MCP Deployment Guide

version 1.0

Copyright notice

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:

Revision date	Description
March 30, 2017	1.0 GA

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.

Seealso

- The Minimum hardware requirements section in the MCP Reference Architecture guide

Plan your deployment

The configuration of you MCP installation depends on the individual requirements your OpenStack 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.

The tables below are examples of the deployment planning. The parameters included in these tables are applicable to the manual deployment described in this guide and cover the planned nodes, nodes' names, and the networking schema for the OpenStack environment.

Underlay networks

Description	VLAN ID	CIDR	Gateway	Allocation range	External Access	Comments
IPMI					Accessible by deployment and operations teams	
MaaS NAT / Proxy (temporary)	20	172.17.8.225/32	172.17.8.193	172.17.8.225/32	Temporary IP for external NAT	
PXE	101	172.31.35.0/24	172.31.35.1	.11-243	Pull packages from internet or local mirror	
Control	102	172.31.110.0/24	172.31.110.1	.11-243	Pull packages from internet or local mirror	
Proxy Network	200	172.31.111.0/29	172.31.111.1	.4-.6 (if starting at .0)	Accessible by users/admins	Need 3 addresses
Data Network 1	103	172.31.112.0/24	172.31.112.1	.11-243	Accessible from/to Cloud Gateway Router	
Data Network 2	104	172.31.104.0/24	172.31.104.1	.11-243	Optional	

Hosts details

Hostname	Description	MGMT, PXE, DHCP (VLAN 35)	Control Plane (VLAN 110)	Web/API Proxy (VLAN 111)	Cloud Underlay (DATA 1)(VLAN 112)

		172.31.35.0/24	172.31.110.0/24	172.31.111.0/29	172.31.112.0/24
cfg01	Salt Master	172.31.35.12	172.31.110.12		
mas01	Metal as a Service	172.31.35.13	172.31.110.13		
	OpenStack Controller VIP	DHCP dynamic	172.31.110.20		
ctl01	OpenStack Controller		172.31.110.21		
ctl02	OpenStack Controller		172.31.110.22		
ctl03	OpenStack Controller		172.31.110.23		
	OpenContrail Controller VIP		172.31.110.30		
ntw01	OpenContrail Controller		172.31.110.31		
ntw02	OpenContrail Controller		172.31.110.32		
ntw03	OpenContrail Controller		172.31.110.33		
	OpenContrail Analytics VIP		172.31.110.40		
nal01	OpenContrail Analytics		172.31.110.41		
nal02	OpenContrail Analytics		172.31.110.42		
nal03	OpenContrail Analytics		172.31.110.43		
	RabbitMQ VIP		172.31.110.50		
rmq01	RabbitMQ server		172.31.110.51		
rmq02	RabbitMQ server		172.31.110.52		
rmq03	RabbitMQ server		172.31.110.53		
	MySQL Galera Database VIP		172.31.110.60		

dbs01	MySQL Galera Database		172.31.110.61		
dbs02	MySQL Galera Database		172.31.110.62		
dbs03	MySQL Galera Database		172.31.110.63		
	Ceilometer VIP		172.31.110.70		
mdb01	Ceilometer collectors MongoDB		172.31.110.71		
mdb02	Ceilometer collectors MongoDB		172.31.110.72		
mdb03	Ceilometer collectors MongoDB		172.31.110.73		
prxvip01	Dashboard & API VIP (External)			172.31.111.6	
prx01	Dashboard & API proxy		172.31.110.81	172.31.111.4	
prx02	Dashboard & API proxy		172.31.110.82	172.31.111.5	
	Monitoring VIP		172.31.110.15		
mon01	Monitoring server		172.31.110.16		
mon02	Monitoring server		172.31.110.17		
	Metering VIP		172.31.110.90		
mtr01	Metering Server Graphite		172.31.110.91		
mtr02	Metering Server Graphite		172.31.110.92		
	Logging VIP		172.31.110.95		
log01	Logging Server, Elastic cluster		172.31.110.96		

log02	Logging Server, Elastic cluster		172.31.110.97		
log03	Logging Server, Elastic cluster		172.31.110.98		
bmk01	Benchmark Rally Server		172.31.110.19		
kvm01	KVM node with controllers	172.31.35.11	172.31.110.201		
kvm02	KVM node with controllers	172.31.35.14	172.31.110.202		
kvm03	KVM node with controllers	172.31.35.15	172.31.110.203		
cmp01	Compute Node (KVM)	172.31.35.30	172.31.110.101		172.31.112.11

Create a project repository manually

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.

To create a project repository manually:

1. Initialize your project repository:

```
git init
git remote add origin [YOUR-GIT-REPO-URL]

mkdir -p classes/cluster
mkdir -p nodes

RECLASS_REPO=$PWD
```

2. Initialize submodule:

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

3. Commit and push.
4. Proceed to Create a deployment metadata model.

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 the set of models that combine the already created system objects into different solutions. A 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 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.

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 available internally, for the Mirantis deployment engineers. If you have questions, please contact Mirantis support.

The workflow of a model creation includes the following stages:

1. Definition of 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.

   > Note
   >
   > If you prefer publishing to the project repository, verify that the dedicated repository is configured correctly and Jenkins can access it. See Create a project repository manually for details.

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 overriden afterwards.

> Note
>
> Currently, Cookiecutter can generate models with basic configurations. You may need to manually customize your model after generation to meet specific requirements of your deployment, for example, four interfaces bonding.

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 manually
> - 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 leaved to default DHCP from installation. Setups for the NFV scenarios are not covered and require manual configuration.

Basic deployment parameters

Parameter	Default JSON output	Description
Cluster name	cluster_name: deployment_name	The name of the cluster that will be used as cluster/<cluster_name>/ in the project directory structure
Cluster domain	cluster_domain: deploy-name.local	The name of the domain that will be used as part of the cluster FQDN
Public host	public_host: ${_param:openstack_proxy_address}	The name or IP address of the public endpoint for the deployment
Reclass repository	reclass_repository: https://github.com/Mirantis/mk-lab-salt-model.git	The URL to your project Git repository containing your models
Cookiecutter template URL	cookiecutter_template_url: git@github.com:Mirantis/mk2x-cookiecutter-reclass-model.git	The URL to the Cookiecutter template repository
Cookiecutter template branch	cookiecutter_template_branch: master	The branch of the Cookiecutter template repository to use, master by default
Deployment type	deployment_type: physical	The supported deployment types include: • Physical for the OpenStack platform • Physical and Heat for the Kubernetes platform
Publication method	publication_method: email	The method to obtain the template. Available options include: • Send to the e-mail address • Commit to repository

Services deployment parameters

Parameter	Default JSON output	Description
Platform	platform: openstack_enabled	The platform to generate the model for: • The OpenStack platform supports OpenContrail, StackLight, and CI/CD sub-clusters enablement. If the OpenContrail is not enabled, the model will define OVS as a network engine. • The Kubernetes platform supports StackLight and CI/CD sub-clusters enablement, and presupposes Calico networking.

