the ./infra/config.yml file:

```yaml
classes:
- system.salt.master.formula.pkg.manila
- system.keystone.client.service.manila
- system.reclass.storage.system.openstack_share_single

parameters:
  reclass:
    storage:
      class_mapping:
        expression: <node_hostname>__startswith__share
      node_class:
        value_template:
          - cluster.<node_cluster>.openstack.share
      cluster_param:
        openstack_share_node01_address:
          value_template: <node_control_ip>
  node:
    openstack_share_node01:
      params:
        single_share_address: ${_param:openstack_share_node01_share_address}
        tenant_address: 10.1.0.110

3. Configure the OpenStack nodes:

- In ./openstack/init.yml, add the following configuration:

```yaml
parameters:
  _param:
    manila_version: ${_param:openstack_version}
    manila_service_host: ${_param:cluster_vip_address}
    mysql_manila_password: workshop
    keystone_manila_password: workshop
  linux:
    network:
      host:
        share01:
          address: ${_param:openstack_share_node01_address}
          names:
```
- share01
- share01.{$_param:cluster_domain}

• In ./openstack/control.yml, define the following classes:

```
classes:
- system.haproxy.proxy.listen.openstack.manila
- system.galera.server.database.manila
- system.apache.server.site.manila
- system.manila.common.cluster
```

• In ./openstack/share.yml, add the following configuration:

```
classes:
- system.linux.system.repo.mcp.openstack
- system.linux.system.repo.mcp.extra
- system.linux.system.repo.saltstack.xenial
- system.manila.api.cluster
- system.manila.common.cluster
- system.manila.share.backend.lvm
- cluster.virtual-mcp11-ovs-manila

parameters:
  _param:
    primary_interface: ens4
    share_interface: ens5
    linux_system_codename: xenial
    interface_mtu: 1450
    cluster_vip_address: ${_param:openstack_control_address}
    linux_system_codename: xenial

linux:
  network:
    concat_iface_files:
      - src: '/etc/network/interfaces.d/50-cloud-init.cfg'
      dst: '/etc/network/interfaces'
    bridge: openvswitch

interface:
  dhcp_int:
    enabled: true
    name: ens3
    proto: dhcp
    type: eth
    mtu: ${_param:interface_mtu}
  primary_interface:
    enabled: true
    name: ${_param:primary_interface}
    proto: static
    address: ${_param:single_address}
    netmask: 255.255.255.0
    mtu: ${_param:interface_mtu}
4. Proceed to Install the Manila components.

Install the Manila components

After you have configured the deployment model as described in Modify the deployment model, install the Manila components that include the manila-api, manila-scheduler, manila-share, manila-data, and other services. Use the procedure below for both new or existing clusters.

To install the Manila components:

1. Install manila-api:
   
   ```
   salt -C 'I@manila:api' state.sls manila.api -b 1
   ```

2. Install manila-scheduler:
   
   ```
   salt -C 'I@manila:scheduler' state.sls manila.scheduler
   ```

3. Install manila-share:
   
   ```
   salt -C 'I@manila:share' state.sls manila.share
   ```

4. Install manila-data:
   
   ```
   salt -C 'I@manila:data' state.sls manila.data
   ```

5. Install the Manila client:
   
   ```
   salt -C 'I@manila:client' state.sls manila.client
   ```

6. Log in to any OpenStack controller node.

7. Verify that the Manila services are enabled and running:
   
   ```
   salt 'cfg01*' cmd.run 'source keystonercv3; manila list'
   salt 'cfg01*' cmd.run 'source keystonercv3; manila service-list'
   ```

Deploy a Ceph cluster manually
Ceph is a storage back end for cloud environments. This section guides you through the manual deployment of a Ceph cluster.

**Warning**
Converged storage is not supported.

**Note**
Prior to deploying a Ceph cluster:

1. Verify that you have selected Ceph enabled while generating a deployment model as described in Define the deployment model.
2. Verify that OpenStack services, such as Cinder, Glance, and Nova are up and running.
3. Verify and, if required, adjust the Ceph setup for disks in the classes/cluster/<CLUSTER_NAME>/ceph/osd.yml file.

To deploy a Ceph cluster:

1. Log in to the Salt Master node.
2. Update modules and states on all Minions:

   ```
salt '*' saltutil.sync_all
   ```

3. Run basic states on all Ceph nodes:

   ```
salt '*' state.sls linux,openssh,salt,ntp,rsyslog
   ```

4. Generate admin and mon keyrings:

   ```
salt -C '@ceph:mon:keyring:mon or @ceph:common:keyring:admin' state.sls ceph.mon
   salt -C '@ceph:mon' saltutil.sync_grains
   salt -C '@ceph:mon:keyring:mon or @ceph:common:keyring:admin' mine.update
   ```
5. Deploy Ceph mon nodes:
   - If your Ceph version is older than Luminous:
     ```bash
     salt -C 'I@ceph:mon' state.sls ceph.mon
     ```
   - If your Ceph version is Luminous or newer:
     ```bash
     salt -C 'I@ceph:mon' state.sls ceph.mon
     salt -C 'I@ceph:mgr' state.sls ceph.mgr
     ```

6. To prepare the Ceph CRUSH map model, uncomment the example pillar in the classes/cluster/<CLUSTER_NAME>/ceph/setup.yml file and modify it as required.

7. Deploy Ceph osd nodes:
   ```bash
   salt -C 'I@ceph:osd' state.sls ceph.osd
   salt -C 'I@ceph:osd' saltutil.sync_grains
   salt -C 'I@ceph:osd' state.sls ceph.osd.custom
   salt -C 'I@ceph:osd' saltutil.sync_grains
   salt -C 'I@ceph:osd' mine.update
   salt -C 'I@ceph:setup' state.sls ceph.setup
   ```

8. Deploy RADOS Gateway:
   ```bash
   salt -C 'I@ceph:radosgw' saltutil.sync_grains
   salt -C 'I@ceph:radosgw' state.sls ceph.radosgw
   ```

9. Set up the Keystone service and endpoints for Swift or S3:
   ```bash
   salt -C 'I@keystone:client' state.sls keystone.client
   ```

10. Connect Ceph to your MCP cluster:
    ```bash
        salt -C 'I@ceph:common and I@glance:server' state.sls ceph.common,ceph.setup.keyring,glance
        salt -C 'I@ceph:common and I@glance:server' service.restart glance-api
        salt -C 'I@ceph:common and I@glance:server' service.restart glance-glare
        salt -C 'I@ceph:common and I@glance:server' service.restart glance-registry
        salt -C 'I@ceph:common and I@cinder:controller' state.sls ceph.common,ceph.setup.keyring,cinder
        salt -C 'I@ceph:common and I@nova:compute' state.sls ceph.common,ceph.setup.keyring
        salt -C 'I@ceph:common and I@nova:compute' saltutil.sync_grains
        salt -C 'I@ceph:common and I@nova:compute' state.sls nova
    ```

11. Modify and apply the generated CRUSH map:
    1. View the CRUSH map generated in the /etc/ceph/crushmap file and modify it as required. Before applying the CRUSH map, verify that the settings are correct.
    2. Apply the following state:
3. Once the CRUSH map is set up correctly, add the following snippet to the classes/cluster/<CLUSTER_NAME>/ceph/osd.yml file to make the settings persist even after a Ceph OSD reboots:

```yaml
ceph:
  osd:
    crush_update: false
```

4. Apply the following state:

```
salt -C 'I@ceph:osd' state.sls ceph.osd
```

Once done, if your Ceph version is Luminous or newer, you can access the Ceph dashboard through http://<active_mgr_node_IP>:7000/. Run ceph -s on a cmn node to obtain the active mgr node.

**Deploy Xtrabackup for MySQL**

MCP uses the Xtrabackup utility to back up MySQL databases.

To deploy Xtrabackup for MySQL:

1. Apply the xtrabackup server state:

```
salt -C 'I@xtrabackup:server' state.sls xtrabackup
```

2. Apply the xtrabackup client state:

```
salt -C 'I@xtrabackup:client' state.sls openssh.client,xtrabackup
```

**Post-deployment procedures**

After your OpenStack environment deployment has been successfully completed, perform a number of steps to verify all the components are working and your OpenStack installation is stable and performs correctly at scale.

**Run non-destructive Rally tests**

Rally is a benchmarking tool that enables you to test the performance and stability of your OpenStack environment at scale.

The Tempest and Rally tests are integrated into the MCP CI/CD pipeline and can be managed through the DriveTrain web UI.

For debugging purposes, you can manually start Rally tests from the deployed Benchmark Rally Server (bmk01) with the installed Rally benchmark service or run the appropriate Docker container.

To manually run a Rally test on a deployed environment:
1. Validate the input parameters of the Rally scenarios in the `task_arguments.yaml` file.

2. Create the Cirros image:

```
Note
If you need to run Glance scenarios with an image that is stored locally, download it from https://download.cirros-cloud.net/0.3.5/cirros-0.3.5-i386-disk.img:

wget https://download.cirros-cloud.net/0.3.5/cirros-0.3.5-i386-disk.img
```

```
openstack image create --disk-format qcow2 --container-format bare --public --file ./cirros-0.3.5-i386-disk.img cirros
```

3. Run the Rally scenarios:

```
rally task start <name_of_file_with_scenarios> --task-args-file task_arguments.yaml
```

or

```
rally task start combined_scenario.yaml --task-args-file task_arguments.yaml
```

Troubleshoot

This section provides solutions to the issues that may occur while installing Mirantis Cloud Platform.

Troubleshooting of an MCP installation usually requires the salt command usage. The following options may be helpful if you run into an error:

- `-l LOG_LEVEL, --log-level=LOG_LEVEL`
  Console logging log level. One of all, garbage, trace, debug, info, warning, error, or quiet. Default is warning

- `--state-output=STATE_OUTPUT`
  Override the configured STATE_OUTPUT value for minion output. One of full, terse, mixed, changes, or filter. Default is full.

To synchronize all of the dynamic modules from the file server for a specific environment, use the saltutil.sync_all module. For example:

```
salt '*' saltutil.sync_all
```

Troubleshooting the server provisioning

This section includes the workarounds for the following issues:
Virtual machine node stops responding
If one of the control plane VM nodes stops responding, you may need to redeploy it.

Workaround:

1. From the physical node where the target VM is located, get a list of the VM domain IDs and VM names:

   
   virsh list

2. Destroy the target VM (ungraceful powering off of the VM):

   
   virsh destroy DOMAIN_ID

3. Undefine the VM (removes the VM configuration from KVM):

   
   virsh undefine VM_NAME

4. Verify that your physical KVM node has the correct salt-common and salt-minion version:

   
   apt-cache policy salt-common
   apt-cache policy salt-minion

   **Note**
   If the salt-common and salt-minion versions are not 2015.8, proceed with Install the correct versions of salt-common and salt-minion.

5. Redeploy the VM from the physical node meant to host the VM:

   
   salt-call state.sls salt.control

6. Verify the newly deployed VM is listed in the Salt keys:

   
   salt-key

7. Deploy the Salt states to the node:

   
   salt 'OST_NAME*' state.sls linux,ntp,openssh,salt

8. Deploy service states to the node:

   
   salt 'HOST_NAME*' state.sls keepalived,haproxy,SPECIFIC_SERVICES
Note
You may need to log in to the node itself and run the states locally for higher success rates.

Troubleshoot Ceph
This section includes workarounds for the Ceph-related issues that may occur during the deployment of a Ceph cluster.

Troubleshoot an encrypted Ceph OSD
During the deployment of a Ceph cluster, an encrypted OSD may fail to be prepared or activated and thus fail to join the Ceph cluster. In such case, remove all the disk partitions as described below.

Workaround:
1. From the Ceph OSD node where the failed encrypted OSD disk resides, erase its partition table:

   ```
   dd if=/dev/zero of=/dev/<<ADD>> bs=512 count=1 conv=notrunc
   ```

2. Reboot the server:

   ```
   reboot
   ```

3. Run the following command twice to create a partition table for the disk and to remove the disk data:

   ```
   ceph-disk zap /dev/<<ADD>>;
   ```

4. Remove all disk signatures using `wipefs`:

   ```
   wipefs --all --force /dev/<<ADD>>*;
   ```

**Deploy a Kubernetes cluster manually**

Kubernetes is the system for containerized applications automated deployment, scaling, and management. This section guides you through the manual deployment of a Kubernetes cluster on either bare metal or a cloud provider, such as AWS or OpenStack. For an easier deployment process, use the automated DriveTrain deployment procedure described in Deploy a Kubernetes cluster.

**Prerequisites**

The following are the prerequisite steps for a manual MCP Kubernetes deployment:
1. Prepare six nodes:
   - 1 x configuration node - a host for the Salt Master node. Can be a virtual machine.
   - 3 x Kubernetes Master nodes (ctl) - hosts for the Kubernetes control plane components and etcd.
   - 2 x Kubernetes Nodes (cmp) - hosts for the Kubernetes pods, groups of containers that are deployed together on the same host.
2. For an easier deployment and testing, the following usage of three NICs is recommended:
   - 1 x NIC as a PXE/DHCP/Salt network (PXE and DHCP are third-party services in a data center, unmanaged by SaltStack)
   - 2 x NICs as bond active-passive or active-active with two 10 Gbit slave interfaces
3. If you are going to deploy Kubernetes on top of an AWS or OpenStack environment, deploy a corresponding AWS or OpenStack environment that you will use for a Kubernetes cluster deployment.
4. Create a project repository.
5. Create a deployment metadata model. Optionally, Enable Virtlet.
6. If you have swap enabled on the ctl and cmp nodes, modify the deployment model as described in Add swap configuration to a Kubernetes deployment model.
7. Define interfaces.
8. Install a base infrastructure:
   - 1. Get the virtual machines images.
   - 2. Prepare images for the Foundation node.
   - 3. Install the Salt Master node.
Now, proceed to Deploy a Kubernetes cluster.

Salt formulas used in the Kubernetes cluster deployment

MCP Kubernetes cluster standard deployment uses the following Salt formulas to deploy and configure a Kubernetes cluster:

- **salt-formula-kubernetes**
  - Handles Kubernetes hyperkube binaries, CNI plugins, Calico manifests
- **salt-formula-etcd**
  - Provisions etcd clusters
- **salt-formula-docker**
  - Installs and configures the Docker daemon
- **salt-formula-bird**
  - Customizes BIRD templates used by Calico to provide advanced networking scenarios for route distribution through BGP

Add swap configuration to a Kubernetes deployment model
If you have swap enabled on the ctl and cmp nodes, configure your Kubernetes model to make kubelet work correctly with swapping.

To add swap configuration to a Kubernetes deployment model:

1. Open your Git project repository.
2. In classes/cluster/<cluster-name>/kubernetes/control.yml, add the following snippet:

   ```yaml
   ...
   parameters:
     kubernetes:
       master:
         kubelet:
           fail_on_swap: False
   ...
   ```

3. In classes/cluster/<cluster-name>/kubernetes/compute.yml, add the following snippet:

   ```yaml
   ...
   parameters:
     kubernetes:
       pool:
         kubelet:
           fail_on_swap: False
   ...
   ```

Now, proceed with further MCP Kubernetes cluster configuration as required.

Define interfaces

Since Cookiecutter is simply a tool to generate projects from templates, it cannot handle all networking use-cases. Your cluster may include a single interface, two interfaces in bond, bond and management interfaces, and so on.

This section explains how to handle 3 interfaces configuration:

- eth0 interface for pxe
- eth1 and eth2 as bond0 slave interfaces

To configure network interfaces:

1. Open your MCP Git project repository.
2. Open the `{cookiecutter.cluster_name}`/kubernetes/init.yml file for editing.
3. Add the following example definition to this file:

   ```yaml
   parameters:
   ...
   _param:
     deploy_nic: eth0
     primary_first_nic: eth1
     primary_second_nic: eth2
   ```
deploy_nic:
  name: ${_param:deploy_nic}
  enabled: true
  type: eth
  proto: static
  address: ${_param:deploy_address}
  netmask: 255.255.255.0
primary_first_nic:
  name: ${_param:primary_first_nic}
  enabled: true
  type: slave
  master: bond0
  mtu: 9000
  pre_up_cmds:
  - /sbin/ethtool --offload eth6 rx off tx off tso off gro off
primary_second_nic:
  name: ${_param:primary_second_nic}
  type: slave
  master: bond0
  mtu: 9000
  pre_up_cmds:
  - /sbin/ethtool --offload eth7 rx off tx off tso off gro off
bond0:
  enabled: true
  proto: static
  type: bond
  use_interfaces:
  - ${_param:primary_first_nic}
  - ${_param:primary_second_nic}
  slaves: ${_param:primary_first_nic} ${_param:primary_second_nic}
  mode: active-backup
  mtu: 9000
  address: ${_param:single_address}
  netmask: 255.255.255.0
  name_servers:
  - {{ cookiecutter.dns_server01 }}
  - {{ cookiecutter.dns_server02 }}

Deploy a Kubernetes cluster

After you complete the prerequisite steps described in Prerequisites, deploy your MCP Kubernetes cluster manually using the procedure below.