Networking deployment parameters

Parameter	Default JSON output	Description
DNS Server 01	dns_server01: 8.8.8.8	The IP address of the dns01 server
DNS Server 02	dns_server02: 8.8.4.4	The IP address of the dns02 server
Deploy network subnet	deploy_network_subnet: 10.0.0.0/24	The IP address of the deploy network with the network mask
Deploy network gateway	deploy_network_gateway: 10.0.0.1	The IP gateway address of the deploy network
Control network subnet	control_network_subnet: 10.0.1.0/24	The IP address of the control network with the network mask
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	tenant_network_gateway: 10.0.2.1	The IP gateway address of the tenant network
Control VLAN	control_vlan: '10'	The Control plane VLAN ID
Tenant VLAN	tenant_vlan: '20'	The Data plane VLAN ID

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 networking
- CI/CD
- Repositories

Salt Master

Parameter	Default JSON output	Description
Salt Master address	salt_master_address: 10.0.1.90	The IP address of the Salt Master node on the control network
Salt Master management address	salt_master_management_address: 10.0.0.90	The IP address of the Salt Master node on the management network
Salt Master hostname	salt_master_hostname: cfg01	The hostname of the Salt Master node

Ubuntu MAAS

Parameter	Default JSON output	Description
MAAS hostname	maas_hostname: mas01	The hostname of the MAAS virtual server
MAAS deploy address	maas_deploy_address: 10.0.0.16	The IP address of the MAAS node on the deploy network

Publication options

Parameter	Default JSON output	Description
Email address	email_address: <your-email>	The email address where the generated Reclass model will be sent to

OpenStack networking

Parameter	Default JSON output	Description
Openstack network engine	openstack_network_engine: opencontrail	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.

CI/CD

Parameter	Default JSON output	Description
OpenLDAP enabled	openldap_enabled: 'True'	Enables OpenLDAP authentication

Repositories

Parameter	Default JSON output	Description
Local repositories	local_repositories: 'False'	If true, changes repositories URLs to local mirrors. The local_repo_url parameter should be added manually after model generation.

Product related parameters

The tables in this section outline the product configuration parameters including infrastructure, CI/CD, OpenContrail, OpenStack, and Stacklight 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
- OpenContrail service parameters
- OpenStack product parameters
- StackLight product parameters

Infrastructure product parameters

Section	Default JSON output	Description
Infra kvm01 hostname	infra_kvm01_hostname: kvm01	The hostname of the first KVM node
Infra kvm01 control address	infra_kvm01_control_address: 10.0.1.241	The IP address of the first KVM node on the control network
Infra kvm01 deploy address	infra_kvm01_deploy_address: 10.0.0.241	The IP address of the first KVM node on the management network
Infra kvm02 hostname	infra_kvm02_hostname: kvm02	The hostname of the second KVM node

Infra kvm02 control address	infra_kvm02_control_address: 10.0.1.242	The IP address of the second KVM node on the control network
Infra kvm02 deploy address	infra_kvm02_deploy_address: 10.0.0.242	The IP address of the second KVM node on the management network
Infra kvm03 hostname	infra_kvm03_hostname: kvm03	The hostname of the third KVM node
Infra kvm03 control address	infra_kvm03_control_address: 10.0.1.243	The IP address of the third KVM node on the control network
Infra kvm03 deploy address	infra_kvm03_deploy_address: 10.0.0.243	The IP address of the third KVM node on the management network
Infra KVM VIP address	infra_kvm_vip_address: 10.0.1.240	The virtual IP address of the KVM cluster
Infra deploy NIC	infra_deploy_nic: eth0	The NIC used for PXE of the KVM hosts
Infra primary first NIC	infra_primary_first_nic: eth1	The first NIC in the KVM bond
Infra primary second NIC	infra_primary_second_nic: eth2	The second NIC in the KVM bond
Infra bond mode	infra_bond_mode: active-backup	The bonding mode for the KVM nodes. Available options include: • active-backup • balance-xor • broadcast • 802.3ad • balance-ltb • balance-alb To decide which bonding mode best suits the needs of your deployment, you can consult the official Linux bonding documentation.

OpenStack compute count	openstack_compute_count: '100'	The number of compute nodes to be generated. The naming convention for compute nodes is cmp000 - cmp${openstack_compute_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

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

OpenContrail service parameters

Section	Default JSON output	Description
OpenContrail analytics address	opencontrail_analytics_address: 10.0.1.30	The virtual IP address of the OpenContrail analytics cluster
OpenContrail analytics hostname	opencontrail_analytics_hostname: nal	The hostname of the OpenContrail analytics cluster
OpenContrail analytics node01 address	opencontrail_analytics_node01_address: 10.0.1.31	The virtual IP address of the first OpenContrail analytics node on the control network

OpenContrail analytics node01 hostname	opencontrail_analytics_node01_hostname: nal01	The hostname of the first OpenContrail analytics node on the control network
OpenContrail analytics node02 address	opencontrail_analytics_node02_address: 10.0.1.32	The virtual IP address of the second OpenContrail analytics node on the control network
OpenContrail analytics node02 hostname	opencontrail_analytics_node02_hostname: nal02	The hostname of the second OpenContrail analytics node on the control network
OpenContrail analytics node03 address	opencontrail_analytics_node03_address: 10.0.1.33	The virtual IP address of the third OpenContrail analytics node on the control network
OpenContrail analytics node03 hostname	opencontrail_analytics_node03_hostname: nal03	The hostname of the second OpenContrail analytics node on the control network
OpenContrail control address	opencontrail_control_address: 10.0.1.20	The virtual IP address of the OpenContrail control cluster
OpenContrail control hostname	opencontrail_control_hostname: ntw	The hostname of the OpenContrail control cluster
OpenContrail control node01 address	opencontrail_control_node01_address: 10.0.1.21	The virtual IP address of the first OpenContrail control node on the control network
OpenContrail control node01 hostname	opencontrail_control_node01_hostname: ntw01	The hostname of the first OpenContrail control node on the control network
OpenContrail control node02 address	opencontrail_control_node02_address: 10.0.1.22	The virtual IP address of the second OpenContrail control node on the control network
OpenContrail control node02 hostname	opencontrail_control_node02_hostname: ntw02	The hostname of the second OpenContrail control node on the control network
OpenContrail control node03 address	opencontrail_control_node03_address: 10.0.1.23	The virtual IP address of the third OpenContrail control node on the control network
OpenContrail control node03 hostname	opencontrail_control_node03_hostname: ntw03	The hostname of the third OpenContrail control node on the control network
OpenContrail router01 address	opencontrail_router01_address: 10.0.1.100	The IP address of the first OpenContrail gateway router for BGP

OpenContrail router01 hostname	opencontrail_router01_hostname: rtr01	The hostname of the first OpenContrail gateway router for BGP
OpenContrail router02 address	opencontrail_router02_address: 10.0.1.101	The IP address of the second OpenContrail gateway router for BGP
OpenContrail router02 hostname	opencontrail_router02_hostname: rtr02	The hostname of the second OpenContrail gateway router for BGP

OpenStack product parameters

Section	Default JSON output	Description
Compute primary first NIC	compute_primary_first_nic: eth1	The first NIC in the OpenStack compute bond
Compute primary second NIC	compute_primary_second_nic: eth2	The second NIC in the OpenStack compute bond
Compute bond mode	compute_bond_mode: active-backup	The bond mode for the compute nodes
OpenStack compute rack01 hostname	openstack_compute_rack01_hostname: cmp	The compute hostname prefix
OpenStack compute rack01 single subnet	openstack_compute_rack01_single_subnet: 10.0.0.1	The control plane network prefix for compute nodes
OpenStack compute rack01 tenant subnet	openstack_compute_rack01_tenant_subnet: 10.0.2	The data plane netwrok prefix for compute nodes
OpenStack control address	openstack_control_address: 10.0.1.10	The virtual IP address of the control cluster on the control network
OpenStack control hostname	openstack_control_hostname: ctl	The hostname of the VIP control cluster
OpenStack control node01 address	openstack_control_node01_address: 10.0.1.11	The IP address of the first control node on the control network
OpenStack control node01 hostname	openstack_control_node01_hostname: ctl01	The hostname of the first control node
OpenStack control node02 address	openstack_control_node02_address: 10.0.1.12	The IP address of the second control node on the control network

OpenStack control node02 hostname	openstack_control_node02_hostname: ctl02	The hostname of the second control node
OpenStack control node03 address	openstack_control_node03_address: 10.0.1.13	The IP address of the third control node on the control network
OpenStack control node03 hostname	openstack_control_node03_hostname: ctl03	The hostname of the third control node
OpenStack database address	openstack_database_address: 10.0.1.50	The virtual IP address of the database cluster on the control network
OpenStack database hostname	openstack_database_hostname: dbs	The hostname of the VIP database cluster
OpenStack database node01 address	openstack_database_node01_address: 10.0.1.51	The IP address of the first database node on the control network
OpenStack database node01 hostname	openstack_database_node01_hostname: dbs01	The hostname of the first database node
OpenStack database node02 address	openstack_database_node02_address: 10.0.1.52	The IP address of the second database node on the control network
OpenStack database node02 hostname	openstack_database_node02_hostname: dbs02	The hostname of the second database node
OpenStack database node03 address	openstack_database_node03_address: 10.0.1.53	The IP address of the third database node on the control network
OpenStack database node03 hostname	openstack_database_node03_hostname: dbs03	The hostname of the third database node
OpenStack message queue address	openstack_message_queue_address: 10.0.1.40	The vitrual IP address of the message queue cluster on the control network
OpenStack message queue hostname	openstack_message_queue_hostname: msg	The hostname of the VIP message queue cluster
OpenStack message queue node01 address	openstack_message_queue_node01_address: 10.0.1.41	The IP address of the first message queue node on the control network

OpenStack message queue node01 hostname	openstack_message_queue_node01_hostname: msg01	The hostname of the first message queue node
OpenStack message queue node02 address	openstack_message_queue_node02_address: 10.0.1.42	The IP address of the second message queue node on the control network
OpenStack message queue node02 hostname	openstack_message_queue_node02_hostname: msg02	The hostname of the second message queue node
OpenStack message queue node03 address	openstack_message_queue_node03_address: 10.0.1.43	The IP address of the third message wueue node on the control network
OpenStack message queue node03 hostname	openstack_message_queue_node03_hostname: msg03	The hostname of the third message queue node
OpenStack benchmark node01 address	openstack_benchmark_node01_address: 10.0.1.95	The IP address of a benchmark node on the control network
OpenStack benchmark node01 hostname	openstack_benchmark_node01_hostname: bmk01	The hostname of a becnhmark node
OpenStack proxy address	openstack_proxy_address: 10.0.1.80	The virtual IP address of a proxy cluster on the control network
OpenStack proxy hostname	openstack_proxy_hostname: prx	The hostname of the VIP proxy cluster
OpenStack proxy node01 address	openstack_proxy_node01_address: 10.0.1.81	The IP address of the first proxy node on the control network
OpenStack proxy node01 hostname	openstack_proxy_node01_hostname: prx01	The hostname of the first proxy node
OpenStack proxy node02 address	openstack_proxy_node02_address: 10.0.1.82	The IP address of the second proxy node on the control network
OpenStack proxy node02 hostname	openstack_proxy_node02_hostname: prx02	The hostname of the second proxy node
OpenStack telemetry address	openstack_telemetry_address: 10.0.1.75	The virtual IP address of a telemetry cluster on the control network
OpenStack telemetry hostname	openstack_telemetry_hostname: mdb	The hostname of the VIP telemetry cluster

OpenStack telemetry node01 address	openstack_telemetry_node01_address: 10.0.1.76	The IP address of the first telemetry node on the control network
OpenStack telemetry node01 hostname	openstack_telemetry_node01_hostname : mdb01	The hostname of the first telemetry node
OpenStack telemetry node02 address	openstack_telemetry_node02_address: 10.0.1.77	The IP address of the second telemetry node on the control network
OpenStack telemetry node02 hostname	openstack_telemetry_node02_hostname : mdb02	The hostname of the second telemetry node
OpenStack telemetry node03 address	openstack_telemetry_node03_address: 10.0.1.78	The IP address of the third telemetry node on the control network
OpenStack telemetry node03 hostname	openstack_telemetry_node03_hostname : mdb03	The hostname of the third telemetry node
OpenStack version	openstack_version: mitaka	The version of OpenStack to be deployed

StackLight product parameters

Section	Default JSON output	Description
StackLight log address	stacklight_log_address: 10.0.1.60	The virtual IP address of the StackLight logging cluster
StackLight log hostname	stacklight_log_hostname: log	The hostname of the StackLight logging cluster
StackLight log node01 address	stacklight_log_node01_address: 10.0.1.61	The IP address of the first StackLight logging node
StackLight log node01 hostname	stacklight_log_node01_hostname: log01	The hostname of the first StackLight logging node
StackLight log node02 address	stacklight_log_node02_address: 10.0.1.62	The IP address of the second StackLight logging node
StackLight log node02 hostname	stacklight_log_node02_hostname: log02	The hostname of the second StackLight logging node
StackLight log node03 address	stacklight_log_node03_address: 10.0.1.63	The IP address of the third StackLight logging node
StackLight log node03 hostname	stacklight_log_node03_hostname: log03	The hostname of the third StackLight logging node