To deploy the Kubernetes cluster:

1. Log in to the Salt Master node.
2. Update modules and states on all Minions:

```
salt '*' saltutil.sync_all
```

3. Register all discovered compute nodes. Run the following command on every compute node:

```py
salt-call event.send "reclass/minion/classify" \
"{"node_master_ip": "<config_host>"}, \
"node_os": "<os_codename>"}, \
"node_deploy_ip": "<node_deploy_network_ip>"}, \
"node_deploy_iface": "<node_deploy_network_iface>"}, \
"node_control_ip": "<node_control_network_ip>"}, \
"node_control_iface": "<node_control_network_iface>"}, \
"node_tenant_ip": "<node_tenant_network_ip>"}, \
"node_tenant_iface": "<node_tenant_network_iface>"}, \
"node_external_ip": "<node_external_network_ip>"}, \
"node_external_iface": "<node_external_network_iface>"}, \
"node_baremetal_ip": "<node_baremetal_network_ip>"}, \
"node_baremetal_iface": "<node_baremetal_network_iface>"}, \
"node_domain": "<node_domain>"}, \
"node_cluster": "<cluster_name>"}, \
"node_hostname": "<node_hostname>"}"
```

Modify the parameters passed with the command above as required. The table below provides the description of the parameters required for a compute node registration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_host</td>
<td>IP of the Salt Master node</td>
</tr>
<tr>
<td>os_codename</td>
<td>Operating system code name. Check the system response of lsb_release -c for it</td>
</tr>
<tr>
<td>node_deploy_network_ip</td>
<td>Minion deploy network IP address</td>
</tr>
<tr>
<td>node_deploy_network_iface</td>
<td>Minion deploy network interface</td>
</tr>
<tr>
<td>node_control_network_ip</td>
<td>Minion control network IP address</td>
</tr>
<tr>
<td>node_control_network_iface</td>
<td>Minion control network interface</td>
</tr>
<tr>
<td>node_tenant_network_ip</td>
<td>Minion tenant network IP address</td>
</tr>
<tr>
<td>node_tenant_network_iface</td>
<td>Minion tenant network interface</td>
</tr>
<tr>
<td>node_external_network_ip</td>
<td>Minion external network IP address</td>
</tr>
<tr>
<td>node_external_network_iface</td>
<td>Minion external network interface</td>
</tr>
</tbody>
</table>
4. Log in to the Salt Master node.

5. Create and distribute SSL certificates for services using the salt state:

   salt "*" state.sls salt

6. Perform Linux system configuration to synchronize repositories and execute outstanding system maintenance tasks:

   salt -C '@docker:host' state.sls linux.system

7. Install Keepalived:

   salt -C '@keepalived:cluster' state.sls keepalived -b 1

8. Install HAProxy:

   salt -C '@haproxy:proxy' state.sls haproxy
   salt -C '@haproxy:proxy' service.status haproxy

9. Install Docker:

   salt -C '@docker:host' state.sls docker.host
   salt -C '@docker:host' cmd.run "docker ps"

10. Install etcd:

    salt -C '@etcd:server' state.sls etcd.server.service
    salt -C '@etcd:server' cmd.run "etcdctl cluster-health"

    Install etcd with the SSL support:

    salt -C '@etcd:server' state.sls salt.minion.cert,etcd.server.service
    salt -C '@etcd:server' cmd.run '. /var/lib/etcd/configenv && etcdctl cluster-health'
11 Install Kubernetes:

```
salt -C 'I@kubernetes:master' state.sls kubernetes.master.kube-addons
salt -C 'I@kubernetes:pool' state.sls kubernetes.pool
```

12 For the Calico setup:

1. Verify the Calico nodes status:

```
salt -C 'I@kubernetes:pool' cmd.run "calicoctl node status"
```

2. Set up NAT for Calico:

```
salt -C 'I@kubernetes:master' state.sls etcd.server.setup
```

13 For the OpenContrail setup, deploy OpenContrail as described in Deploy OpenContrail manually.

14 Run master to check consistency:

```
salt -C 'I@kubernetes:master' state.sls kubernetes exclude=kubernetes.master.setup
```

15 Register addons:

```
salt -C 'I@kubernetes:master' --subset 1 state.sls kubernetes.master.setup
```

16 Restart kubelet:

```
salt -C 'I@kubernetes:pool' service.restart kubelet
```

17 Log in to any Kubernetes ctl node to verify that all nodes have been registered successfully:

```
kubectl get nodes
```

After you deploy Kubernetes, deploy StackLight LMA to your cluster as described in Deploy StackLight LMA components.

**Enable Virtlet**

You can enable Kubernetes to run virtual machines using Virtlet. Virtlet enables you to run unmodified QEMU/KVM virtual machines that do not include an additional Docker layer as in similar solutions in Kubernetes.

Virtlet requires the --feature-gates=MountPropagation=true feature gate to be enabled in the Kubernetes API server and on all kubelet instances. This feature is enabled by default in MCP. Using this feature, Virtlet can create or delete network namespaces assigned to VM pods.

**Deploy Virtlet**

You can deploy Virtlet on either new or existing MCP cluster using the procedure below.
To deploy Virtlet on a new MCP cluster:

1. When generating a deployment metadata model using the ModelDesigner UI, select the Virtlet enabled check box in the Kubernetes Product parameters section.
2. Open your Git project repository.
3. In `classes/cluster/<cluster-name>/kubernetes/compute.yml`, modify the `kubernetes:common:addons:virtlet:` parameters as required to define the Virtlet namespace and image path as well as the number of compute nodes on which you want to enable Virtlet. For example:

   ```yaml
   parameters:
   kubernetes:
     common:
       addons:
         virtlet:
           enabled: true
           namespace: kube-system
           image: mirantis/virtlet:latest
           hosts:
           - cmp0
           - cmp1
   ```

4. If your networking system is OpenContrail, add the following snippet to `classes/cluster/<cluster-name>/opencontrail/compute.yml`:

   ```yaml
   kubernetes:
     pool:
       network:
         hash: 77169cdadb80a5e33e9d9fe093ed0d99
   ```

Proceed with further MCP cluster configuration. Virtlet will be automatically deployed during the Kubernetes cluster deployment.

To deploy Virtlet on an existing MCP cluster:

1. Open your Git project repository.
2. In `classes/cluster/<cluster-name>/kubernetes/compute.yml`, add the following snippet:

   ```yaml
   parameters:
   kubernetes:
     common:
       addons:
         virtlet:
           enabled: true
           namespace: kube-system
           image: mirantis/virtlet:latest
           hosts:
           - cmp0
           - cmp1
   ```
Modify the kubernetes:common:addons:virtlet: parameters as required to define the Virtlet namespace and image path as well as the number of compute nodes on which you want to enable Virtlet.

3. If your networking system is OpenContrail, add the following snippet to classes/cluster/<cluster-name>/opencontrail/compute.yml:

   ```yaml
   kubernetes:
     pool:
       network:
         hash: 77169cdadb80a5e33e9d9fe093ed0d99
   ```

4. Commit and push the changes to the project Git repository.

5. Log in to the Salt Master node.

6. Update your Salt formulas and the system level of your repository:
   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.
   4. Run the salt-call state.sls reclass command.

7. Apply the following states:

   ```
   salt -C '@kubernetes:master' state.sls kubernetes.master.kube-addons
   salt -C '@kubernetes:pool' state.sls kubernetes.pool
   salt -C '@kubernetes:master and *01*' state.sls kubernetes.master.setup
   ```

See also

Verify Virtlet after deployment

Verify Virtlet after deployment

After you enable Virtlet as described in Deploy Virtlet, proceed with the verification procedure described in this section.

To verify Virtlet after deployment:

1. Verify a basic pod startup:
   1. Start a sample VM:

      ```
      kubectl create -f https://raw.githubusercontent.com/Mirantis/virtlet/v0.9.2/examples/cirros-vm.yaml
      kubectl get pods --all-namespaces -o wide -w
      ```

   2. Connect to the VM console:
kubectl attach -it cirros-vm

If you do not see a command prompt, press Enter.

Example of system response:

```
login as 'cirros' user. default password: 'gosubsgo'. use 'sudo' for root.
cirros-vm login: cirros
Password:
```

$ To quit the console, use the ^] key combination.

2. Verify SSH access to the VM pod:

   1. Download the vmssh.sh script with the test SSH key:

   ```
   wget https://raw.githubusercontent.com/Mirantis/virtlet/v0.9.2/examples/{vmssh.sh,vmkey}
   chmod +x vmssh.sh
   chmod 600 vmkey
   ```

   **Note**
   The vmssh.sh script requires kubectl to access a cluster.

   2. Access the VM pod using the vmssh.sh script:

   ```
   ./vmssh.sh cirros@cirros-vm
   ```

   3. Verify whether the VM can access the Kubernetes cluster services:

   1. Verify the DNS resolution of the cluster services:

   ```
   nslookup kubernetes.default.svc.cluster.local
   ```

   2. Verify the service connectivity:

   ```
   curl -k https://kubernetes.default.svc.cluster.local
   ```
Note
The above command will raise an authentication error. Ignore this error.

3. Verify Internet access from the VM. For example:

```
curl -k https://google.com
ping -c 1 8.8.8.8
```

**Deploy OpenContrail manually**

OpenContrail is a component of MCP that provides overlay networking built on top of physical IP-based underlay network for cloud environments. OpenContrail provides more flexibility in terms of network hardware used in cloud environments comparing to other enterprise-class networking solutions.

**Limitations**

The OpenContrail deployment in MCP includes the following limitations:

1. OpenContrail does not support tenant renaming due to architecture limitations.
2. Incorrect route redistribution for the floating IP (FIP) and loss of the traffic from the defined load balancer may occur on highly scaled OpenStack-based environments with OpenContrail 3.2. For details, see MCP Operations Guide: Troubleshoot OpenContrail.
3. The Kubernetes-specific OpenContrail limitations:
   - A Kubernetes container is reachable only from a node where this container is spawned. The container provides support for liveness and readiness probes. But any feature that has a requirement for all nodes to be able to communicate with all containers and vice versa is not supported. For example, proxying requests to pods from an API server. Mirantis recommends avoiding isolation between nodes and pods by configuring routing between them. It can be done by using a simple gateway or a hardware router. For better performance, Mirantis recommends using hardware routers supported by OpenContrail.
   - In the current OpenContrail integration, load balancing is implemented with equal cost multi-path (ECMP). Therefore, the port mapping of a Kubernetes Service is not supported. For that reason, Mirantis recommends using the same ports as in Kubernetes pods for the Kubernetes services.
   - Exposing a service with the NodePort type is not supported. Use the LoadBalancer type with the OpenContrail public network instead. For details, see the official Kubernetes documentation.
Requirements for federated Kubernetes

The OpenContrail requirements when deployed with federated Kubernetes on the MCP cluster are as follows:

• LoadBalancer must be configured to work with the port mapping of a Kubernetes Service.
• The Kubernetes Service network must be configured for all namespaces. The federation control plane must be configured in a new namespace.
• The floating IP must be reachable inside the Kubernetes cluster or kubefed must be run outside the cluster.

Deploy OpenContrail

This section instructs you how to manually deploy OpenContrail on your Mirantis Cloud Platform (MCP) cluster.

To deploy OpenContrail:

1. Log in to the Salt Master node.
2. Run the following basic states to prepare the OpenContrail nodes:

        salt -C 'ntw* or nal*' saltutil.refresh_pillar
        salt -C '@opencontrail:database' saltutil.sync_all
        salt -C '@opencontrail:database' state.sls salt.minion,linux,ntp,openssh

3. Deploy and configure Keepalived and HAProxy:

        salt -C '@opencontrail:database' state.sls keepalived,haproxy

4. Log in to any OpenContrail network node.
5. Deploy OpenContrail:

        salt-call state.sls opencontrail exclude=opencontrail.client
6. Deploy OpenContrail to the remaining network nodes individually:

```
salt-call state.sls opencontrail exclude=opencontrail.client
```

7. Verify the status of the OpenContrail service from one of the OpenContrail network nodes:

```
contrail-status
```

In the output, the services status should be active or backup.

```
Note
It may take some time for all services to finish initializing.
```

If you have issues during the OpenContrail deployment for an OpenStack environment, see *MCP Operations Guide: Troubleshoot OpenContrail*.

8. Deploy OpenContrail to the analytics nodes:

1. Log in to the Salt Master node.

2. Apply the opencontrail state to the first Cassandra node:

```
salt 'nal01*' state.sls opencontrail exclude=opencontrail.client
```

```
Warning
The deployment may fail with two errors:
```
• Error stating that the iptables service failed. Can be ignored.
• Error stating that net.netfilter is unavailable.
Rerun the opencontrail state until only a single iptables failure remains.

9. Log in to the Salt Master node.
10. Deploy OpenContrail to the remaining network nodes simultaneously:

```
salt 'nal*' state.sls opencontrail exclude=opencontrail.client
```

11. Configure the OpenContrail resources:

```
salt -C '@opencontrail:database and *01*' state.sls opencontrail.client
```

12. Apply the following states to deploy the OpenContrail vRouters:

```
salt -C 'cmp*' saltutil.refresh_pillar
salt -C '@opencontrail:compute' saltutil.sync_all
salt -C '@opencontrail:compute' state.highstate exclude=opencontrail.client
salt -C '@opencontrail:compute' cmd.run 'reboot'
salt -C '@opencontrail:compute' state.sls opencontrail.client
```

See also
• Requirements for federated Kubernetes
• Limitations

Integrate Barbican to OpenContrail LBaaSv2

The Transport Layer Security (TLS) termination on OpenContrail HAProxy load balancer requires Barbican. Barbican is a REST API that is used for secured storage as well as for provisioning and managing of secrets such as passwords, encryption keys, and X.509 certificates.

To connect to the Barbican API, OpenContrail requires configuring the authentication in /etc/contrail/contrail-lbaas-auth.conf and the Barbican client library package python-barbicanclient installed on compute nodes.

To install the Barbican client library:

1. Deploy Barbican.
2. Open your Git project repository.
3. Include the following class to classes/cluster/<cluster_name>/openstack/compute.yml:
4. Commit and push the changes to the project Git repository.

5. Log in to the Salt Master node.

6. Update your Salt formulas at the system level:

   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.

7. Apply the following state:

   ```
   salt -C 'I@barbican:client' state.apply barbican
   ```

To configure OpenContrail for the Barbican authentication:

1. Open the classes/cluster/<cluster_name>/ directory of your Git project repository.

2. In openstack/compute.yml, include the following class:

   ```
   - service.opencontrail.compute.lbaas.barbican
   ```

3. In openstack/init.yml, edit the following parameters:

   ```
   opencontrail_barbican_user: admin
   opencontrail_barbican_password: ${_param:keystone_admin_password}
   opencontrail_barbican_tenant: admin
   ```

4. Commit and push the changes to the project Git repository.

5. Log in to the Salt Master node.

6. Update your Salt formulas at the system level:

   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.

7. Log in to the Salt Master node.

8. Apply the following state:

   ```
   salt -C 'I@opencontrail:compute' state.apply opencontrail
   ```

See also

Use HTTPS termination in OpenContrail load balancer
Enable TSN support

While deploying your MCP cluster with OpenContrail, you can connect the OpenContrail virtual network to a bare metal server through a top-of-rack (ToR) switch. Using this feature on large deployments enhances the performance of the tenant-to-tenant networking and simplifies communication with the virtual instances that run on the OpenContrail cluster.

A basic ToR services node (TSN) of the OpenContrail cluster consists of two physical servers that host the ToR Service node and ToR agents. TSN is the multicast controller of the ToR switches.

The modification of the MCP DriveTrain pipeline is not required since deployment of a TSN is the same as deploying a compute node. You only have to modify the TARGET_SERVERS field when enabling TSN on an existing MCP cluster. The configuration of TSN and the ToR agent is part of the OpenContrail compute role along with Keepalived and HAProxy.

Add a ToR services node to MCP

This section describes how to add one top-of-rack (ToR) services node (TSN) with one ToR agent to manage one ToR switch for the OpenContrail cluster in MCP.

Before you proceed with the procedure:

• If you are performing the initial deployment of your MCP cluster, verify that you have created the deployment metadata model as described in Create a deployment metadata model.

• If you are making changes to an existing MCP cluster:
  1. Verify that two physical servers dedicated for TSN are provisioned by MAAS as described in Provision physical nodes using MAAS.
  2. Verify that these two nodes are ready for deployment:


Caution!

If any of these states fail, fix the issue provided in the output and re-apply the state before you proceed to the below procedure.

To add a TSN to an MCP cluster:

1. Open your Git project repository.

2. In classes/cluster/<cluster_name>/infra/config.yml, include the following class:


3. In classes/cluster/<cluster_name>/opencontrail/init.yml, add the TSN IP addresses and ToR VIP address. For example:
opencontrail_tor01_node01_address: 172.16.174.61
opencontrail_tor01_node02_address: 172.16.174.62
opencontrail_tor01_vip_address: 172.16.174.60
opencontrail_tor01_node01_tenant_address: 172.16.175.61
opencontrail_tor01_node02_tenant_address: 172.16.175.62

4. In classes/cluster/<cluster_name>/opencontrail/, create a tor01.yml file with the following content:

```
classes:
  - system.haproxy.proxy.listen.opencontrail.tor
  - system.salt.minion.cert.opencontrail.tor
  - system.opencontrail.tor.cluster
  - system.opencontrail.client.resource.tor.yml
  - cluster.deployment_name.opencontrail.compute

parameters:
  _param:
    keepalived_vip_interface: bond0.$(_param:control_vlan)
    keepalived_vip_virtual_router_id: 61
    contrail_client_virtual_router_type: tor-service-node
    cluster_node01_address: ${_param:opencontrail_tor01_node01_address}
    cluster_node02_address: ${_param:opencontrail_tor01_node02_address}
    cluster_vip_address: ${_param:opencontrail_tor01_vip_address}
    deploy_nic: one1
    primary_first_nic: ten1
    primary_second_nic: ten2
```

5. Commit and push the changes to the project Git repository.

6. Select from the following options:

- If you are performing the initial deployment of your MCP cluster, proceed with the further cluster configuration as required.

- If you are making changes to an existing MCP cluster, re-run the Salt configuration on the Salt Master node to apply changes:

  1. Log in to the Salt Master node.
  2. Update your Salt formulas and the system level of your repository:

     1. Change the directory to /srv/salt/reclass.
     2. Run the git pull origin master command.
     3. Run the salt-call state.sls salt.master command.
     4. Run the salt-call state.sls reclass command.

  3. In the Jenkins web UI, find and open the Deploy - OpenStack Compute node pipeline.

  4. Open the Build with Parameters section.

  5. Specify the following parameters:
6. Click Deploy. To view the deployment process, see View the deployment details.

7. Configure a physical switch dedicated for TSN as described in Prepare a physical switch for TSN.

See also
Add second ToR agent to an existing TSN

Add second ToR agent to an existing TSN

After you enable TSN for OpenContrail as described in Add a ToR services node to MCP, you should have TSN running one ToR agent tor01 with ID 0 managed by default. You can also add another ToR agent to an existing TSN.

To add a ToR agent to an existing TSN:

1. Open your Git project repository.

2. In classes/cluster/<cluster_name>/opencontrail/, add the following snippet to the tor01.yml file:

```
parameters:
  opencontrail:
    compute:
      tor:
        agent:
          tor02:
            id: 1
            address: ${_param:single_address}
            port: 6634
            ssl:
              enabled: True

  haproxy:
    proxy:
      listen:
        contrail_tor02:
          type: contrail-tor
          service_name: contrail
          binds:
            - address: ${_param:cluster_vip_address}
```
3. Commit and push the changes to the project Git repository.

4. Log in to the Salt Master node.

5. Update your Salt formulas at the system level:
   1. Change the directory to /srv/salt/reclass.
   2. Run the git pull origin master command.
   3. Run the salt-call state.sls salt.master command.

6. Apply the following states:

   ```
salt '*' saltutil.refresh_pillar
salt -C 'I@opencontrail:compute:tor' state.sls haproxy,opencontrail
   ```

Now, you have two ToR agents tor01 with IDs 0 and 1 on the TSN of your OpenContrail cluster.

Prepare a physical switch for TSN

After adding the top-of-rack (ToR) services node (TSN) as described in Add a ToR services node to MCP, you must prepare a physical switch for TSN depending on your needs. This section describes how to configure the Cumulus Supermicro SSE-X3648S/R switch as an example.

The Cumulus Supermicro SSE-X3648S/R switch has the following limitations:

- VXLAN is supported only on switches in the Cumulus Linux hardware compatibility list using the Broadcom Tomahawk, Trident II+, and Trident II chipsets, as well as the Mellanox Spectrum chipset.
- VXLAN encapsulation over layer 3 sub-interfaces, for example, swp3.111, is not supported. Therefore, configure VXLAN uplinks only as layer 3 interfaces without any sub-interfaces.
- The VXLAN tunnel endpoints cannot share a common subnet. Therefore, configure at least one layer 3 hop between the VXLAN source and destination.

Note
For details about the Cumulus limitations, see the official Cumulus documentation.
To prepare the Cumulus switch for TSN:

1. Log in to the Cumulus node.
2. Add openvswitch-vtep to autostart:
   ```bash
   sudo sed -i.bak s/START=no/START=yes/g /etc/default/openvswitch-vtep
   systemctl enable openvswitch-vtep
   ```

3. Configure the Open vSwitch Database (OVSDB) in one of the following modes:
   - **Plain Transmission Control Protocol (PTCP) mode:**
     1. Verify that ovsdb listens on TCP by updating
        `/usr/share/openvswitch/scripts/ovs-ctl-vtep`. For example:
        ```bash
        set "$@" --remote=ptcp:6633:$LOCALIP
        ```
     2. Configure tunnel_ips and management_ips:
        ```bash
        sudo vtep-ctl set Physical_switch <switch_name> tunnel_ips=<tunnel_ip>
        sudo vtep-ctl set Physical_switch <switch_name> management_ips=<management_ip>
        ```
   - **Secure Sockets Layer (SSL) mode:**
     1. Log in to the TSN node.
     2. Copy the certificates to the Cumulus node:
        ```bash
        scp /etc/contrail/ssl/certs/tor.key IP:<Cumulus_IP> $PRIVATE_KEY_PATH
        scp /etc/contrail/ssl/certs/tor.crt IP:<Cumulus_IP> $CERTIFICATE_PATH
        scp /etc/contrail/ssl/certs/ca.crt IP:<Cumulus_IP> $BOOTSTRAP_CA_CERT_PATH
        ```
     3. Log in to the Cumulus node.
     4. Configure OVSDB to use generated certificates by updating
        `/usr/share/openvswitch/scripts/ovs-ctl-vtep`:
        ```bash
        set "$@" --private-key=$PRIVATE_KEY_PATH
        set "$@" --certificate=$CERTIFICATE_PATH
        set "$@" --bootstrap-ca-cert=$BOOTSTRAP_CA_CERT_PATH
        ```
     5. Define the OVSDB controller for the ToR switch. For example:
        ```bash
        sudo vtep-ctl set-manager ssl:192.168.100.17:6632
        ```

See also

- **VXLAN Layer 2 Gateway and OVSDB configuration example** in the official Juniper documentation
Verify TSN after deployment

Once you enable TSN support as described in Enable TSN support, verify that all services are up and running.

To verify TSN after deployment:

1. Log in to any OpenStack controller ctl node.
2. Verify that the Ironic bare metal nodes are in the available state:

   ironic node-list

Example of system response extract:

+----------------+-------+-------------+-----------+------------------+
|UUID            |Name   |Instance UUID|Power State|Provisioning State|
+----------------+-------+-------------+-----------+------------------+
|653309c7-a9f3...|cz7788 | None        | power off | available        |
+----------------+-------+-------------+-----------+------------------+
|7541cb97-8427...|cz7396 | None        | power off | available        |
+----------------+-------+-------------+-----------+------------------+
|0f09bef9-baed...|cz7787 | None        | power off | available        |
+----------------+-------+-------------+-----------+------------------+
|4fa42ff7-21d8...|cz7789 | None        | power off | available        |
+----------------+-------+-------------+-----------+------------------+

3. Verify the bare metal nodes settings using the ironic node-show <node_name> command. For example:

   ironic --ironic-api-version latest node-show cz7789

The nodes must be registered with the neutron network_interface to pass the connection information to Neutron.

4. Verify the connection information of the nodes that is stored in the local_link_connection field. For example:

   ironic --ironic-api-version latest node-port-list cz7789

Example of system response:

+-------------------------------+------------------+
|UUID                        |Address           |
+-------------------------------+------------------+
|0c187be9-86b0-4527-89-15-0baf24c5d263|0c:c4:7a:6a:fb:1e |
ironic --ironic-api-version latest node-port-show 0c187be9-86b0-4527-89-15-0baf24c5d263

Example of system response:

```
+---------------------------+--------------------------------------+
|Property                   |Value                                 |
+---------------------------+--------------------------------------+
|address                    |0c:c4:7a:6a:fb:1e                     |
|created_at                 |2018-01-07T08:55:32+00:00             |
|extra                      |{}                                    |
|internal_info              |{}                                    |
|local_link_connection      |{u'switch_info': u'cz-eth1303-3',     |
|                           |u'port_id': u'swp5', u'switch_id':    |
|                           |u'00:00:00:00:00:00'}                 |
|node_uuid                  |4fa42ff7-21d8-49db-bcdf-090939fc4859  |
|portgroup_uuid             |None                                  |
|pxe_enabled                |True                                  |
|updated_at                 |2018-01-08T09:47:55+00:00             |
|uuid                       |0c187be9-86b0-4527-89-15-0baf24c5d263 |
+---------------------------+--------------------------------------+
```

5. Log in to the Cumulus switch node.
6. Identify physical switches available on the node:

vtep-ctl list-ps

Example of system response:

```
cz-eth1303-3
```
7. Verify the port bindings of the switch. Use the port_id value displayed in the local_link_connection field. For example, swp5:

```
vtep-ctl list-bindings cz-eth1303-3 swp5
```

The output displays no switch ports bound to the Contrail network.

8. Log in to any OpenStack controller ctl node.

9. Display the Nova VMs spawned in the network:

```
nova list
```

Example of system response extract:

```
+---------------------+------+------+----------+-----------+--------------------+
|ID                   |Name  |Status|Task State|Power State|Networks            |
+---------------------+------+------+----------+-----------+--------------------+
|49074df4-4d4b-47a3...|demovm|ACTIVE| -        |Running    |demo-test=10.11.12.3|
+---------------------+------+------+----------+-----------+--------------------+
```

10. Provision two new bare metal servers in the same network where the Nova VMs are spawned. For example:

```
nova boot --image ubuntu16-server.qcow2 --key-name demo --nic net-name=demo-test --flavor bm_flavor --min-count 2 demo-BM-test
```

Use the nova list command to verify the provisioning status.

11. Provision the remaining bare metal servers in a different network. For example, in demo-test2:

```
nova boot --image ubuntu16-server.qcow2 --key-name demo --nic net-name=demo-test2 --flavor bm_flavor --min-count 2 demo-BM-test2
```

12. Log in to the Cumulus switch node.

13. Verify that the corresponding ports are bound to the tenant network used by Ironic for provisioning. For example:

```
vtep-ctl list-bindings cz-eth1303-3 swp5
```
14 Log in to any OpenStack controller ctl node.

15 Verify that the instances that you have created are placed on the Ironic bare metal nodes:

```
ironic node-list
```

### Example of system response extract:

<table>
<thead>
<tr>
<th>UUID</th>
<th>Name</th>
<th>Instance UUID</th>
<th>Power State</th>
<th>Provisioning State</th>
</tr>
</thead>
<tbody>
<tr>
<td>653309c7-a9f3...</td>
<td>cz7788</td>
<td>bd82aa9b-c034...</td>
<td>power on</td>
<td>active</td>
</tr>
<tr>
<td>7541cb97-8427...</td>
<td>cz7396</td>
<td>f51bf0b7-7bbd...</td>
<td>power on</td>
<td>deploying</td>
</tr>
<tr>
<td>0f09bef9-baedd...</td>
<td>cz7787</td>
<td>806ae56b-64e1...</td>
<td>power on</td>
<td>deploying</td>
</tr>
<tr>
<td>4fa42ff7-21d8...</td>
<td>cz7789</td>
<td>e60dae5f-5a2c...</td>
<td>power on</td>
<td>active</td>
</tr>
</tbody>
</table>

Wait until the Provisioning state of the bare metal machines is active.

16 Log in to the Cumulus switch node.

17 Verify that the nodes ports are connected to the tenant network. For example:
for i in {2..5}; do echo "$swp$i $(vtep-ctl list-bindings cz-eth1303-3 swp$i)"; done

Example of system response:

swp2 0000 Contrail-2b141da7-db89-49a5-9120-fb5228980761
swp3 0000 Contrail-22de534e-9ab3-4fe6-bb41-71accde56c47e
swp4 0000 Contrail-a85c8ab2-84ef-418b-aa1b-0d45a626afff
swp5 0000 Contrail-4c951bfc-ac90-54ff-1048-cd56c85a6265

18 Verify that the bare metal machines are reachable from compute nodes and the OpenContrail vRouter. For example:

1. Log in to any bare metal machine.
2. Run the following command:
   
   `ip route`

   Example of system response:

   default via 10.11.12.1 dev eno1
   10.11.12.0/24 dev eno1  proto kernel  scope link  src 10.11.12.4

3. Run the following command:
   
   `ping 10.11.12.3`
   `ping 10.11.12.2`

See also

- MCP Operations Guide: Ironic operations

See also

- OpenContrail SDN Lab testing in Mirantis blog
- Using ToR Switches and OVSDB in Juniper documentation
Deploy compute nodes

Provisioning and deploying of OpenStack or Kubernetes compute nodes (cmpX, cmp0X) is relatively straightforward and should be performed after the bare-metal provisioning through MAAS is done. You can run all states at once. Though, this has to be done multiple times with a reboot involved for changes to network configuration to take effect. The ordering of dependencies is not yet orchestrated.

To deploy a compute node:

1. Log in to the Salt Master node.
2. Verify that the new machines have connectivity with the Salt Master node:
   
   ```
   salt 'cmp*' test.ping
   ```

3. Run the reclass.storage state to refresh the deployed pillar data:
   
   ```
   salt 'cfg*' state.sls reclass.storage
   ```

4. Apply the Salt data sync and base states for Linux, NTP, OpenSSH, and Salt on the target nodes:
   
   ```
   salt 'cmp*' saltutil.sync_all
   salt 'cmp*' saltutil.refresh_pillar
   salt 'cmp*' state.sls linux,ntp,openssh,salt
   ```

   **Note**

   Refreshing the pillar data must be done every time you run the reclass state on the cfg node.

5. Apply all states for the target nodes:
   
   ```
   salt 'cmp*' state.apply
   ```
You may need to apply the states multiple times to get a successful deployment. If after two runs you still have errors, reboot the target nodes and apply the states again.

You may have an error stating that iptables is down. Ignore this error.

6. Reboot the target nodes.

After you deploy compute nodes, deploy StackLight LMA to your MCP cluster as described in Deploy StackLight LMA components.

**Deploy the DevOps Portal manually**

The DevOps Portal collects a comprehensive set of data about the cloud, offers visualization dashboards, and enables the operator to interact with a variety of tools.

The DevOps Portal is currently supported for OpenStack environments only.

This section instructs you on how to manually deploy the DevOps Portal with the Operations Support System (OSS) services available. Eventually, you will be able to access the DevOps Portal at the VIP address of the deployment on port 8800 with the following services installed:

- Push Notification service
- Runbook Automation service
- Security Audit service
- Cleanup service
- PostgreSQL database management system
- Elasticsearch back end
- Gerrit and Jenkins as part of the CI/CD deployment, will be available from the DevOps Portal web UI
- OpenLDAP and aptly as part of the CI/CD deployment
Caution!

Before you can deploy the DevOps Portal, you must complete the steps described in Deploy DriveTrain.

MCP enables you to configure the OSS services metadata in a Reclass model using Cookiecutter. Therefore, if you are performing the initial deployment of your MCP environment, you should have already configured your deployment model with the OSS parameters during the Create a deployment metadata model using the Model Designer UI stage considering the dependencies described in MCP Reference Architecture: Dependencies between services. If so, skip the procedure described in Configure services in the Reclass model and proceed to Deploy OSS services manually.