StackLight monitor address	stacklight_monitor_address: 10.0.1.70	The virtual IP address of the StackLight monitoring cluster
StackLight monitor hostname	stacklight_monitor_hostname: mon	The hostname of the StackLight monitoring cluster
StackLight monitor node01 address	stacklight_monitor_node01_address: 10.0.1.71	The IP address of the first StackLight monitoring node
StackLight monitor node01 hostname	stacklight_monitor_node01_hostname: mon01	The hostname of the first StackLight monitoring node
StackLight monitor node02 address	stacklight_monitor_node02_address: 10.0.1.72	The IP address of the second StackLight monitoring node
StackLight monitor node02 hostname	stacklight_monitor_node02_hostname: mon02	The hostname of the second StackLight monitoring node
StackLight monitor node03 address	stacklight_monitor_node03_address: 10.0.1.73	The IP address of the third StackLight monitoring node
StackLight monitor node03 hostname	stacklight_monitor_node03_hostname: mon03	The hostname of the third StackLight monitoring node
StackLight telemetry address	stacklight_telemetry_address: 10.0.1.85	The virtual IP address of a StackLight telemetry cluster
StackLight telemetry hostname	stacklight_telemetry_hostname: mtr	The hostname of a StackLight telemetry cluster
StackLight telemetry node01 address	stacklight_telemetry_node01_address: 10.0.1.86	The IP address of the first StackLight telemetry node
StackLight telemetry node01 hostname	stacklight_telemetry_node01_hostname: mtr01	The hostname of the first StackLight telemetry node
StackLight telemetry node02 address	stacklight_telemetry_node02_address: 10.0.1.87	The IP address of the second StackLight telemetry node
StackLight telemetry node02 hostname	stacklight_telemetry_node02_hostname: mtr02	The hostname of the second StackLight telemetry node
StackLight telemetry node03 address	stacklight_telemetry_node03_address: 10.0.1.88	The IP address of the third StackLight telemetry node
StackLight telemetry node03 hostname	stacklight_telemetry_node03_hostname: mtr03	The hostname of the third StackLight telemetry node

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:

```
git add *
git commit –m "Initial commit"

git pull -r
git push --set-upstream origin master
```

Install a base infrastructure

Base infrastructure of your MCP cluster includes the Salt Master node and virtualized control plane (VCP). This section explains 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.

Fetch source images:

1. Log in to the Foundation node console.
2. Download the images:
   - ubuntu-16-04-x64-mcp1.0.qcow2 for physical servers, Salt Master, MaaS, and DriveTrain nodes
   - ubuntu-14-04-x64-mcp1.0.qcow2 for the OpenStack controller nodes
3. Place the images to /var/lib/libvirt/images/.
4. 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
   ```

Prepare the image 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 the image for Salt Master and MaaS nodes:

1. Create a new sparsely allocated image:

   ```
   qemu-img create -f qcow2 -o preallocation=off <name>.qcow2 <num>G
   ```

   For example:

   ```
   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:

```
apt-get install libguestfs-tools
```

3. Use the virt-resize command to expand the partition and file system:

```
virt-resize --expand /dev/vda1 <src>.src.qcow2 / <name>.qcow2
```

For example:

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

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.

Prepare the Foundation node

The Foundation node is a physical node that hosts Salt Master, MaaS, and each of the first VM nodes of the control plane.

To prepare the Foundation node for the Salt Master deployment:

1. Install Ubuntu 16.04 with OpenSSH server and Standard system utilities on it.
2. Create the bridge to provision network on the foundation node:

```
# brctl addbr br-pxe
  brctl addif br-pxe <PROVISION_INTERFACE>
```

Where <PROVISION_INTERFACE> is the interface name connected to the PXE network.

Seealso

- Prepare the image for the Foundation node

Install the Salt Master node

Using libvirt CLI

Create config drive iso

```
apt-get install -y mkisofs curl virtinst cpu-checker qemu-kvm
curl -q https://raw.githubusercontent.com/larsks/virt-utils/master/create-config-drive |\
sed s,/bin/sh,/bin/bash,g >\
create-config-drive.sh
chmod +x create-config-drive.sh

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

Boot VM

1. Initialize the VM_HOSTNAME variable. For example, VM_HOSTNAME=cfg01.
2. Boot the VM:

```
# using: /var/lib/libvirt/images/cfg01.qcow2

virt-install --name $VM_HOSTNAME --ram 8096 --vcpus=4 --accelerate \
--network bridge:br-pxe,model=virtio \
--disk path=/var/lib/libvirt/images/cfg01.qcow2,size=200,bus=virtio,cache=none \
--boot hd --vnc --noreboot --autostart \
--disk path=/var/lib/libvirt/images/vm-config.iso,device=cdrom

virsh --connect qemu:///system start $VM_HOSTNAME
```

Using Virtual Machine Manager UI

To install the Salt Master node:

1. Import the existing disk image:
    1. Use the cfg01.qcow2 image downloaded and resized earlier
        1. Set OS Type to Linux.
        2. Set Version to Ubuntu 16.04 LTS.
        3. Click Forward.
2. Set the desired amount of RAM. For example, 8096 MB.
3. Set the desired number of vCPUs. For example, 4.
4. Click Forward.
5. Rename the VM to cfg01.
6. Select Customize the configuration before install.
7. Click Network Selection and select Specify shared device name.
8. Enter the configured bridge name for the PXE network - br-pxe.
9. Click Finish.

Bootstrap the Salt Master node

To bootstrap the Salt Master node:

1. Log in to the Salt Master node console.
2. Install the required packages:

```
apt-get update
apt-get install git curl subversion
```

3. Download the deploy scripts to the /srv/salt/scripts directory:

```
svn export --force https://github.com/salt-formulas/salt-formulas/trunk/deploy/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 and add password to the /srv/salt/reclass/classes/system directory:

> Note
>
> You may want to ping to a specific tag, for exmaple, mcp1.0.

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

7. Run the salt-mastet-init.sh script from /srv/salt/scripts with the MASTER_HOSTNAME=$SALT_MASTER_FQDN parameter:

> Note
>
> FQDN must match the specification under nodes/cfg01*.yml in your repository

```
cd /srv/salt/scripts
MASTER_HOSTNAME=cfg01.infra.ci.local ./salt-master-init.sh
```

8. Verify 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:
   ```

9. To verify the Reclass model:

   ```
   source /srv/salt/scripts/salt-master-init.sh

   verify-salt-master
   verify-salt-minions
   ```

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:

```
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/*

# The Loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 172.31.35.12/24

auto eth1
iface eth1 inet static
address 172.31.110.12/24
gateway 172.31.110.1
dns-nameservers 8.8.8.8
dns-search mk.slab.local
```

    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

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

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

```
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

Verify the Reclass model on the Salt Master node

To validate the Reclass model and node pillars:

To perform verification on salt master host:

1. To verify salt master pillar:

   ```
   source /srv/salt/scripts/salt-master-init.sh
   verify_salt_master
   ```

2. To verify minions pillars:

   1. Source salt-master-init.sh script and execute the verify_salt_minion function:

      ```
      source /srv/salt/scripts/salt-master-init.sh
      verify_salt_minion
      ```

Verify the model on a Minion

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 Virt 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
#- system.salt.control.cluster.cicd_control_cluster
- cluster.cz-bud-mirantis-net.cicd.gluster

#     cluster:
#       internal:
#         node:
#           mdb01:
#             image: ${_param:salt_control_xenial_image}
#           mdb02:
#             image: ${_param:salt_control_xenial_image}
#           mdb03:
#             image: ${_param:salt_control_xenial_image}
#           ctl01:
#             image: ${_param:salt_control_xenial_image}
#           ctl02:
#             image: ${_param:salt_control_xenial_image}
#           ctl03:
#             image: ${_param:salt_control_xenial_image}
#           dbs01:
```

```
#          image: ${_param:salt_control_xenial_image}
#        dbs02:
#          image: ${_param:salt_control_xenial_image}
#        dbs03:
#          image: ${_param:salt_control_xenial_image}
#        msg01:
#          image: ${_param:salt_control_xenial_image}
#        msg02:
#          image: ${_param:salt_control_xenial_image}
#        msg03:
#          image: ${_param:salt_control_xenial_image}
#        prx01:
#          image: ${_param:salt_control_xenial_image}
#        prx02:
#          image: ${_param:salt_control_xenial_image}
```

3. Run Salt Virt to start the MAAS virtual server.
4. Uncomment the lines in /srv/salt/reclass/classes/cluster/cz-bud-mirantis-net/infra/kvm.yml that you commented in Step 2.
5. Log in to the mas01 node console.