Configure services in the Reclass model

If the Reclass model of your deployment does not include metadata for OSS services, you must define it in the Reclass model before proceeding with the deployment of the DevOps portal.

Note

Before proceeding, consider the dependencies described in MCP Reference Architecture: Dependencies between services.

To configure OSS services in the Reclass model:

1. In the init.yml file in the /srv/salt/reclass/classes/cluster/${_param:cluster_name}/cicd/control/ directory, define the required classes.

   The following code snippet contains all services currently available. To configure your deployment for a specific use case, comment out the services that are not required:

   ```yaml
   classes:
   # GlusterFS
   - system.glusterfs.server.volume.devops_portal
   - system.glusterfs.server.volume.elasticsearch
   - system.glusterfs.server.volume.mongodb
   - system.glusterfs.server.volume.postgresql
   - system.glusterfs.server.volume.pushkin
   - system.glusterfs.server.volume.rundeck
   - system.glusterfs.server.volume.security_monkey
   - system.glusterfs.client.volume.devops_portal
   - system.glusterfs.client.volume.elasticsearch
   - system.glusterfs.client.volume.mongodb
   ```
2. In the init.yml file in the /srv/salt/reclass/classes/cluster/${_param:cluster_name}/cicd/control/ directory, define the required parameters:

• For the Runbook Automation service, define:

```yaml
parameters:
  _param:
  runbook_runbook_public_key: <SSH_PUBLIC_KEY>
```
• For the Security Audit service, define:

```
parameters:
  _param:
    security_monkey_openstack:
      username: <USERNAME>
      password: <PASSWORD>
      auth_url: <KEYSTONE_AUTH_ENDPOINT>
```

The configuration for the Security Audit service above will use the Administrator account to access OpenStack with the admin tenant. To configure the Security Audit service deployment for a specific tenant, define the `security_monkey_openstack` parameter as follows:

```
parameters:
  _param:
    security_monkey_openstack:
      os_account_id: <OS_ACCOUNT_ID>
      os_account_name: <OS_ACCOUNT_NAME>
      username: <USERNAME>
      password: <PASSWORD>
      auth_url: <KEYSTONE_AUTH_ENDPOINT>
      project_domain_name: <PROJ_DOMAIN_NAME>
      project_name: <PROJ_NAME>
      user_domain_name: <USER_DOMAIN_NAME>
```

Warning

The `project_name: <PROJ_NAME>` parameter specifies a project for the Keystone authentication in the Security Audit service. Therefore, the service will not listen by projects, but synchronize issues from all projects in the current environment with the DevOps Portal using the specified project to authenticate.

• For the Janitor service, define:

```
parameters:
  _param:
    janitor_monkey_openstack:
      username: <USERNAME>
      password: <PASSWORD>
      auth_url: <KEYSTONE_AUTH_ENDPOINT>
```
The configuration for the Janitor service above will use the Administrator account to access OpenStack with the admin tenant. To configure the Security Audit service deployment for a specific tenant, define the `janitor_monkey_openstack` parameter as follows:

```yaml
parameters:
  _param:
    janitor_monkey_openstack:
      username: <USERNAME>
      password: <PASSWORD>
      auth_url: <KEYSTONE_AUTH_ENDPOINT>
      project_domain_name: <PROJ_DOMAIN_NAME>
      project_name: <PROJ_NAME>
```

3. In the `master.yml` file in the `/srv/salt/reclass/classes/cluster/${_param:cluster_name}/cicd/control/` directory, configure classes and parameters as required:

- Define classes for the DevOps Portal and services as required:

```yaml
classes:
  # DevOps Portal
  - service.devops_portal.config

  # Elasticsearch
  - system.elasticsearch.client
  - system.elasticsearch.client.index.pushkin
  - system.elasticsearch.client.index.janitor_monkey

  # PostgreSQL
  - system.postgresql.client.pushkin
  - system.postgresql.client.rundeck
  - system.postgresql.client.security_monkey

  # Runbook Automation
  - system.rundeck.server.docker
  - system.rundeck.client
```

- Define parameters for the Runbooks Automation service, if required:

```yaml
parameters:
  _param:
    rundeck_db_user: ${_param:rundeck_postgresql_username}
    rundeck_db_password: ${_param:rundeck_postgresql_password}
    rundeck_db_host: ${_param:cluster_vip_address}
    rundeck_postgresql_host: ${_param:cluster_vip_address}
    rundeck_postgresql_port: ${_param:haproxy_postgresql_bind_port}
```

4. Push all changes of the model to the dedicated project repository.
5. Verify that the metadata of the Salt Master node contains all the required parameters:

```
reclass --nodeinfo=${SALT_MASTER_FQDN}.${ENV_DOMAIN}
salt '*' saltutil.refresh_pillar
salt '*' saltutil.sync_all
salt '${SALT_MASTER_FQDN}.${ENV_DOMAIN}' pillar.get devops_portal
```

For example, for the ci01 node on the cicd-lab-dev.local domain run:

```
reclass --nodeinfo=ci01.cicd-lab-dev.local
salt '*' saltutil.refresh_pillar
salt '*' saltutil.sync_all
salt 'ci01.cicd-lab-dev.local' pillar.get devops_portal
```

### Deploy OSS services manually

Before you proceed with the services installation, verify that you have updated the Reclass model accordingly as described in Configure services in the Reclass model.

To deploy the DevOps portal:

1. Log in to the Salt Master node.
2. Refresh Salt pillars and synchronize Salt modules on all Salt Minion nodes:

```
salt '*' saltutil.refresh_pillar
salt '*' saltutil.sync_all
```

3. Set up GlusterFS:

```
salt -b 1 -C '@glusterfs:server' state.sls glusterfs.server
```

#### Note

The -b option specifies the explicit number of the Salt Minion nodes to apply the state at once to. Therefore, you will get a more stable configuration during the establishment of peers between the services.

4. Mount the GlusterFS volume on Docker Swarm nodes:

```
salt -C '@glusterfs:client' state.sls glusterfs.client
```

5. Verify that the volume is mounted on Docker Swarm nodes:

```
salt '*' cmd.run 'systemctl -a|grep "GlusterFS File System"|grep -v mounted'
```
6. Configure HAProxy and Keepalived for the load balancing of incoming traffic:

```
salt -C "I@haproxy:proxy" state.sls haproxy,keepalived
```

7. Set up Docker Swarm:

```
salt -C 'I@docker:host' state.sls docker.host
salt -C 'I@docker:swarm:role:master' state.sls salt
salt -C 'I@docker:swarm:role:master' mine.flush
salt -C 'I@docker:swarm:role:master' mine.update
salt -C 'I@docker:swarm' state.sls docker.swarm
salt -C 'I@docker:swarm:role:master' cmd.run 'docker node ls'
```

8. Configure the OSS services:

```
salt -C 'I@devops_portal:config' state.sls devops_portal.config
salt -C 'I@rundeck:server' state.sls rundeck.server
```

**Note**
In addition to setting up the server side for the Runbook Automation service, the rundeck.server state configures users and API tokens.

9. Prepare aptly before deployment:

```
salt -C 'I@aptly:publisher' saltutil.refresh_pillar
salt -C 'I@aptly:publisher' state.sls aptly.publisher
```

10. Apply the docker.client state:

```
salt -C 'I@docker:swarm:role:master' state.sls docker.client
```

11. Prepare Jenkins for the deployment:

```
salt -C 'I@docker:swarm' cmd.run 'mkdir -p /var/lib/jenkins'
```

12. Identify the IP address on which HAProxy listens for stats:

```
HAPROXY_STATS_IP=$(salt -C 'I@docker:swarm:role:master' \
  --out=newline_values_only \n  pillar.fetch haproxy:proxy:listen:stats:binds:address)
```
Caution!

You will use the HAPROXY_STATS_IP variable to verify that the Docker-based service you are going to deploy is up in stats of the HAProxy service.

13 Verify that aptly is UP in stats of the HAProxy service:

```
curl -s "http://$HAPROXY_STATS_IP:9600/haproxy?stats;csv" | grep aptly
```

14 Deploy aptly:

```
salt -C 'I@aptly:server' state.sls aptly
```

15 Verify that OpenLDAP is UP in stats of the HAProxy service:

```
curl -s "http://$HAPROXY_STATS_IP:9600/haproxy?stats;csv" | grep openldap
```

16 Deploy OpenLDAP:

```
salt -C 'I@openldap:client' state.sls openldap
```

17 Verify that Gerrit is UP in stats of the HAProxy service:

```
curl -s "http://$HAPROXY_STATS_IP:9600/haproxy?stats;csv" | grep gerrit
```

18 Deploy Gerrit:

```
salt -C 'I@gerrit:client' state.sls gerrit
```

Note

The execution of the command above may hang for some time. If it happens, re-apply the state after its termination.

19 Verify that Jenkins is UP in stats of the HAProxy service:

```
curl -s "http://$HAPROXY_STATS_IP:9600/haproxy?stats;csv" | grep jenkins
```

20 Deploy Jenkins:

```
salt -C 'I@jenkins:client' state.sls jenkins

Note
The execution of the command above may hang for some time. If it happens, re-apply the state after its termination.

21 Verify that the process of bootstrapping of the PostgreSQL container has been finalized:
   
docker service logs postgresql_db | grep "ready to accept"

22 Verify that PostgreSQL is UP in stats of the HAProxy service:
   
curl -s "http://${HAPROXY_STATS_IP}:9600/haproxy?stats;csv" | grep postgresql

23 Initialize OSS services databases by setting up the PostgreSQL client:
   
salt -C 'I@postgresql:client' state.sls postgresql.client

The postgresql.client state application will return errors due to cross-dependencies between the docker.stack and postgresql.client states. To configure integration between Push Notification and Security Audit services:

1. Verify that Push Notification service is UP in stats of the HAProxy service:
   
curl -s "http://${HAPROXY_STATS_IP}:9600/haproxy?stats;csv" | grep pushkin

2. Re-apply the postgresql.client state:
   
salt -C 'I@postgresql:client' state.sls postgresql.client

24 Verify that Runbook Automation is UP in stats of the HAProxy service:
   
curl -s "http://${HAPROXY_STATS_IP}:9600/haproxy?stats;csv" | grep rundeck

25 Deploy Runbook Automation:
   
salt -C 'I@rundeck:client' state.sls rundeck.client

26 Verify that Elasticsearch is UP in stats of the HAProxy service:
   
curl -s "http://${HAPROXY_STATS_IP}:9600/haproxy?stats;csv" | grep elasticsearch
27 Deploy the Elasticsearch back end:

```bash
salt -C 'I@elasticsearch:client' state.sls elasticsearch.client
```

Due to index creation, you may need to re-apply the state above.

28 If required, generate documentation and set up proxy to access it. The generated content will reflect the current configuration of the deployed environment:

```bash
salt -C 'I@sphinx:server' state.sls 'sphinx'
```

Build a custom image of the DevOps Portal

For testing purposes, you may need to create a custom Docker image to use it while deploying the DevOps Portal.

To build a custom Docker image:

1. Before you build the image and upload it to Sandbox, clone the source code of DevOps Portal:
   ```bash
git clone https://gerrit.mcp.mirantis.net/oss/devops-portal
cd devops-portal
```

2. Build your image:
   ```bash
docker build --rm -f docker/Dockerfile -t \
```

3. Push the image into a specific prefix on Sandbox:
   ```bash
docker push docker-sandbox.sandbox.mirantis.net/[USERNAME]/oss/devops-portal:latest
```

Configure Salesforce integration for OSS manually

The Push Notification services can automatically create tickets in Salesforce based on the alarms triggered by the issues that are found by Prometheus Alertmanager. Moreover, the Push Notification service ensures the following:

- The Salesforce tickets are not duplicated. When the same alarm gets triggered multiple times, only one Salesforce ticket is created per alarm.

- The Push Notification service creates one entry in a Salesforce feed, that is a FeedItem, per alarm with a link to an existing ticket. This enables the users to track important changes as well as close the ticket which has been fixed.
Warning

This section describes how to manually configure the Push Notification service Reclase metadata to integrate with Salesforce in an existing OSS deployment. Therefore, if you want to configure the Salesforce integration, perform the procedure below.

Otherwise, if you are performing the initial deployment of your MCP environment, you should have already configured your deployment model with the Salesforce (SFDC) parameters as described in OSS parameters. In this case, skip this section.

To configure Salesforce integration for OSS manually:

1. Collect the following data from Saleforce:
   
   • **auth_url**
     - The URL of a Salesforce instance. The same for the MCP users.
   
   • **username**
     - The username in Salesforce used for integration; all Salesforce cases are created by this user. The unique identifier for an MCP user.
   
   • **password**
     - The password used for logging in to the Support Customer Portal. The unique identifier for an MCP user.
   
   • **environment**
     - The Cloud ID in Salesforce. The unique identifier for an MCP user.
     - The detailed information on a Salesforce Cloud is provided by either Mirantis support engineers or customer depending on whom the Cloud object was created by.
   
   • **consumer_key**
     - The Consumer Key in Salesforce required for Open Authorization (OAuth).
   
   • **consumer_secret**
     - The Consumer Secret from Salesforce required for OAuth.
   
   • **organization_id**
     - The Salesforce Organization ID in Salesforce required for OAuth.

2. Verify that the following services are properly configured and deployed:

   • Elasticsearch
   • PostgreSQL

Note

For the configuration and deployment details, see:
• Configure services in the Reclass model
• Deploy OSS services manually

3. In the `classes/cluster/${_param:cluster_name}/oss/client.yml` file of your deployment model, define the `system.postgresql.client.sfdc` class:

```
classes:
- system.postgresql.client.sfdc
```

4. In the `/srv/salt/reclass/classes/cluster/${_param:cluster_name}/oss/server.yml` file, define the following parameters:

```
parameters:
  _param:
    # SFDC configuration
    sfdc_auth_url: <AUTH_URL>
    sfdc_username: <USERNAME>
    sfdc_password: <PASSWORD>
    sfdc_consumer_key: <CONSUMER_KEY>
    sfdc_consumer_secret: <CONSUMER_SECRET>
    sfdc_organization_id: <ORGANIZATION_ID>
    sfdc_sandbox_enabled: True
```

Note
Sandbox environments are isolated from the production Salesforce clouds. Set the `sfdc_sandbox_enabled` to True to use Salesforce sandbox for testing and evaluation purposes. Verify that you specify the correct sandbox-url value in the `sfdc_auth_url` parameter. Otherwise, set the parameter to False.

5. Push all changes of the model to the dedicated project repository.

6. Refresh pillars and synchronize Salt modules:

```
salt '*' saltutil.refresh_pillar
salt '*' saltutil.sync_modules
```

7. If you have the running pushkin docker stack, remove it and apply the following Salt states:

```
salt -C '@docker:swarm:role:master' state.sls docker.client
salt -C '@postgresql:client' state.sls postgresql.client
```
8. To test whether the Push Notification service is configured properly:

1. View the list of all applications, preconfigured in the Push Notification service, and their details by checking the system response for the following command:

   ```bash
curl -D - http://${HAPROXY_STATS_IP}:8887/apps
   ```

   Example of system response:

   ```json
   {
     "applications": [{
       "login_id": 11,
       "enabled": true,
       "id": 1,
       "name": "notify_service"
     }]
   }
   ```

2. Send the test request to the service using the following command:

   ```bash
curl -i -XPOST -H 'Content-Type: application/json' <PUSH_NOTIFICATION_ENDPOINT> -d \\
   
   
   
   ```

   The table below provides the description of the parameters required for the test request.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_id</td>
<td>The Login ID of an application on behalf of which the notification will be send. Define the parameter according to the login_id parameter value retrieved during the previous step.</td>
</tr>
<tr>
<td>environment</td>
<td>The Cloud ID in Salesforce which the notification will be send to. Define the parameter according to the environment parameter value collected during the first step of this procedure.</td>
</tr>
<tr>
<td>application_id</td>
<td>The ID of an application on behalf of which the notification will be send. Define the parameter according to the id parameter value retrieved during the previous step.</td>
</tr>
</tbody>
</table>

   Example:

   ```bash
curl -i -XPOST -H 'Content-Type: application/json' http://${HAPROXY_STATS_IP}:8887/post_notification_json -d \\
   
   ```
3. Log in to Salesforce and verify that the alert is filed correctly.

Configure email integration for OSS manually

The Push Notification service can route notifications based on the alarms triggered by the issues that are found by Prometheus Alertmanager through email.

Warning
This section describes how to manually configure the Push Notification service Reclass metadata to integrate email routing for notifications in an existing OSS deployment. Therefore, if you want to configure the email routing configuration, perform the procedure below.

Otherwise, if you are performing the initial deployment of your MCP environment, you should have already configured your deployment model with the default Simple Mail Transfer Protocol (SMTP) parameters for the Push Notification service as described in OSS parameters and the OSS webhook parameters as described in StackLight LMA product parameters. In this case, skip this section.

Note
The Push Notification service only routes the received notifications to email recipients. Therefore, you must also provide the Prometheus Alertmanager service with a predefined alert template containing an email handler as described in MCP Operations Guide: Enable notifications through the Push Notification service.

To configure email integration for OSS manually:

1. Obtain the following data:
   - pushkin_smtp_host
     SMTP server host for email routing. Gmail server host is used by default (smtp.gmail.com).
   - pushkin_smtp_port
     SMTP server port for email routing. Gmail server port is used by default (587).
   - webhook_from
     Source email address for notifications sending.
   - pushkin_email_sender_password
     Source email password for notifications sending.
   - webhook_recipients
Comma-separated list of notification recipients.

2. Verify that the following services are properly configured and deployed:

   - Elasticsearch
   - PostgreSQL

   Note
   For the configuration and deployment details, see:
   - Configure services in the Reclass model
   - Deploy OSS services manually

3. In the /srv/salt/reclass/classes/cluster/${_param:cluster_name}/oss/server.yml file, define the following parameters:

   ```yaml
   parameters:
     _param:
     pushkin_smtp_host: smtp.gmail.com
     pushkin_smtp_port: 587
     webhook_from: your_sender@mail.com
     pushkin_email_sender_password: your_sender_password
     webhook_recipients: "recipient1@mail.com,recipient2@mail.com"
   ```

4. Push all changes of the model to the dedicated project repository.

5. Refresh pillars and synchronize Salt modules:

   ```bash
   salt '*' saltutil.refresh_pillar
   salt '*' saltutil.sync_modules
   ```

6. If you have the running pushkin docker stack, remove it and apply the following Salt states:

   ```bash
   salt -C 'I@docker:swarm:role:master' state.sls docker.client
   ```

**Deploy StackLight LMA components**

StackLight LMA is the Logging, Monitoring, and Alerting toolchain, the capacity planning, operational health, and response monitoring solution for Mirantis Cloud Platform (MCP).

StackLight LMA is based on the time-series database and flexible cloud-native monitoring solution called Prometheus. Prometheus provides powerful querying capabilities and integrates with Grafana providing real-time visualization.

This section explains how to configure and install StackLight LMA including the components that it integrates after you deploy a Kubernetes cluster or an OpenStack environment on your MCP cluster.
Caution!

If you plan to install StackLight LMA with the DevOps Portal, you should deploy the OSS sub-cluster first as described in Deploy the DevOps Portal manually.

Before you start installing the StackLight LMA components, verify that your MCP cluster meets the StackLight LMA hardware requirements.

Prerequisites

Before you start installing the StackLight LMA components, complete the following steps:

1. Configure StackLight LMA for installation.
   
   The configuration of StackLight LMA for installation is defined in the Reclass model. See stacklight-salt-model as an example of the Reclass model to install StackLight LMA on Mirantis Cloud Platform. Three levels of the Reclass models are currently collocated on the Salt Master node under the /srv/salt/reclass/classes directory:
   
   • The service level model is imported directly from the metadata/service directory of all MCP formulas. The Reclass parameters that are defined at the service level are the most generic parameters and should not be modified in practice.
   
   • The system level model, which is currently defined in the user Reclass model, imports the service level models and defines additional parameters. The parameters defined in the system level model relate to the system-wide configuration of StackLight LMA, such as the IP address and port number of the Elasticsearch and InfluxDB servers.
   
   • The cluster level model defines the configuration of StackLight LMA for a particular deployment. A user Reclass model to install OpenStack with StackLight LMA must be created. This is where you typically define the name of the InfluxDB database, username, password of the InfluxDB admin, and others.

2. Deploy Docker Swarm master:

   ```
   salt -C 'I@docker:host' state.sls docker.host
   salt -C 'I@docker:swarm:role:master' state.sls docker.swarm
   ```

3. Deploy Docker Swarm workers:

   ```
   salt -C 'I@docker:swarm:role:manager' state.sls docker.swarm -b 1
   ```

4. Deploy Keepalived:

   ```
   salt -C 'I@keepalived:cluster' state.sls keepalived -b 1
   ```

5. Deploy NGINX proxy:

   ```
   salt -C 'I@nginx:server' state.sls nginx
   ```
6. Verify that you have Internet access to download several external packages that are not included in the standard Ubuntu distribution. If there is no Internet access, these repositories must be mirrored on MCP.

**Install back ends for StackLight LMA**

StackLight LMA integrates several back-end servers to visualize an environment monitoring and health statuses. This section describes how to install various back-end visualization solutions, including: InfluxDB, Elasticsearch, and Kibana. For a Kubernetes-based MCP cluster, additionally install Galera.

**Install Elasticsearch and Kibana**

The Elasticsearch and Kibana servers must be installed on the log cluster of the Mirantis Cloud Platform.

**Limitations**

StackLight LMA has the following limitations for the Elasticsearch and Kibana deployment:

- The Elasticsearch and Kibana cluster must be installed on a minimum of three nodes to avoid split-brain issues.
- Advanced cluster operations may require manual steps.

**Configure Elasticsearch and Kibana**

The configuration parameters of the Elasticsearch engine and Kibana dashboards are defined in the corresponding Salt formulas. For details and the configuration examples, see the GitHub projects for [SaltStack Elasticsearch formula](https://github.com/SaltStack/SaltStack) and [SaltStack Kibana formula](https://github.com/SaltStack/SaltStack).

**Deploy Elasticsearch and Kibana**

The deployment of Elasticsearch and Kibana consists of the server and the client deployment.

To deploy Elasticsearch and Kibana:

1. Deploy the Elasticsearch and Kibana services:

   ```
salt -C 'I@elasticsearch:server' state.sls elasticsearch.server -b 1
salt -C 'I@kibana:server' state.sls kibana.server -b 1
```

2. Deploy the Elasticsearch and Kibana clients that will configure the corresponding servers:

   ```
salt -C 'I@elasticsearch:client' state.sls elasticsearch.client
salt -C 'I@kibana:client' state.sls kibana.client
```

**Verify Elasticsearch and Kibana after deployment**

After you deploy Elasticsearch and Kibana, verify that they are up and running using the steps below.

To verify the Elasticsearch cluster:
1. Log in to one of the log hosts.
2. Run the following command:

   ```bash
   curl http://log:9200
   ```

   Example of the system response:

   ```json
   curl http://log:9200
   {
     "name": "log01",
     "cluster_name": "elasticsearch",
     "cluster_uuid": "KJM5s5CkTNKGkh807gcCg",
     "version": {
       "number": "2.4.4",
       "build_hash": "fcbb46df45562a9cf00c604b30849a6dec6b017",
       "build_timestamp": "2017-06-03T11:33:16Z",
       "build_snapshot": false,
       "lucene_version": "5.5.2"
     },
     "tagline": "You Know, for Search"
   }
   ```

To verify the Kibana dashboard:

1. Log in to the Salt Master node.
2. Identify the prx VIP of your MCP cluster:

   ```bash
   salt-call pillar.get _param:openstack_proxy_address
   ```

3. Open a web browser.
4. Paste the prx VIP and the default port 5601 to the web browser address field. No credentials are required.

   Once you access the Kibana web UI, you must be redirected to the Kibana Logs analytics dashboard.

Install InfluxDB

The InfluxDB server must be installed on the monitoring cluster of the Mirantis Cloud Platform.

Limitations

StackLight LMA has the following limitations for the InfluxDB deployment:

- InfluxDB should be deployed on three nodes in the active/passive failover mode. It means that only one node stores data at a time, while other nodes are idle.

- The current implementation of StackLight LMA for MCP uses the latest version of the community edition, InfluxDB 1.4. However, InfluxDB 1.4 does not support clustering. If you
want to deploy InfluxDB in cluster mode for HA and scale-out, use the InfluxEnterprise version.

Configure InfluxDB
The configuration parameters of the InfluxDB are defined in the Salt formula. For details and configuration examples, see the SaltStack InfluxDB formula GitHub project.

Deploy InfluxDB
Depending on the cluster Reclass model, the InfluxDB server may run on one or several nodes of the monitoring cluster. But only one InfluxDB server is active at a time.

Note
To use a fully supported InfluxDB cluster for HA and scale-out, install the InfluxEnterprise version separately.

To deploy InfluxDB:
1. Log in to the Salt Master node.
2. Run the following command:

   salt -C 'I@influxdb:server' state.sls influxdb

By applying this state, you install InfluxDB on the Salt minion nodes that have the influxdb:server role defined in the Salt pillar.

Note
To identify the Salt minion nodes that have the influxdb:server role defined in the Salt pillar, run:

   salt -C 'I@influxdb:server' grains.get id

Verify InfluxDB
Depending on the number of nodes and deployment setup, deploying InfluxDB may take up to several hours.

To verify InfluxDB:
1. Log in to the Salt Master node.
2. Run the following command:
3. In the interactive InfluxDB CLI, view the dump file of all the collected time series:

```
> show series
```

Install Galera (MySQL)

For the Kubernetes-based MCP clusters, you must also install Galera as a back end for StackLight LMA. Galera is a synchronous multi-master database cluster based on the MySQL storage engine.

To install Galera:

1. Log in to the Salt Master node.
2. Apply the galera state:

```
salt -C 'I@galera:master' state.sls galera
salt -C 'I@galera:slave' state.sls galera -b 1
```

3. Verify that Galera is up and running:

```
salt -C 'I@galera:master' mysql.status | grep -A1 wsrep_cluster_size
salt -C 'I@galera:slave' mysql.status | grep -A1 wsrep_cluster_size
```

Install the StackLight LMA components

After you deploy the Elasticsearch and the InfluxDB back ends and their components as described in Install back ends for StackLight LMA, proceed to configuring and installing Prometheus-based StackLight LMA.

**Warning**

If any of the steps below fail, do not proceed without resolving the issue.

To install the StackLight LMA components:

1. Log in to the Salt Master node.
2. Install Telegraf:

```
salt -C 'I@telegraf:agent or I@telegraf:remote_agent' state.sls telegraf
```

This formula installs the Telegraf package, generates configuration files, and starts the Telegraf service.
3. Configure Prometheus exporters:

```bash
salt -C 'I@prometheus:exporters' state.sls prometheus
```

4. Configure Fluentd:

```bash
salt -C 'I@fluentd:agent' state.sls fluentd.agent
```

5. Generate the configuration for services running in Docker Swarm:

```bash
salt -C 'I@docker:swarm and I@prometheus:server' state.sls prometheus -b 1
```

6. Deploy Prometheus long-term storage. Skip this step if you have selected InfluxDB as a long-term storage when creating the deployment metadata model.

```bash
salt -C 'I@prometheus:relay' state.sls prometheus
```

7. Deploy the monitoring containers:

```bash
salt -C 'I@docker:swarm:role:master and I@prometheus:server' state.sls docker
```

8. Configure the Grafana client:

```bash
salt -C 'I@grafana:client' state.sls grafana.client
```

9. Proceed to Verify the StackLight LMA components after deployment.

### Verify the StackLight LMA components after deployment

Once you install the StackLight LMA components as described in Install the StackLight LMA components, verify that all components have been successfully deployed and all services are up and running.

To verify the StackLight LMA components:

1. Log in to the Salt Master node.

2. Verify that all the monitoring services running in Docker Swarm have their expected number of replicas:

```bash
salt -C 'I@docker:client:stack:monitoring' cmd.run 'docker service ls'
```

Example:

```
root@sup01:~# docker service ls
ID           NAME                              MODE       REPLICAS IMAGE
j0hrith0agyux monitoring_server                 replicated 1/1      prometheus:latest
pqqeqda711a69 dashboard_grafana                 replicated 1/1      grafana/grafana:latest
xrdmspdxojs monitoring_pushgateway            replicated 2/2      pushgateway:latest
```
3. Verify the status of the containers:

```bash
salt -C 'I@docker:swarm:role:master and I@prometheus:server' cmd.run \
'docker service ps $(docker stack services -q monitoring)'
```

4. Inspect the monitoring containers logs for any unusual entries:

```bash
salt -C 'I@docker:swarm:role:master and I@prometheus:server' cmd.run \
'for i in $(docker stack services -q monitoring); do docker service logs --tail 10 $i; done'
```

5. Verify that the Fluentd service is running:

```bash
salt -C 'I@fluentd:agent' service.status td-agent
```

6. Verify Prometheus Relay:

```bash
salt -C 'I@prometheus:relay' service.status prometheus-relay
```

7. If deployed, verify Prometheus long-term storage:

```bash
salt -C 'I@prometheus:relay' service.status prometheus
```

8. Verify the Prometheus web UI:

   1. Connect to the Prometheus web UI as described in the corresponding section of the MCP Operations Guide.
   2. From the Status drop-down list, select Targets.
   3. Verify that all targets are in the UP state.
   4. Click the Alerts tab.
   5. Verify that no alerts are active.

9. Verify the Alertmanager web UI:

   1. Connect to the Alertmanager web UI as described in Use the Alertmanager web UI.
   2. Click Alerts.
   3. Verify that no alerts are active.
10 Verify the Grafana dashboards:
   1. Enter the prx VIP on port 3000 by default.
   2. Authenticate using your credentials as described in Connect to Grafana. You should be redirected to the Grafana Home page with a list of available dashboards sorted by name.
   3. Verify that all nodes are listed in the System dashboard.
11 Verify the Kibana dashboards by connecting to Kibana as described in the Connect to Kibana.

See also
- Logging, monitoring, and alerting planning
- Logging, monitoring, and alerting operations

**Finalize the deployment**
The last step of a manual deployment is ensuring highstates on all nodes.

To ensure highstates:

1. Log in to the Salt Master node.
2. Verify that all machines have connectivity with the Salt Master node:
   ```
   salt '*' test.ping
   ```
3. Ensure highstate on the Salt Master node:
   ```
   salt-call state.apply -l debug
   ```
4. Ensure highstate on the GlusterFS nodes one by one to avoid race condition:
   ```
   salt -C 'I@glusterfs:server' state.apply -b1 -l debug
   ```
5. Ensure highstate on the rest of the nodes:
   ```
   salt -C '* and not I@glusterfs:server and not cfg*' state.apply -l debug
   ```
Enable NFV features

Network Functions Virtualization (NFV) is a powerful technology that leverages virtualization of particular network functions which allows a better flexibility in network administration and enables you to use network hardware more efficiently.

MCP supports the following NFV features:

- Data Plane Development Kit or DPDK is a set of libraries and drivers to perform fast packet processing in the user space that OVS/vRouter can use to move network packets processing from a kernel to a user space. OVS/vRouter with DPDK acceleration on compute nodes reduces the processing time of network packets transferred between a host’s network interface and a guest bypassing the host's kernel. Moreover, DPDK leverages benefits of usage of other technologies such as Huge Pages, CPU pinning, and NUMA topology scheduling.

- SR-IOV is an extension to the PCI Express (PCIe) specification that enables a network adapter to separate access to its resources among various PCIe hardware functions: Physical Function (PF) and Virtual Functions (VFs). As a result, you can achieve near bare-metal performance, since network devices can forward traffic directly to a VF bypassing the host.

- Multiqueue for DPDK-based vrouters enables the scaling of packet sending/receiving processing to the number of available vCPUs of a guest by using multiple queues.

The following table shows compatibility matrix for MCP of NFV features for different deployments.

<table>
<thead>
<tr>
<th>Type</th>
<th>Host OS</th>
<th>Kernel</th>
<th>Huge Pages</th>
<th>DPDK</th>
<th>SR-IOV</th>
<th>NUMA</th>
<th>CPU pinning</th>
<th>Multiqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVS</td>
<td>Xenial</td>
<td>4.8</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Kernel vRouter</td>
<td>Xenial</td>
<td>4.8</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DPDK vRouter</td>
<td>Trusty</td>
<td>4.4</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No (version 3.2)</td>
</tr>
<tr>
<td>DPDK OVS</td>
<td>Xenial</td>
<td>4.8</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Enable DPDK

Enabling Data Plane Development Kit (DPDK) strongly requires Huge Pages configuration before an application start. To perform fast packet processing, a DPDK-based network application may require to use isolated CPUs and resources spread on the multi-NUMA topology. These configurations are common for both OVS and OpenContrail.
Warning
Before you proceed with the DPDK enabling, verify that you have performed the following procedures:

1. Enable Huge Pages
2. Configure NUMA and CPU pinning architecture

Enable OVS DPDK
This section explains how to prepare for and enable OVS DPDK in MCP.

Warning
Before you proceed with the DPDK enabling, verify that you have performed the following procedures:

1. Enable Huge Pages
2. Configure NUMA and CPU pinning architecture

Limitations
The usage of the OVS DPDK feature in MCP includes the following limitations:

- OVS DPDK can be used only for tenant traffic
- Compute with DPDK cannot be used for non-dpdk workload

Prepare your environment for OVS DPDK
This section describes the initialization steps needed to prepare your deployment for the enablement of the OVS DPDK feature.

Warning
Before you proceed with the DPDK enabling, verify that you have performed the following procedures:

1. Enable Huge Pages
2. Configure NUMA and CPU pinning architecture

To prepare your environment for OVS DPDK:

1. Specify the DPDK driver.
DPDK Environment Abstract Layer (EAL) uses either Userspace I/O (UIO) module or VFIO to provide userspace access on low-level buffers. MCP supports both configurations.

**Note**

To use VFIO approach, verify that both kernel and BIOS are configured to use I/O virtualization. This requirement is similar to SR-IOV Intel IOMMU and VT-d being enabled.

To use one of Userspace I/O drivers, define the `compute_dpdk_driver` parameter. For example:

```
compute_dpdk_driver: uio # vfio
```

2. In respect to the parameter specified above, configure the DPDK physical driver. There is one-to-one dependency of what driver must be selected for physical DPDK NIC based on the configured I/O mechanism. For example:

```
dpdk0:
...
  driver: igb_uio # vfio-pci
```

3. To enable the physical DPDK device to run several RX/TX queues for better packet processing, configure the following parameter specifying the number of queues to be used. For example:

```
dpdk0:
...
  n_rxq: 2 # number of RX/TX queues
```

**Note**

The increasing number of queues results in PMD threads consuming more cycles to serve physical device. We strongly recommend that you configure the number of physical queues not greater that CPUs configured for the DPDK-based application.

Enable OVS DPDK support

Before you proceed with the procedure, verify that you have performed the preparatory steps described in Prepare your environment for OVS DPDK.

While enabling DPDK for Neutron Open vSwitch, you can configure a number of settings specific to your environment that assist in optimizing your network performance, such as manual pinning and others.
To enable OVS DPDK:

1. Verify your NUMA nodes on the host operating system to see what vCPUs are available. For example:

   ```bash
   lscpu | grep NUMA
   NUMA node(s):          1
   NUMA node0 CPU(s):     0-11
   ```

2. Include the class to cluster.<name>.openstack.compute and configure the dpdk0 interface. Select from the following options:

   - Single interface NIC dedicated for DPDK:

     ```yaml
     ...
     - system.neutron.compute.nfv.dpdk
     ...
     parameters:
       linux:
         network:
           interfaces:
             ...
             # other interface setup
             ...
             dpdk0:
               name: ${_param:dpdk0_name}
               pci: ${_param:dpdk0_pci}
               driver: igb_uio
               enabled: true
               type: dpdk_ovs_port
               n_rxq: 2
               br-prv:
                 enabled: true
                 type: dpdk_ovs_bridge
     ```

   - OVS DPDK bond with 2 dedicated NICs

     ```yaml
     ...
     - system.neutron.compute.nfv.dpdk
     ...
     parameters:
       linux:
         network:
           interfaces:
             ...
             # other interface setup
             ...
             dpdk0:
               name: ${_param:dpdk0_name}
               pci: ${_param:dpdk0_pci}
     ```
3. Calculate the hexadecimal coremask.

As well as for OpenContrail, OVS-DPDK needs logical cores parameter to be set. Open vSwitch requires two parameters: lcore mask to DPDK processes and PMD mask to spawn threads for poll-mode packet processing drivers. Both parameters must be calculated respectively to isolated CPUs and are representing hexadecimal numbers. For example, if we need to take single CPU number 2 for Open vSwitch and 4 CPUs with numbers 5, 6, 10 and 12 for forwarding PMD threads, we need to populate parameters below with the following numbers:

- The lcore mask example:

- PMD CPU mask example:

4. Define the parameters in the cluster.<name>.openstack.init if they are the same for all compute nodes. Otherwise, specify them in cluster.<name>.infra.config:

- dpdk0_name
  Name of port being added to OVS bridge
- dpdk0_pci
  PCI ID of physical device being added as a DPDK physical interface
- compute_dpdk_driver
  Kernel module to provide userspace I/O support
• compute_ovs_pmd_cpu_mask
  Hexadecimal mask of CPUs to run DPDK Poll-mode drivers
• compute_ovs_dpdk_socket_mem
  Set of amount HugePages in Megabytes to be used by OVS-DPDK daemon taken for each NUMA node. Set size is equal to NUMA nodes count, elements are divided by comma
• compute_ovs_dpdk_lcore_mask
  Hexadecimal mask of DPDK lcore parameter used to run DPDK processes
• compute_ovs_memory_channels
  Number of memory channels to be used.

Example

```yaml
compute_dpdk_driver: uio
compute_ovs_pmd_cpu_mask: "0x6"
compute_ovs_dpdk_socket_mem: "1024"
compute_ovs_dpdk_lcore_mask: "0x400"
compute_ovs_memory_channels: "2"
```

5. Optionally, map the port RX queues to specific CPU cores.

Configuring port queue pinning manually may help to achieve maximum network performance through matching the ports that run specific workloads with specific CPU cores. Each port can process a certain number of Transmit and Receive (RX/TX) operations, therefore it is up to the Network Administrator to decide on the most efficient port mapping. Keeping a constant polling rate on some performance critical ports is essential in achieving best possible performance.

Example

```yaml
dpk0:
  ...
  pmd_rxq_affinity: "0:1,1:2"
```

The example above illustrates pinning of the queue 0 to core 1 and pinning of the queue 1 to core 2, where cores are taken in accordance with pmd_cpu_mask.

6. Specify the MAC address and in some cases PCI for every node.

Example

```yaml
openstack_compute_node02:
  name: ${_param:openstack_compute_node02_hostname}
  domain: ${_param:cluster_domain}
  classes:
    - cluster.${_param:cluster_name}.openstack.compute
  params:
    salt_master_host: ${_param:reclass_config_master}
```
7. If the VXLAN neutron tenant type is selected, set the local IP address on br-prv for VXLAN tunnel termination:

```yaml
linux_system_codename: xenial
dpk0_name: enp5s0f1
dpk1_name: enp5s0f2
dpk0_pci: "0000:05:00.1"
dpk1_pci: "0000:05:00.2"
```

```yaml
... - system.neutron.compute.nfv.dpdk
...
parameters:
  linux:
    network:
      interfaces:
        ...
        # other interface setup
        ...
        br-prv:
          enabled: true
          type: dpdk_ovs_bridge
          address: ${_param:tenant_address}
          netmask: 255.255.255.0
```

8. Select from the following options:

- If you are performing the initial deployment of your environment, proceed with further environment configurations.
- If you are making changes to an existing environment, re-run salt configuration on the Salt Master node:

```
salt "cmp*" state.sls linux.network,neutron
```

Note
For the changes to take effect, servers require a reboot.

9. If you need to set different values for each compute node, define them in `cluster.<NAME>.infra.config`.

Example

```
openstack_compute_node02:
  name: ${_param:openstack_compute_node02_hostname}
```
Enable OpenContrail DPDK

OpenContrail v3.2 uses DPDK libraries v2.1.

Warning
Before you proceed with the DPDK enabling, verify that you have performed the following procedures:

1. Enable Huge Pages
2. Configure NUMA and CPU pinning architecture

A workload running on DPDK vRouter does not provide better pps if an application is not DPDK aware. The performance result is the same as for kernel vRouter.

To enable the OpenContrail DPDK pinning:

1. Verify your NUMA nodes on the host operating system to see what vCPUs are available. For example:

   ```
   lscpu | grep NUMA
   NUMA node(s): 1
   NUMA node0 CPU(s): 0-11
   ```

2. Include the class to cluster.<name>.openstack.compute and configure the vhost0 interface:

   • For a single interface in DPDK:

   ```
   ...
   - system.opencontrail.compute.dpdk
   ```
parameters:
   linux:
      network:
         interfaces:
            ...
            # other interface setup
            ...
            vhost0:
               enabled: true
               type: eth
               mtu: 9000
               address: ${_param:single_address}
               netmask: 255.255.255.0
               name_servers:
                  - 8.8.8.8
                  - 8.8.4.4

3. Set the parameters in cluster.<name>.openstack.init on all compute nodes:
   
   • compute_vrouter_taskset
     Hexadecimal mask of CPUs used for DPDK-vRouter processes
   
   • compute_vrouter_socket_mem
     Set of amount HugePages in Megabytes to be used by vRouter-DPDK taken for each
     NUMA node. Set size is equal to NUMA nodes count, elements are divided by
     comma
   
   • compute_vrouter_dpdk_pci
     PCI of a DPDK NIC. In case of BOND there must be 0000:00:00.0

4. Calculate the hexadecimal mask. To enhance vRouter with DPDK technology, several
isolated host CPUs should be used for such DPDK processes as statistics, queue
management, memory management, and poll-mode drivers. To perform this, you need to
configure the hexadecimal mask of CPUs to be consumed by vRouter-DPDK.

   The way to calculate the hexadecimal mask is simple as a set of CPUs corresponds to the
bits sequence size of CPUs number. 0 on i-th place in this sequence means that CPU number
i will not be taken for usage, and 1 has the opposite meaning. Simple translation of
binary-to-hexadecimal based on bit sequence of size 24 is illustrated below (vRouter is
bound to 4 cores: 14,13,2,1.)

5. Pass the hexadecimal mask to vRouter-DPDK command line using the following parameters.
For example:

   compute_vrouter_taskset: "-c 1,2" # or hexadecimal 0x6
   compute_vrouter_socket_mem: "1024" # or "1024,1024" for 2 NUMA nodes
6. Specify the MAC address and in some cases PCI for every node.
   
   Example

   ```
   openstack_compute_node02:
   name: ${_param:openstack_compute_node02_hostname}
   domain: ${_param:cluster_domain}
   classes:
     - cluster.${_param:cluster_name}.openstack.compute
   params:
     salt_master_host: ${_param:reclass_config_master}
     linux_system_codename: trusty
     compute_vrouter_dpdk_mac_address: 00:1b:21:87:21:99
     compute_vrouter_dpdk_pci: '"0000:05:00.1"'
     primary_first_nic: enp5s0f1 # NIC for vRouter bind
   ```

7. Select from the following options:

   • If you are performing the initial deployment of your environment, proceed with the
     further environment configurations.
   
   • If you are making changes to an existing environment, re-run salt configuration on the
     Salt Master node:

   ```
   salt "cmp*" state.sls opencontrail
   ```

   Note
   For the changes to take effect, servers require a reboot.

8. If you need to set different values for each compute node, define them in
   cluster.<NAME>.infra.config.

   Example

   ```
   openstack_compute_node02:
   name: ${_param:openstack_compute_node02_hostname}
   domain: ${_param:cluster_domain}
   classes:
     - cluster.${_param:cluster_name}.openstack.compute
   params:
     salt_master_host: ${_param:reclass_config_master}
     linux_system_codename: trusty
     compute_vrouter_dpdk_mac_address: 00:1b:21:87:21:99
     compute_vrouter_dpdk_pci: '"0000:05:00.1"'
     compute_vrouter_taskset: "-c 1,2"
   ```
compute_vrouter_socket_mem: "1024"
primary_first_nic: enp5s0f1 # NIC for vRouter bind

Enable SR-IOV

Single Root I/O Virtualization (SR-IOV) is an I/O virtualization technology that allows a single PCIe device to appear as multiple PCIe devices. This helps to optimize the device performance and capacity, as well as hardware costs.

Prerequisites

If you want to use the SR-IOV feature with OpenContrail or Neutron OVS, your environment must meet the following prerequisites:

- Intel Virtualization Technology for Directed I/O (VT-d) and Active State Power Management (ASPM) must be supported and enabled in BIOS
- Physical NIC with Virtual Function (VF) driver installed

Enable ASPM (Active State Power Management) of PCI Devices in BIOS. If required, upgrade BIOS to see ASPM option.

Enable generic SR-IOV configuration

The following procedure is common for both OpenVSwitch and OpenContrail. SR-IOV can be enabled before or after installation on the MCP cluster model level.

To enable SR-IOV:

1. Include the class to cluster.<NAME>.openstack.compute:

   ```
   - system.neutron.compute.nfv.sriov
   ```

   **Note**
   By default, the metadata model contains configuration for 1 NIC dedicated for SR-IOV.

2. Set the following parameters:

   - **sriov_nic01_device_name**
     Name of the interface, where the Virtual Functions are enabled
   - **sriov_nic01_numvfs**
     Number of Virtual Functions
   - **sriov_nic01_physical_network**
     Default is physnet1, label for the physical network the interface belongs to
   - **sriov_unsafe_interrupts**
Default is False, needs to be set to True if your hardware platform does not support interrupt remapping.

For most deployments with 1 NIC for SR-IOV, we recommend the following configuration in `cluster.<name>.openstack.init` on all compute nodes:

```yaml
sriov_nic01_device_name: eth1
sriov_nic01_numvfs: 7
sriov_nic01_physical_network: physnet3
```

3. If you need to set different values for each compute node, specify them in `cluster.<name>.infra.config`.

Example

```yaml
openstack_compute_node02:
  name: ${_param:openstack_compute_node02_hostname}
  domain: ${_param:cluster_domain}
  classes:
    - cluster.${_param:cluster_name}.openstack.compute
  params:
    salt_master_host: ${_param:reclass_config_master}
    linux_system_codename: xenial
    sriov_nic01_device_name: eth1
    sriov_nic01_numvfs: 7
    sriov_nic01_physical_network: physnet3
```

4. If your hardware does not support interrupt remapping, set the following parameter:

```yaml
sriov_unsafe_interrupts: True
```

5. If you need more than one NIC on a compute node, set the following parameters in `cluster.<NAME>.openstack.compute`.

Example

```yaml
...'
  nova:
    compute:
      sriov:
        sriov_nic01:
          devname: eth1
          physical_network: physnet3
        sriov_nic02:
          devname: eth2
          physical_network: physnet4
        sriov_nic03:
          devname: eth3
          physical_network: physnet5
        sriov_nic04:
```
```
devname: eth4
  physical_network: physnet6
linux:
  system:
    sriov: True
    unsafe_interrupts: False
  sysfs:
    sriov_numvfs:
      class/net/eth1/device/sriov_numvfs: 7
      class/net/eth2/device/sriov_numvfs: 15
      class/net/eth3/device/sriov_numvfs: 15
      class/net/eth4/device/sriov_numvfs: 7
```

6. Select from the following options:

- If you are performing the initial deployment your environment, proceed with the further environment configurations.
- If you are making changes to an existing environment, re-run the salt configuration on the Salt Master node:

```
salt "cmp*" state.sls linux,nova
```

**Configure SR-IOV with OpenContrail**

Since OpenContrail does not use Neutron SR-IOV agents, it does not require any special changes on the Neutron side. Port configuration can be done through the Neutron APIs or the OpenContrail UI.

**Configure SR-IOV with OpenVSwitch**

Neutron OVS requires enabling of the sriovnicsswitch mechanism driver on the Neutron server side and the neutron-sriov-nic-agent running on each compute node with this feature enabled.

To configure SR-IOV with OpenVSwitch:

1. Include the class to cluster.<NAME>.openstack.compute:

   ```
   - system.neutron.compute.nfv.sriov
   ```

   **Note**
   By default, the metadata model contains configuration for 1 NIC dedicated for SR-IOV.

2. Include the class to cluster.<NAME>.openstack.control:
3. If you need more than 1 NIC, extend the previous configuration by extra Neutron cluster.<NAME>.openstack.compute.

Example

...  
neutron:
  compute:
    backend:
      sriov:
        sriov_nic01:  
          devname: eth1
          physical_network: physnet3
        sriov_nic02:  
          devname: eth2
          physical_network: physnet4
        sriov_nic03:  
          devname: eth3
          physical_network: physnet5
        sriov_nic04:  
          devname: eth4
          physical_network: physnet6

Create instances with SR-IOV ports

To enable the SR-IOV support, you must create virtual instances with SR-IOV ports.

To create virtual instances with SR-IOV ports:

1. Create a network and a subnet with a segmentation ID. For example:

   neutron net-create --provider:physical_network=physnet3 \\  
   --provider:segmentation_id=100 net04  
neutron subnet-create net04 a.b.c.d/netmask

2. Request the ID of the Neutron network where you want the SR-IOV port to be created. For example:

   net_id=`neutron net-show net04 | grep " \ id\ " | awk '{ print $4 }'`

3. Create an SR-IOV port with one of the available VNIC driver types that are direct, normal, direct-physical, and macvtap:

   port_id=`neutron port-create $net_id --name sriov_port \  
   --binding:vnic_type direct | grep " \ id\ " | awk '{ print $4 }'`

4. Create a virtual instance with the SR-IOV port created in step 3:
nova boot --flavor m1.large --image ubuntu_14.04 --nic port-id=$port_id test-sriov

See also
Using SR-IOV functionality in the official OpenStack documentation

See also
Enable Multiqueue
Enable Huge Pages

Huge Pages is a technology that supports 2MB and 1GB size memory pages. Huge Pages reduces time to access data stored in the memory by using bigger memory pages, which leads to fewer page entries to look up by CPU when choosing a page associated with a current process. Use of Huge Pages is beneficial in operations and processes that require large amount of memory.

Warning
Verify that CPU supports HugePages before you proceed.

Enable the Huge Pages support

This section explains how to configure the support for the Huge Pages feature in your MCP deployment.

To enable Huge Pages:

1. Log in to the host machine.
2. To verify that CPU supports Huge Pages, analyze the system response of the following command:

   cat /proc/cpuinfo

   In the system output, search for the parameters:
   • PSE - support of 2MB hugepages
   • PDPE1GB - support of 1GB hugepages
3. Include the class in cluster.<name>.openstack.compute:

   - system.nova.compute.nfv.hugepages

4. Set the parameters in cluster.<name>.openstack.init on all compute nodes:

   compute_hugepages_size: 1G # or 2M
   compute_hugepages_count: 40
   compute_hugepages_mount: /mnt/hugepages_1G # or /mnt/hugepages_2M

5. Select from the following options:
   • If you are performing the initial deployment your environment, proceed with the further environment configurations.
   • If you are making changes to an existing environment, re-run the salt configuration on the Salt Master node:
salt "cmp*" state.sls linux, nova

6. Reboot the affected servers.

7. If you need to set different values for each compute node, define them in cluster.<name>.infra.config for each node.

Example:

```yaml
openstack_compute_node02:
  name: ${_param:openstack_compute_node02_hostname}
  domain: ${_param:cluster_domain}
  classes:
    - cluster.${_param:cluster_name}.openstack.compute
  params:
    salt_master_host: ${_param:reclass_config_master}
    linux_system_codename: xenial
    compute_hugepages_size: 1G # or 2M
    compute_hugepages_count: 40
    compute_hugepages_mount: /mnt/hugepages_1G # or /mnt/hugepages_2M
```

See also

Boot a virtual machine with Huge Pages

**Boot a virtual machine with Huge Pages**

This section explains how to boot a VM with Huge Pages.

To boot a virtual machine with Huge Pages:

1. Create a new flavor or use an existing one to use with Huge Pages. To create a new image flavor:

   ```
   . openrc admin admin
   nova flavor-create huge 999 1024 4 1
   ```

2. Add the size of huge pages to the image flavor:

   ```
   nova flavor-key huge set hw:mem_page_size=2048
   ```

3. Verify the image flavor exists:

   ```
   nova flavor-show huge
   ```

   Example of system response
Create a new image or use an existing image. You need an Ubuntu image and the default Cirros image.

To create a new Ubuntu image:

```
$ glance --os-image-api-version 1 image-create --name ubuntu \
    --location https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img \
    --disk-format qcow2 --container-format bare
```

5. Boot a new instance using the created flavor:
```
$ nova boot --flavor huge --image ubuntu inst1
```

6. Verify that the new VM uses 512 huge pages:
```
$ grep Huge /proc/meminfo
```

Example of system response

```
AnonHugePages:  1138688 kB
HugePages_Total: 1024
HugePages_Free:  512
HugePages_Rsvd:  0
HugePages_Surp:  0
Hugepagesize:   2048 kB
```
**Configure NUMA and CPU pinning architecture**

NUMA and CPU pinning is a shared memory architecture that describes the placement of main memory modules on processors in a multiprocessor system. You can leverage NUMA when you have data strongly associated with certain tasks or users. In such case, CPU can use its local memory module to access data reducing access time.

NUMA usage is beneficial on particular workloads, for example, on configurations where data is often associated with certain tasks or users.

**Enable NUMA and CPU pinning**

Before you proceed with enabling DPDK in your deployment, the NUMA and CPU pinning enablement is required.

To enable NUMA and CPU pinning:

1. Verify your NUMA nodes on the host operating system:

   ```sh
   lscpu | grep NUMA
   ```

   **Example of system response**

   ```
   NUMA node(s):          1
   NUMA node0 CPU(s):     0-11
   ```

2. Include the class to cluster.<NAME>.openstack.compute:

   ```
   - system.nova.compute.nfv.cpu_pinning
   ```

3. Set the parameters in cluster.<name>.openstack.init on all compute nodes:

   - `compute_kernel_isolcpu`
     Set of host CPUs to be isolated from system. Kernel will not assign internal processes on this set of CPUs. This parameter is configured in `grub`
   - `nova_cpu_pinning`
     Subset of CPUs isolated on previous step. This parameter is used by Nova to run VMs only on isolated CPUs with dedicated pinning. Nova vCPU pinning set is configured in the `nova.conf` file after system isolates appropriate CPUs

   **Example**

   ```
   nova_cpu_pinning: "1,2,3,4,5,7,8,9,10,11"
   compute_kernel_isolcpu: ${_param:nova_cpu_pinning}
   ```

4. Select from the following options:

   - If you are performing the initial deployment, proceed with the further environment configurations.
• If you are making changes to an existing environment, re-run the salt configuration on the Salt Master node:

```bash
classic "cmp*" state.sls linux,nova
```

Note
To take effect, servers require a reboot.

5. If you need to set different values for each compute node, define them in `cluster.<name>.infra.config`.

Example

```bash
openstack_compute_node02:
  name: ${_param:openstack_compute_node02_hostname}
  domain: ${_param:cluster_domain}
  classes:
    - cluster.${_param:cluster_name}.openstack.compute
  params:
    salt_master_host: ${_param:reclass_config_master}
    linux_system_codename: xenial
    nova_cpu_pinning: "1,2,3,4,5,7,8,9,10,11"
    compute_kernel_isolcpu: "1,2,3,4,5,7,8,9,10,11"
```

Boot a VM with two NUMA nodes

This example demonstrates booting a VM with two NUMA nodes.

To boot VM with two NUMA nodes:

1. Create a new flavor or use an existing one to use with NUMA. To create a new flavor, run:

```bash
. openrc admin admin
nova flavor-create m1.numa 999 1024 5 4
```

2. Add numa_nodes to the flavor.

Note
vCPUs and RAM will be divided equally between the NUMA nodes.
nova flavor-key m1.numa set hw:numa_nodes=2
nova flavor-show m1.numa

Example of system response:

+----------------------------+------------------------+
| Property                   | Value                  |
+----------------------------+------------------------+
| OS-FLV-DISABLED:disabled   | False                  |
| OS-FLV-EXT-DATA:ephemeral  | 0                      |
| disk                       | 5                      |
| extra_specs                | {"hw:numa_nodes": "2"} |
| id                         | 999                    |
| name                       | m1.numa                |
| os-flavor-access:is_public | True                   |
| ram                        | 1024                   |
| rxtx_factor                | 1.0                    |
| swap                       |                        |
| vcpus                      | 4                      |
+----------------------------+------------------------+

3. Create a new image or use an existing image.

Note
You need an Ubuntu image and the default Cirros image.

To create a new Ubuntu image:

```
    glance --os-image-api-version 1 image-create --name ubuntu
            --location https://cloud-images.ubuntu.com/trusty/current/
            trusty-server-cloudimg-amd64-disk1.img
            --disk-format qcow2 --container-format bare
```

4. To enable SSH connections:

1. Add a new rule to the security group:

   nova secgroup-add-rule default tcp 22 22 0.0.0.0/0

2. Create a new SSH key pair or use the existing key pair. To create a new ssh key pair:

   ssh-keygen

3. Add the key pair to Nova:
nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp

5. Verify free memory before you boot the VM:

```
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 3856 MB
node 0 free: 718 MB
node 1 cpus: 2 3
node 1 size: 3937 MB
node 1 free: 337 MB
node distances:
  node 0 1
  0: 10 20
  1: 20 10
```

6. Boot a new instance using the created flavor:

```
nova boot --flavor m1.numa --image ubuntu --key-name my_kp inst1
```

7. Verify if free memory has been changed after booting the VM:

```
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 3856 MB
node 0 free: 293 MB  # was 718 MB
node 1 cpus: 2 3
node 1 size: 3937 MB
node 1 free: 81 MB   # was 337 MB
node distances:
  node 0 1
  0: 10 20
  1: 20 10
```

8. Retrieve the instance’s IP:

```
nova show inst1 | awk '/network/ {print $5}'
```
Example of system response:

```
10.0.0.2
```

9. Connect to the VM using SSH:

```
ssh ubuntu@10.0.0.2
```

10. Install numactl:

```
sudo apt-get install numactl
```

11. Verify the NUMA topology on the VM:

```
umactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 489 MB
node 0 free: 393 MB
node 1 cpus: 2 3
node 1 size: 503 MB
node 1 free: 323 MB
node distances:
  node   0   1
    0:  10  20
    1:  20  10
```

---

**Boot a VM with CPU and memory pinning**

This example demonstrates booting VM with CPU and memory pinning.

To boot VM with CPU and memory pinning:

1. Create a new flavor with specific division of vCPUs and RAM between the NUMA nodes:

```
. openrc admin admin
nova flavor-create m1.numa_2 9992 1024 5 4
```

2. Add numa_nodes and other specific options to the flavor:

```
nova flavor-key m1.numa_2 set hw:numa_nodes=2 hw:numa_cpus.0=0,2 \ hw:numa_cpus.1=1,3 hw:numa_mem.0=324 hw:numa_mem.1=700
nova flavor-show m1.numa_2 | grep extra
```

Example of system response:
3. Create a new image or use an existing image.

Note
You need an Ubuntu image or the default Cirros image.

To create a new Ubuntu image:

```bash
glance --os-image-api-version 1 image-create --name ubuntu \
    --location https://cloud-images.ubuntu.com/trusty/current/
    trusty-server-cloudimg-amd64-disk1.img \
    --disk-format qcow2 --container-format bare
```

4. To enable SSH connections:

1. Add a new rule to the security group:

```bash
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

2. Create a new SSH key pair or use the existing key pair. To create a new ssh key pair, run:

```bash
ssh-keygen
```

3. Add the key pair to Nova:

```bash
nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp
```

5. Boot a new instance using the created flavor:

```bash
nova boot --flavor m1.numa_2 --image ubuntu --key-name my_kp inst2
```

6. Verify if free memory has been changed after booting the VM:

```bash
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 2
node 0 size: 3856 MB
node 0 free: 293 MB           # was 718 MB
```
7. Retrieve the instance’s IP:

    nova show inst2 | awk '/network/ {print $5}'

Example of system response:

    10.0.0.3

8. Connect to the VM using SSH:

    ssh ubuntu@10.0.0.3

9. Install numactl:

    sudo apt-get install numactl

10. Verify the NUMA topology on the VM:

    numactl -H

Example of system response:

    available: 2 nodes (0-1)
    node 0 cpus: 0 2
    node 0 size: 303 MB
    node 0 free: 92 MB
    node 1 cpus: 3 1
    node 1 size: 689 MB
    node 1 free: 629 MB
    node distances:
    node 0 1
    0: 10 20
    1: 20 10

You can see that the NUMA topology has two NUMA nodes. Total RAM size is about 1GB:
• node-0 CPUs are 0 and 2
• node-1 CPUs are 1 and 3, node-1 RAM is about 324 MB
• node-2 RAM is about 700 as specified in the m1.numa_2 flavor
Enable Multiqueue

The MCP Multiqueue enables the scaling of packet sending/receiving processing to the number of available vCPUs of a guest by using multiple queues. The feature includes:

- **Multiqueue for DPDK-based vroute**
  
  Is supported by OpenVSwitch only. Underlay configuration for OVS is a part of DPDK interfaces and is defined by the `n_rxq` parameter. For example:

```plaintext
...  
- system.neutron.compute.nfv.dpdk  
...  
parameters:  
  linux:  
    network:  
      interfaces:  
        ...  
        # other interface setup  
        ...  
        dpdk0:  
          name: ${_param:dpdk0_name}  
          pci: ${_param:dpdk0_pci}  
          driver: igb_uio  
          bond: dpdkbond1  
          enabled: true  
          type: dpdk_ovs_port  
          n_rxq: 2  
        dpdk1:  
          name: ${_param:dpdk1_name}  
          pci: ${_param:dpdk1_pci}  
          driver: igb_uio  
          bond: dpdkbond1  
          enabled: true  
          type: dpdk_ovs_port  
          n_rxq: 2
```

- **Multiqueue Virtio**

  Is supported by OpenContrail and OVS

**Provision a VM with Multiqueue**

To provision a VM with Multiqueue:

1. Set the image metadata property with the Multiqueue enabled:

   ```bash
   nova image-meta <IMAGE_NAME> set hw_vif_multiqueue_enabled="true"
   ```

2. After the VM is spawned, use the following command on the virtio interface in the guest to enable multiple queues inside the VM:
ethtool -L <INTERFACE_NAME> combined <#queues>
Configure LDAP integration with MCP

This section describes how to integrate your LDAP server with Keystone and a host operating system in MCP. This configuration is not enabled by default and, therefore, requires manual modifications in your cluster model.

Configure LDAP with Keystone server

To configure LDAP integration with Keystone server in MCP, you must create a separate file for this definition in your cluster model. In this section, the ldap.yml file is used as an example. You must also set up the rights mapping for users and groups. If required, you can also specify filtering.

To configure LDAP with Keystone server:

1. In your Git project repository, open the cluster/<cluster_name>/openstack/ directory of your cluster model.
2. In this directory, create the ldap.yml file.
3. Create a configuration for the LDAP integration in this file.

Example:

```yaml
parameters:
  keystone:
    server:
      service_name: apache2
      domain:
        <DOMAIN NAME>:
          description: ""
          backend: ldap
          identity:
            backend: ldap
            assignment:
              backend: sql
          ldap:
            url: ldap://<LDAP ADDRESS>
            bind_user: CN=<UserName>,OU=<OU-name>,DC=<DomainName>,DC=<DomainExtension>
            query_scope: sub
            page_size: 1000
            password: <LDAP PASSWORD>
            suffix: DC=<DomainName>,DC=<DomainExtension>
            user_tree_dn: DC=<DomainName>,DC=<DomainExtension>
            group_tree_dn: DC=<DomainName>,DC=<DomainExtension>
            user_objectclass: person
            user_id_attribute: sAMAccountName
            user_name_attribute: sAMAccountName
            user_pass_attribute: userPassword
            user_enabled_attribute: userAccountControl
            user_mail_attribute: mail
```
4. In `cluster/<cluster_name>/openstack/control.yml`, include the previously created class to the bottom of the classes section:

```yaml
classes:
  ...
  cluster.<cluster_name>.openstack.ldap
parameters:
  ...
```

5. Add parameters for Horizon to `cluster/<cluster_name>/openstack/proxy.yml`:

```yaml
parameters:
  horizon:
    server:
      multidomain: true
```

6. Enforce the Keystone update using the Jenkins Deploy - update service(s) config pipeline or directly using Salt:

```bash
salt -C 'I@keystone:server and *01*' state.sls keystone
salt -C 'I@keystone:server and not *01*' state.sls keystone
salt -C 'I@horizon:server' state.sls horizon
```

7. Verify the LDAP integration:

```bash
source /root/keystonercv3
openstack user list --domain <your_domain>
```
8. Grant the admin role to a specific user:
   
   1. Obtain the user ID:
      
      ```bash
      openstack user list --domain <your_domain> | grep <user_name>
      | <user_id> | <user_name> |
      ```
      
      2. Set the admin role:
      
      ```bash
      openstack role add --user <user_id> admin --domain <your_domain>
      ```

   **Configure LDAP with host OS**

   To configure the pluggable authentication module (PAM) on a host operating system to support
   LDAP authentication in MCP, you must create a separate file for this definition in your cluster
   model and add it to all the nodes where you want to enable this authentication method.

   In this section, the ldap.yml file is used as an example.

   To enable PAM authentication:

   1. Open the Git project repository with your cluster model.
   2. Create the cluster/<cluster_name>/infra/auth/ldap.yml file.
   3. Create a configuration for your LDAP server in this file.