```
virsh console $(virsh list|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 authentification, 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':
is not a valid commissioning_distro_series.  It should be one of: \'---\'."]}', 'default_osystem': '{"default_
', 'default_distro_series': '{"default_distro_series": ["\'xenial\' is not a valid release.  It should be one of:
_discovery_interval', 'upstream_dns', 'enable_third_party_drivers', 'maas_name', 'default_storage_layout
      Started: 12:10:24.686243
     Duration: 1544.69 ms
      Changes:
```

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 infra_maas_database_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.

Deploy physical nodes using MAAS

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

This section describes how to deploy 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 deploy include kvm02 and kvm03 KVM nodes along with the compute cmp01 KVM node.

You can provision physical nodes automatically or manually:

- An automated deployment requires you to define IPMI and MAC addresses in your Reclass model. After you enforce all servers, the Salt Master node commissions and deploys them automatically.
- A manual deployment enables commissioning nodes through the MAAS web UI.

Warning

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

Automatically commission and deploy the physical nodes

This section explains how to define physical nodes in a Reclass model to automatically commission and then deploy 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:

   ```
   maas:
     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:

   ```
   ipmitool -U ipmi_user-P passowrd -H kvm02.ipmi.net raw 0x30 0x21 1| tail -c 18
   ```

2. If required, the modifynode_info_scripts.py script on the /usr/lib/python3/dist-packages/provisioningserver/refresh/ directory to configure unified

NICs naming in MAAS. For example, to set one for a 1 Gb Ethernet and ten for a 10 Gb Ethernet, define the following script content for IPADDR_SCRIPT:

```
IPADDR_SCRIPT = dedent("""\
  #!/bin/bash

  i=1
  j=1

  ethernets=$(ip -o l | grep -E '^[0-9]+:\ .*link/ether.*' | awk -F":" '/,/{gsub(/ /, "", $0); print $2}' | grep -
  declare -A interfaces
  for iface in ${ethernets[@]}; do
    speed=`ethtool $iface | grep '40000'`
    if [[ "$?" == 0 ]]; then
      interfaces[$iface]="fourty$i"
      ((i++))
    else
      speed=`ethtool $iface | grep '10000' `
      if [[ "$?" == 0 ]]; then
        interfaces[$iface]="ten$i"
        ((i++))
      else
        interfaces[$iface]="one$j"
        ((j++))
      fi
    fi
  done

  for i in ${!interfaces[@]}; do
    sedline+="-e s/${i}/${interfaces[$i]}/ "
  done

  ip addr | sed ${sedline}
  """)
```

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

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

   All nodes are automatically commissioned.

4. Verify the status of machines 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.

5. Verify that all servers have correct NIC names and configurations.
6. Proceed to Deploy the automatically commissioned physical nodes.

Deploy the automatically commissioned physical nodes

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

To deploy 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:
    ----------
    machines:
        - hostname:kvm02,system_id:anc6a4,status:Deploying
    summary:
        ----------
        Deploying:
            1
```

4. When all servers have been deployed, 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 deploy the physical nodes

This section explains how to discover, commission, and deploy 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:

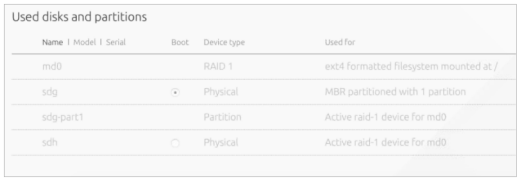
1. Power on a physical node.

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-x), 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 14.04 LTS “Trusty Tahr”, respectively.
6. Leave the remaining parameters as defaults.
7. 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.
   9. In File system, select ext4.
   10. Set / as Mount point.
   11. Click Create RAID.

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

8. Repeat the above steps for each physical node.
9. Proceed to Manually deploy the physical nodes.

Manually deploy the physical nodes

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

To manually deploy 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 14.04 LTS “Trusty Tahr” with the Xenial(hwe-x) kernel is selected.
6. Click Go to deploy the node.
7. Repeat the above steps for each node.

Now, your physical nodes are deployed 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.

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:

```
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 VMs on the first KVM node.

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 2016.3.4 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 2016.3.4 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. Remove extra from /etc/apt/sources.list.d/tcpcloud_contrail.list:

```
deb [arch=amd64] http://apt.tcpcloud.eu/nightly trusty tcp oc30
```

4. Sync the packages index files:

```
apt-get update
```

5. Verify the versions:

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

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

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

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

```
apt-get install salt-common=2016.3.4
apt-get install salt-minion=2016.3.4
```

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

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

9. Verify that the required version is installed:

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

10 Repeat the procedure on each physical node.
.

Create and provision VMs on the first KVM node

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 node that does not host the Salt Master node and MaaS node. The correct physical node name used in the installation described in this guide to perform the next step is kvm02.

> 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 -v mas | grep -v cfg | xargs -n 1 virsh destroy
virsh list --name --all | grep -v mas | grep -v cfg | xargs -n 1 virsh undefine
```

3. Log in to the Salt Master node console

4. Run the libvirt state:

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

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

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

6. To set the created virtual machines to automatically start after powering on the physical nodes, run the virsh autostart command on each of the physical nodes:

```
virsh list --name | xargs -n 1 virsh autostart
```

7. On kvm02, run the Salt state for the minions to create the environment virtual machines:

```
salt-call state.sls salt
```

8. Verify that the required version of salt-common is installed:

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

9. Run the salt state:

```
salt-call state.sls salt
```

> Note
>
> The libvirt_service error may occur:
>
> ```
> ID: libvirt_service
> Function: service.running
> Name: libvirt-bin
> Result: False
> Comment: The named service libvirt-bin is not available
> Started: 22:09:35.231541
> Duration: 30.584 ms
> ```
>
> This is a known issue and can be ignored.

10 In the state.sls file, change lines 55 and 56 from libvirt-bin to libvirtd. For example:
.