   Example:

   ```yaml
   parameters:
   linux:
    system:
     auth:
      enabled: true
    ldap:
     enabled: true
     binddn: CN=<UserName>,OU=<OU-name>,DC=<DomainName>,DC=<DomainExtension>
     bindpw: <Password>
     uri: ldap://<LDAP URL>
     base: DC=<DomainName>,DC=<DomainExtension>
     ldap_version: 3
     pagesize: 1000
     referrals: "off"
    ##You can also setup grouping, mapping, and filtering using these parameters.
    filter:
     passwd: (&(&(objectClass=person)(uidNumber=*)(unixHomeDirectory=*))(unixHomeDirectory=*))
     shadow: (&(&(objectClass=person)(uidNumber=*)(unixHomeDirectory=*)))(unixHomeDirectory=*)
     group: (&(objectClass=group)(gidNumber=*))
    map:
     passwd:
      uid: sAMAccountName
      homeDirectory: unixHomeDirectory
   ```
4. In `cluster/<cluster_name>/openstack/cluster.yml`, include the previously created class to the bottom of the classes section:

```yaml
classes:
  ...
  cluster.<cluster_name>.infra.auth.ldap
  cluster.<cluster_name>
parameters:
  ...
```

5. Enforce the `linux.system` update using the Jenkins Deploy - update service(s) config pipeline or directly using Salt:

```bash
salt '<target_node>*' state.sls linux.system
```
Deployment customizations guidelines

This section contains instructions that do not belong to a specific part of the deployment workflow. Otherwise speaking, the procedures included in this section are optional and contain only customizations guidelines that can be skipped if you perform the default MCP deployment.

The procedures below are referenced from the sections where they can merge into the general deployment workflow. You should not perform these procedures as standalone instructions. And always remember to continue the deployment exactly from the step that referenced you to this section.

Generate configuration drives manually

You may need to manually generate the configuration drives for an automated MCP deployment after you customize their content to meet specific requirements of your deployment. This section describes how to generate the configuration drives using the create-config-drive script.

To generate a configuration drive for the cfg01 VM:

1. Download the create-config-drive script for generating the configuration drive:

   ```
   export MCP_VERSION="master"
   wget -O /root/create-config-drive \
   https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/
   ${MCP_VERSION}/config-drive/create_config_drive.sh
   chmod +x /root/create-config-drive
   ```

2. Download the Salt Master configuration script:

   ```
   wget -O /root/user_data.sh \
   https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/
   ${MCP_VERSION}/config-drive/master_config.sh
   ```

3. In user_data.sh, modify the lines that start with export to fit your environment. If you use local (aptly) repositories, choose the following parameters to point to your local repositories address on port 8088:

   - MCP_VERSION
   - PIPELINES_FROM_ISO=false
   - PIPELINE_REPO_URL
   - MCP_SALT_REPO_KEY
   - MCP_SALT_REPO_URL

4. Choose from the following options:

   - If you do not use local repositories:

     1. Clone the master branches of the mk-pipelines and pipeline-library Git repositories:

        ```
        git clone https://github.com/Mirantis/mk-pipelines.git /root/mk-pipelines
        git clone https://github.com/Mirantis/pipeline-library.git /root/pipeline-library
        ```
If you require a specific release version, for example, 2018.1, run the following command:

```
export version="tags/2018.1"
git clone https://github.com/Mirantis/mk-pipelines.git /root/mk-pipelines
git clone https://github.com/Mirantis/pipeline-library.git /root/pipeline-library
pushd /root/mk-pipelines
git checkout ${version} ; popd
pushd /root/pipeline-library
git checkout ${version} ; popd
```

2. Put your Reclass model that contains the classes/cluster, classes/system, nodes, .git, and .gitmodules directories in /root/model.

3. Run the configuration drive generator script:

```
/root/create-config-drive -u /root/user_data.sh -h cfg01 \ 
--model /root/model --mk-pipelines /root/mk-pipelines \ 
--pipeline-library /root/pipeline-library cfg01-config.iso
```

The generated configuration drive becomes available as the cfg01-config.iso file.

- If you use local repositories:
  1. Put your Reclass model that contains the classes/cluster, classes/system, nodes, .git, and .gitmodules directories in /root/model.
  2. Run the configuration drive generator script:

```
/root/create-config-drive -u /root/user_data.sh -h cfg01 \ 
--model /root/model cfg01-config.iso
```

The generated configuration drive becomes available as the cfg01-config.iso file.

To generate a configuration drive for the APT VM:

1. Download the create-config-drive script for generating the configuration drive:

```
wget -O /root/create-config-drive \ 
https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/$MCP_VERSION/config-drive/create_config_drive.sh
chmod +x /root/create-config-drive
```

2. Download the mirror configuration script:

```
wget -O /root/user_data.sh \ 
https://raw.githubusercontent.com/Mirantis/mcp-common-scripts/$MCP_VERSION/config-drive/mirror_config.sh
```

3. In user_data.sh, modify the lines that start with export to fit your environment.

4. Run the configuration drive generator script:

```
/root/create-config-drive -u /root/user_data.sh -h apt apt-config.iso
```
The generated configuration drive should now be available as the apt-config.iso file.

To generate a simple configuration drive for any cloud-image:

1. Install the cloud-image-utils tool:

   ```
   apt-get install -y cloud-image-utils
   ```

2. For example, create a configuration file with the `config-drive-params.yaml` name.

3. In this file, enable the password access for root and Ubuntu users. For example:

   ```yaml
   #cloud-config
debug: True
ssh_pwauth: True
disable_root: false
chpasswd:
  list:
    root:r00tme
    ubuntu:r00tme
expire: False
runcmd:
  - sed -i 's/PermitRootLogin.*/PermitRootLogin yes/g' /etc/ssh/sshd_config
  - sed -i 's/PasswordAuthentication.*/PasswordAuthentication yes/g' /etc/ssh/sshd_config
  - service sshd restart
   ```

4. Create the configuration drive:

   ```
   cloud-localds --hostname testvm --dsmode local mynewconfigdrive.iso config-drive-params.yaml
   ```

Now, you can use mynewconfigdrive.iso with any cloud-image. For example, with the MCP VCP images or any other image that has cloud-init pre-installed.

### Add custom commissioning scripts

Using MAAS, you can extend the default commissioning logic with additional user-defined scripts. Each defined script will be applied to a VM commissioning by default.

For example, to set custom NIC names that are oneXX for a 1 GB Ethernet and tenXX for a 10 GB Ethernet, refer to the following procedures.

In the examples below, the default `00-maas-05-simplify-network-interfaces` script from the salt-formulas-maas package is used. The script is located on the Salt Master node in the `/srv/salt/env/prd/maas/files/commisioning_scripts/` directory.

To automatically add the commissioning script using Salt:

1. Prepare a script for commissioning and save it on the MAAS node. For example, use the default script from the salt-formulas-maas package.

2. Enable automatic importing of the script by defining it in `/srv/salt/reclass/classes/cluster/<CLUSTER_NAME>/infra/maas.yml`. 
Caution!

The commissioning script name is important. If you have several scripts, they will run in the alphanumerical order depending on their name.

3. Run the following command:

```
salt-call -l debug --no-color maas.process_commissioning_scripts
```

Example of system response:

```
...  
local: 
-------- 
errors: 
-------- 
success:  
- 00-maas-05-simplify-network-interfaces
```

The script 00-maas-05-simplify-network-interfaces is uploaded to MAAS from the /etc/maas/files/commissioning_scripts/ directory.

After the importing is done, proceed with commissioning depending on your use case as described in Provision physical nodes using MAAS.

To manually add the commissioning script using the MAAS web UI:

1. Log in to the MAAS web UI through salt_master_management_address/MAAS with the following credentials:
   - Username: mirantis
   - Password: r00tme
2. Go to the Settings tab.
3. Scroll to Commissioning scripts.
4. Click Upload script to chose a file for uploading. For example, use the default script from the salt-formulas-maas package.
Caution!

The commissioning script name is important. If you have several scripts, they will run in the alphanumeric order depending on their name.

After the importing is done, proceed with commissioning depending on your use case as described in Provision physical nodes using MAAS.

**Configure PXE booting over UEFI**

This section explains how to configure the Preboot Execution Environment (PXE) to boot a hardware server from the network over Unified Extensible Firmware Interface (UEFI), which details the interface between the platform firmware and the operating system at boot time.

During the manual MCP infrastructure deployment, the PXE boot takes place when you add new physical servers that are not yet loaded with an operating system to your deployment. The Foundation node is installed with all the necessary software from a virtual machine image. All other hardware servers are installed remotely by MAAS using PXE boot. If required, you can configure a server to boot from network over UEFI.

To configure the UEFI network boot:

1. Configure the server in BIOS to use UEFI on boot time:
   1. On the Advanced tab, set the Launch Storage OpROM policy option to UEFI only:

   ![Advanced tab configuration](image)

   2. On the Boot tab, specify the UEFI network connection as the first boot device. For example:
2. During commissioning through MAAS, verify that the server uses UEFI. For example:

```
Note
If you perform standard PXE boot, the MAAS commissioning process will not recognize
UEFI.
```

See also
- Provision physical nodes using MAAS

### Add a custom disk layout per node in the MCP model

In MAAS, you can define the disk layout, either flat or Logical Volume Manager (LVM), as well as the partitioning schema per server. This section describes how to define these parameters in the MAAS section of the MCP model. The disk configuration applies during the deployment process. If you want to define the disk configuration after deployment, you can use salt-formula-linux that
also has a capability to set up LVM partitioning. But the whole definition for each Volume Group must be either in the maas or linux section, since the linux state cannot override or extend an existing Volume Group created using MAAS but can create a new one.

Caution!

You can define the disk configuration in the model before the deployment starts. But be aware that enforcing of this configuration to MAAS using the salt state must be done after servers are commissioned and before they are deployed. Basically, maas.machines.storage works only if a server is in the READY state.

Caution!

The maas.machines.storage state overlaps with the linux.storage state. Therefore, we recommend using only one of them. If your deployment requires both, be aware that:

- The linux.storage configuration must match the maas.machines.storage configuration.
- MAAS may use an inexplicit mapping. For example, the following MAAS configuration will create an inexplicit mapping to sda1. And this specific sda1 device must be defined in the linux.storage configuration.

```yaml
maas:
  ...
  vg0:
    type: lvm
    devices:
      - sda
  ...
```

You can use several options to design the disk layout in a deployment depending on specific use cases. This section includes three most common examples that can be combined to get your desired configuration.

To define a different disk layout with custom parameters

The default layouts used by MAAS are flat and Logical Volume Manager (LVM). Flat layout creates a root partition on the first disk of a server. LVM creates a Volume Group on this partition with one volume per root. By default, in both types of disk layout, the entire space on the first disk is used. If you want to change this behavior, redefine the disk layout parameters.

The following examples illustrate a modified configuration of the default values for partition size as well as LVM names for Volume Group and Logical Volume:
• Flat layout:

```yaml
maas:
  region:
    machines:
      server1:
        disk_layout:
          type: flat
          root_size: 10G  # sda disk has more than 10G
          root_device: sda
          bootable_device: sda
```

• LVM layout:

```yaml
maas:
  region:
    machines:
      server1:
        disk_layout:
          type: lvm
          root_size: 20G  # sda disk has more than 20G
          root_device: sda
          bootable_device: sda
          volume_group: vg0
          volume_name: root
          volume_size: 10G  # If not defined, whole root partition is used.
```

Caution!

When defining the disk layout in the model, do not modify the rest of the disk using the MAAS dashboard. Each run of `maas.machines.storage` deletes and recreates the disk configuration of a server. Currently, this state is not idempotent.

To define a custom partitioning schema

To define a more complex configuration for disks, use the disk section under the `disk_layout` parameter.

The following example illustrates how to create partitions on the sda disk and a Volume Group with Logical Volumes on the sdb disk. Be aware that sdb is also defined without any partitioning schema. Therefore, you can enforce no partition to be present on sdb. Also, due to the `volume_group1` dependency on this device, it must be defined with some configuration in the model. In the example below, it has no partitioning schema.

Example of creating partitions and Logical Volumes:
Caution!

The naming convention for partition in MAAS does not allow using custom names. Therefore, key names in YAML for partition are always part1, part2, ..., partN.

To define the software RAID

Using the disk section from the previous example, you can create the software RAID on servers. You can use this device for LVM or you can define a partitioning schema directly on this device.
The following example illustrates how to create raid 1 on sda and sdb with the partitioning schema. In this example, we use flat layout that creates a root partition on sda, but this partition is eventually deleted because sda is defined without any partitioning schema.

Example of creating the software RAID disks:

```yaml
maas:
  region:
    machines:
      server3:
        disk_layout:
          type: flat
          bootable_device: sda
        disk:
          sda:
            type: physical
          sdb:
            type: physical
        md0:
          type: raid
          level: 1
        devices:
          - sda
          - sdb
        partition_schema:
          part1:
            size: 10G
            type: ext4
            mount: '/'
          part2:
            size: 5G
          part3:
            size: 25G
```

To apply changes to MAAS

To enforce the disk configuration on servers in MAAS, run the maas state on a node where the MAAS model is included. Usually, this is the cfg01 node.

```
salt-call state.sls maas.machines.storage
```

Now, proceed with the MCP deployment depending on your use case as described in Provision physical nodes using MAAS.

**Enable NTP authentication**

This section describes how to enable Network Time Protocol (NTP) authentication in a deployment model and apply it to your environment.

To configure authentication for NTP:
1. Log in to the Salt Master node.

2. Create the `classes/cluster/<cluster_name>/infra/ntp_auth.yml` file with the following configuration as an example:

   ```yaml
   ntp:
     client:
       enabled: true
     auth:
       enabled: true
     secrets:
       1:
         secret_type: 'M'
         secret: '<Runrabbitrundigthath>'
         trustedkey: true
       2:
         secret_type: 'M'
         secret: '<Howiwishyouwereherew>'
         trustedkey: true
     stratum:
       primary:
         server: <ntp1.example.com>
         key_id: 1
       secondary:
         server: <ntp2.example.com>
         key_id: 2
   ```

   In the secrets and stratum sections, specify your own keys and strata servers accordingly.

   The key_id parameter for each strata server represents the id of a secret from the secrets section.

   The above configuration example enables authentication for two servers. For a specific use case, see `salt-formula-ntp/README.rst`.

3. In the `classes/cluster/<cluster_name>/infra/init.yml` file, include the following class to distribute the settings across all nodes:

   ```yaml
   classes:
     - cluster.<cluster_name>.infra.ntp_auth
   ```

4. Apply the ntp state on the Salt Master node:

   ```bash
   salt '*' state.sls ntp
   ```

See also

- `ntp-genkeys`
Enable a watchdog

This section describes how to enable a watchdog in your MCP cluster and applies to both existing and new MCP deployments.

The watchdog detects and recovers servers from serious malfunctions which can include hardware faults as well as program errors. While operating normally, the server resets the watchdog preventing it from generating a timeout signal. Otherwise, the watchdog initiates corrective actions to restore the normal operation of a system.

This functionality can be implemented through either a watchdog timer, which is a hardware device, or a software-only softdog driver.

To install and configure the watchdog:

1. Log in to the Salt Master node.
2. In the `classes/cluster/<cluster_name>/init.yml` or `classes/cluster/<cluster_name>/init/init.yml` file of your Reclass model, include the following class:

   ```yaml
   classes:
     - system.watchdog.server
   ```

3. In the `classes/cluster/<cluster_name>/infra/config.yml` file of your Reclass model, add the watchdog server configuration. For example:

   ```yaml
   watchdog:
     server:
       admin: root
       enabled: true
       interval: 1
       log_dir: /var/log/watchdog
       realtime: yes
       timeout: 60
       device: /dev/watchdog

   # Salt Stack will automatically detect the necessary kernel module
   # which needs to be loaded (ex. hpwdt, iTCO_wdt).
   # If the hardware model is not predefined in map.jinja, the default
   # watchdog driver is used: softdog
   # You may specify the kernel module parameters if needed:
   kernel:
     parameter:
       soft_panic: 1
       parameter: value
       parameter_only_without_value: none
   ```

4. Select from the following options:
• If you are performing the initial deployment of your environment, the watchdog service will be installed during the Finalize stage of the Deploy - OpenStack pipeline. See Deploy an OpenStack environment for details.

• If you are enabling the watchdog service in an existing environment, apply the changes to the deployment model to install the service:

```bash
salt '*' state.sls watchdog
```

5. Verify that the watchdog service is enabled in your deployment:

```bash
salt '*' cmd.run "service watchdog status"
```