```
}, merge=salt['grains.filter_by']({
    'trusty': {
        'libvirt_bin': '/etc/default/libvirtd',
        'libvirt_service': 'libvirtd',
    }
```

11. To verify that the minions are in sync, run the following command on the Salt Master node:

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

12. On the affected nodes, re-run the libvirt state. In our example, it is kvm02:

```
salt-call state.sls libvirt
```

13. To restart the libvirt service on the affected nodes, run the following command on the Salt Master node:

```
salt '*kvm02*' cmd.run 'service libvirtd restart'
```

14. Re-run the Salt state on kvm02:

```
salt-call state.sls salt
```

15. Sync your Git repository and local repository on your Salt Master node (the local path is /srv/salt/reclass). Pull the synced changes to the local server:

```
root@cfg01:/srv/salt/reclass# git pull
```

16. On kvm02, re-run the linux and salt states:

```
salt-call state.sls linux,salt
```

Provision VMs on the second KVM node

Provisioning of the control plane VMs that run on the second kvm03 physical node is simpler because all the configurations were made for kvm02. Though, you may still get some issues that require manual intervention.

To provision the control plane VMs on the second KVM node:

1. Log in to kvm03.
2. In the /etc/apt/sources.list.d/salt.list file, remove the repository so you can install the correct version of salt-common and salt-minion. For example:

```
#deb [arch=amd64] http://repo.saltstack.com/apt/ubuntu/ubuntu14/2016.3 trusty main
```

3. Install the correct versions of salt-common and salt-minion.

4. Verify the salt-common and salt-minion versions
5. Run the libvirt and salt states on kvm03:

```
salt-call state.sls libvirt,salt.control
```

6. Log in to the Salt Master cfg01 node.
7. Run the salt.highstate against kvm03:

```
salt '*kvm03*' state.highstate
```

Provision VMs on the third KVM node

The provisioning of the control plane VMs hosted by the physical node where the mas01 MaaS node and cfg01 Salt Master node reside should be performed last.

To provision the control plane VMs on the third KVM node:

1. Log in to the mas01 node.
2. Connect the kvm01 controller node through SSH. For example:

```
ssh ubuntu@172.31.25.11
```

3. Switch into the root context:

```
sudo -i
```

4. Run the Salt states for linux, ntp, openssh, and libvirt:

```
salt-call state.sls linux,ntp,openssh,libvirt
```

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 need to set up the KVM physical nodes right after the physical nodes are deployed with MaaS as described in deploy_physical_nodes.

To set up the physical nodes for CI/CD:

1. Perform the initial Salt configuration:

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

2. Set up network interfaces and the SSH access:

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

3. Wait until the network interfaces come up and finish with the linux state application:

```
salt -C 'I@salt:control' state.sls linux
```

4. Install libvirt:

```
salt -C 'I@salt:control' state.sls libvirt
```

5. Enable virtual IP:

```
salt -C 'I@salt:control' state.sls keepalived
```

6. Deploy the GlusterFS cluster:

```
salt -C 'I@glusterfs:server' state.sls glusterfs.server.service
salt -C 'I@glusterfs:server and *01*' state.sls glusterfs.server.setup
```

7. Synchronize modules and states:

```
salt -C 'I@salt:control' saltutil.sync_all
```

8. Spin virtual machines through salt-virt:

```
salt -C 'I@salt:control' state.sls salt.control
```

9. Verify that the CI nodes appear in Salt.

Once the CI nodes are up and running, you can start deploying your CI/CD infrastructure as described in 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:

```
salt -C 'ci*' cmd.run 'salt-call state.sls salt.minion'
salt -C 'ci*' state.sls salt.minion,linux,openssh,ntp
```

4. Synchronize modules and states:

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

5. Mount Gluster volumes from the KVM nodes:

```
salt -C 'I@glusterfs:client and I@docker:host' state.sls glusterfs.client
```

6. Configure virtual IP and HAProxy balancing:

```
salt -C 'I@haproxy:proxy and I@docker:host' state.sls haproxy,keepalived
```

7. Install Docker:

```
salt -C 'I@docker:host' state.sls docker.host
```

8. Initial Docker swarm leader:

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

9. Update the Salt mine to enable other swarm nodes to connect to leader:

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

10. Complete the Docker swarm deployment:

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

11. Verify that all nodes are in the cluster:

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

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

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

13. Configure the Aptly service:

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

14. Configure the OpenLDAP service for Jenkins and Gerrit:

```
salt -C 'I@openldap:client' cmd.run 'salt-call state.sls openldap'
```

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

.

```
salt -C 'I@gerrit:client' cmd.run 'salt-call state.sls gerrit'
```

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

16 Configure the Jenkins service, create users, add pipelines, and so on:

.

```
salt -C 'I@jenkins:client' cmd.run 'salt-call 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 DevOps portal using the manual installation procedure described in Deploy the DevOps portal 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

Seealso

- View the deployment details
- Deploy the DevOps portal 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-cleanup 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.

Seealso

- Deploy CI/CD using Heat templates

Deploy an MCP cluster using DriveTrain

After you have installed the MCP CI/CD infrastructure as descibed in Deploy DriveTrain, the Jenkins web UI can be reached through the Jenkins master IP address. This section contains procedures explaining how to deploy MCP clusters using the 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.

To deploy an OpenStack environment using CI/CD:

> 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.

1. Go to the central Jenkins lab.
2. Log in to the Jenkins web UI using your credentials.
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. Specify the following parameters:

Deploy - OpenStack environment parameters

Parameter	Description and values
ASK_ON_ERROR	If checked, Jenkins will ask either to stop a pipeline or continue execution in case of Salt state fails on any task

INSTALL	Specifies the components you need to install. The available values include: • core • kvm • openstack • stacklight • ovs or contrail (depending on the network plugin)
SALT_MASTER_CREDENTIALS	Includes credentials to Salt API
SALT_MASTER_URL	Specifies the reachable IP address of the Salt Master node and port on which Salt API listens. For example, http://172.18.170.28:6969 To find out on which port Salt API listens: 1. Log in to the Salt Master node. 2. Search for the port in the /etc/salt/master.d/_api.conf file. 3. Verify that the Salt Master node is listening on that port: `netstat -tunelp | grep <PORT>`
STACK_TYPE	Specifies the environment type. Use physical for a bare metal deployment

6. Click Build.

Seealso

- View the deployment details

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.

Seealso

- View the deployment details

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. To view the whole output, click Full log.

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, StackLight, and others.

Deploy the DevOps portal

The DevOps portal is the MCP Control Plane component that is responsible for managing your MCP deployment as the main administrative dashboard for the entire environment.

> Note
>
> The DevOps portal is provided as a technical preview.

Deploy the DevOps portal manually

Before you can deploy the DevOps portal, you must complete the steps described in Deploy DriveTrain.

> Note
>
> The DevOps portal is provided as a technical preview.

To manually deploy the DevOps portal:

1. Log in to the Salt Master node.
2. Upload the DevOps Portal formula:
    1. Clone the source code of the formula:

        ```
        git clone https://gerrit.mcp.mirantis.net/oss/salt-formula-devops-portal \
        /usr/share/salt-formulas/env/_formulas/devops_portal
        ```

    2. Create a symbolic link to the formula subdirectory:

        ```
        ln -s /usr/share/salt-formulas/env/_formulas/devops_portal/devops_portal \
        /usr/share/salt-formulas/env/devops_portal
        ```

    3. Create symbolic links to the service metadata subdirectory:

        ```
        ln -s /usr/share/salt-formulas/env/_formulas/devops_portal/metadata/service/ \
        /usr/share/salt-formulas/reclass/service/devops_portal
        ```

```
ln -s /usr/share/salt-formulas/reclass/service/devops_portal \
/srv/salt/reclass/classes/service/devops_portal
```

4. Create a symbolic link to the devops_utils.py module:

```
ln -s /usr/share/salt-formulas/env/_formulas/devops_portal/_modules/devops_utils.py \
/usr/share/salt-formulas/env/_modules/devops_utils.py
```

3. Configure the cluster metadata by editing the following configuration files:

1. Add the following classes and parameters to the /srv/salt/reclass/classes/cluster/cicd_lab_dev/cicd/control/init.yml:

```
classes:
  - system.glusterfs.server.volume.devops_portal
  - system.glusterfs.server.volume.rundeck
  - system.glusterfs.client.volume.devops_portal
  - system.glusterfs.client.volume.rundeck
  - system.docker.swarm.service.devops_portal
  - system.docker.swarm.service.rundeck
  - system.haproxy.proxy.listen.oss.devops_portal
  - system.haproxy.proxy.listen.oss.rundeck
  # DevOps Portal
  - system.devops_portal.service.gerrit
  - system.devops_portal.service.jenkins
  - system.devops_portal.service.rundeck
```

2. Add the following class to the /srv/salt/reclass/classes/cluster/cicd_lab_dev/cicd/control/master.yml:

```
classes:
  - service.devops_portal.config
```

4. Verify that metadata of the Salt Master node contains all the required parameters.

```
reclass --nodeinfo=$SALT_MASTER_FQDN.$ENV_DOMAIN
salt '*' saltutil.refresh_pillar
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 'ci01.cicd-lab-dev.local' pillar.get devops_portal
```

5. Apply the Salt state to the deploy DevOps portal:

1. Synchronize modules:

```
salt '*' saltutil.sync_modules
```

2. Apply states in the following strict order:

```
salt '*' state.sls glusterfs
salt -C 'I@devops_portal:control' state.sls devops_portal.config
salt '*' state.sls docker.client.service
salt '*' state.sls haproxy.proxy
```

Now, you can access the DevOps portal at the VIP address of the deployment on port 8800.

Build a custom image of the DevOps portal

For testing purposes, you may need to create a custom Docker image for 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:

```
git clone https://gerrit.mcp.mirantis.net/oss/devops-portal
cd devops-portal
```

2. Build your image:

```
docker build --rm -f docker/Dockerfile -t \
docker-sandbox.sandbox.mirantis.net/[USERNAME]/oss/devops-portal:latest .
```

3. Push the image into some specific prefix on Sandbox:

```
docker push docker-sandbox.sandbox.mirantis.net/[USERNAME]/oss/devops-portal:latest
```

Now, you can use this specific Docker image to deploy the DevOps Portal.

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:
>
> 2. Galera
> 3. HAProxy
> 1. Keepalived

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:

```
salt -C 'I@keepalived:cluster' state.sls keepalived -b 1
```

To verify the VIP addresses:

```
salt -C 'I@keepalived:cluster' cmd.run "ip a | grep 172.16.10.2"
```

> 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
```

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:

```
salt -C 'I@rabbitmq:server' state.sls rabbitmq
```

To verify the RabbitMQ status:

```
salt -C 'I@rabbitmq:server' cmd.run "rabbitmqctl cluster_status"
```

Deploy Galera

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:

```
salt -C 'I@galera:master' state.sls galera
salt -C 'I@galera:slave' state.sls galera
```

To verify Galera:

```
salt -C 'I@galera:master' mysql.status | grep -A1 wsrep_cluster_size
salt -C 'I@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:

```
salt -C 'I@haproxy:proxy' state.sls haproxy
salt -C 'I@haproxy:proxy' service.status haproxy
salt -I 'haproxy:proxy' service.restart rsyslog
```

Deploy Memcached

Memcached is used for caching data for different OpenStack services such as Keystone, for example.

To deploy Memcached:

```
salt -C 'I@memcached:server' state.sls memcached
```

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; keystone 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@glance:server' 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 Neutron

To install Neutron:

```
salt -C 'I@neutron:server' state.sls neutron -b 1
salt -C 'I@neutron:gateway' state.sls neutron
salt -C 'I@keystone:server' cmd.run ". /root/keystonerc; neutron agent-list"
```

> Warning
>
> For installations with the OpenContrail setup, perform the API check after the Contrail control installation.

Deploy Horizon

To install Horizon:

```
salt -C 'I@horizon:server' state.sls horizon
salt -C 'I@nginx:server' state.sls nginx
```

Deploy Tenant Telemetry

Tenant Telemetry collects metrics about the OpenStack resources and provides this data through the Ceilometer API.

To install 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 StackLight back ends.

> 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 proxy nodes

To install proxy nodes:

1. Add NAT for br2:

```
root@kvm01:~# iptables -t nat -A POSTROUTING -o br2 -j MASQUERADE
root@kvm01:~# echo “1” > /proc/sys/net/ipv4/ip_forward
root@kvm01:~# iptables-save > /etc/iptables/rules.v4
```

2. Deploy linux, openssh, and salt states to the proxy nodes:

```
root@cfg01:~# salt 'prx*' state.sls linux,openssh,salt
```

3. Verify the connection to Horizon:

    1. You may need first to configure a SOCKS proxy or similar to the environment network to gain access from within your browser.
    2. In a browser, connect to each of the proxy IPs and its VIP to verify that they are active.

Install OpenContrail

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. However, OpenContrail only supports KVM and Citrix Xen hypervisors.

Install OpenContrail

To install OpenContrail:

1. Install OpenContrail with salt-call directly from the network nodes.

   > Note
   >
   > This step may take about 5 minutes to complete.

   ```
   salt-call state.sls opencontrail
   ```

   > Warning
   >
   > The deployment may fail with two errors:
   >
   > - Error stating that iptables service failed. Can be ignored.
   > - Error stating that net.netfilter is unavailable.
   >
   > Rerun the OpenContrail state until only a single iptables failure remains.

2. Install OpenContrail on the remaining network nodes individually:

   ```
   salt-call state.sls opencontrail
   ```

3. Verify the status of the service from one of the network nodes:

   ```
   contrail-status
   ```

   In the output, the services status should be active.

   > Note
   >
   > It may take some time for all services to finish initializing.

   If you have issues during the OpenContrail deployment, check the Troubleshoot OpenContrail section of this document.

4. Deploy OpenContrail to the analytics nodes:

    1. From the Salt Master node, deploy the opencontrail state to the first of the Cassandra nodes:

        ```
        salt 'nal01*' state.sls opencontrail
        ```

        Warning

        The deployment may fail with two errors:

        - Error stating that iptables service failed. Can be ignored.
        - Error stating that net.netfilter is unavailable.

        Rerun the OpenContrail state until only a single iptables failure remains.

5. On the Salt Master node, install OpenContrail on the remaining network nodes simultaneously:

    ```
    salt 'nal*' state.sls opencontrail
    ```

6. To verify the status of services, restart the Nova API service on the controller nodes to reflect the newly installed OpenContrail dependencies from the Salt Master node:

    ```
    salt 'ctl*' service.restart nova-api
    ```

Configure OpenContrail

It may happen that your installation does not use SSH overall because of not having a certificate authority available. By default, OpenContrail uses SSL and requires certificate authentication. If you attempt to access the OpenContrail UI through the proxy with such configuration, the UI will accept your credentials but will end up in logging you out immediately. As a workaround, you can use HTTP directly to the OpenContrail UI management VIP bypassing the proxy. You can log in and perform the management functions without being logged out.

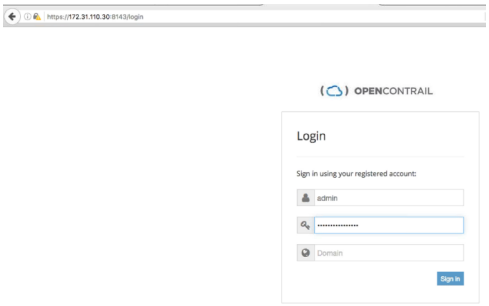
To configure OpenContrail:

1. Obtain the Administrator password.

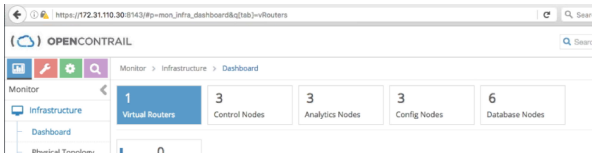
    1. On the Salt Master:

        ```
        salt 'ctl01*' cmd.run 'cat /root/keystonerc'
        ```

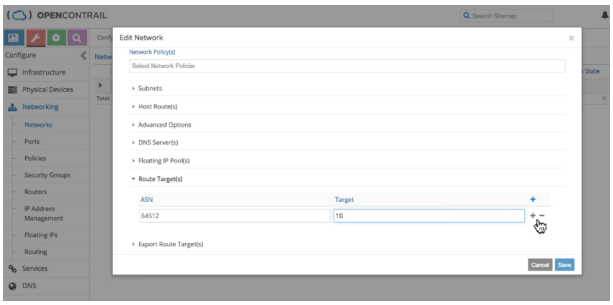
    2. Record the OS_PASSWORD. This is the Administrator password you need to pass for the OpenContrail registration.
2. Log in to the OpenContrail web UI. Use the OpenStack controller node VIP on port 8143.



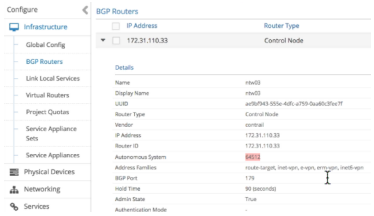
3. Verify that at this point you have the following nodes registered in OpenContrail:
   - 1 Virtual Router (cmp001)
   - 3 Control Nodes (ntw0[1-3])
   - 3 Analytics Nodes (nal03)
   - 3 Config Nodes (nwt0[1-3])
   - 6 Database Nodes (ntw0[1-3],nal0[1-3])



4. Navigate on the left to Networking > Networks.
5. Click the gear symbol for the FIPVN network and select Edit.
6. In the Edit Network window, expand Route Target(s).
7. Click the + symbol to add an ASN and target.
8. Enter the router ASN used during provisioning and 10 for the Target:



9. Click Save.
10. Verify the configuration.
    1. On the left, navigate to Infrastructure > BGP Routers.
    2. Expand one of the ntw0* controller nodes and verify that the ASN is set correctly.
       In this example: Autonomous System 64512



Deploy compute nodes

Provision a compute node

Provisioning of compute nodes 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. Ordering of dependencies is not yet orchestrated.

To provision a compute node:

1. Verify that /classes/system/reclass/storage/system/<cluster-name>_compute.yml contains correct host information.

   > Note
   >
   > In this file, create as many hosts as the number of compute nodes in your environment.

2. Run the reclass.storage state on the Salt Master node to generate the target node definitions:

   ```
   salt '*cfg*' state.sls reclass.storage
   ```

3. Verify that the target nodes have connectivity with the Salt Master:

   ```
   salt '*cmp[<num>]*' test.ping
   ```

4. Run the Salt highstate against the node(s):

   > Note
   >
   > The highstate means that all states in the node definition will be run.

> Note
>
> You may see errors during the first run of highstate.

```
salt '*cmp[<num>]*' state.highstate
```

5. Reboot the node to apply network configuration changes.
6. Re-run the Salt highstate against the nodes:

> Note
>
> If you still see some failures, run the highstate again or remediate the specific failures unless you identify them as benign.

```
salt '*cmp[<num>]*' state.highstate
```

Deploy vRouter on compute nodes

To deploy vRouter:

> Note
>
> Perform the procedure above on each cmp compute node.

1. Example of the salt command for cmp001:

```
salt "cmp001*" cmd.run '/usr/share/contrail-utils/provision_vrouter.py \
   --host_name cmp001 --host_ip 172.31.112.11 --api_server_ip 172.31.110.30 \
   --oper add --admin_user admin --admin_password 9kFLrGit63yatq3e \
   --admin_tenant_name admin --openstack_ip 172.31.110.20'
```

2. On the compute node, create a virtual gateway interface:

> Note
>
> This step is for scenarios where a simple gateway is used.

```
salt "cmp001*" cmd.run '/usr/share/contrail-utils/provision_vgw_node.py \
  --oper create --interface vgw1 --subnets 172.31.113.0/24 --routes 0.0.0.0/0 \
  --vrf default-domain:admin:FIPVN:FIPVN'
```

3. Fix errors in the OpenContrail UI for Global config → Forwarding options. Example of the command to run on the ntw01 node:

```
/usr/share/contrail-utils/encap.py add admin 9kFLrGit63yatq3e admin 172.31.110.30
```

4. Provision the linklocal service:

```
/usr/share/contrail-utils/provision_linklocal.py --admin_user admin \
    --admin_password 9kFLrGit63yatq3e --linklocal_service_name metadata \
    --linklocal_service_ip 163.154.169.254 --linklocal_service_port 80 \
    --ipfabric_service_ip 172.31.110.20 --ipfabric_service_port 8775 \
    --api_server_ip 172.31.110.30
```

5. On the compute node cmp001, restart the vRouter supervisor service:

```
service supervisor-vrouter restart
```

Deploy a compute node

To deploy a cmp001 compute node:

1. Verify that linux/system/compute.yml has valid information about the host:

    1. Change the Cookiecutter key:value names to include a three-digit number identifier.

       As noted previously, this was hardcoded for two. Therefore, some items are not populated correctly. In this case, the compute host information can be dropped.

       Before:

```
host:
      local:
        address: ${_param:single_address}
        names:
          - ${linux:system:name}.${linux:system:domain}
          - ${linux:system:name}
```

       After:

```
host:
      local:
        address: ${_param:single_address}
        names:
          - cmp001.mk.slab.local
          - cmp001
```

2. Optional. Create as many hosts as you have compute nodes in your OpenStack environment.

   Before:

   ```
   host:
       local:
         address: ${_param:single_address}
         names:
          - ${linux:system:name}.${linux:system:domain}
          - ${linux:system:name}
   ```

   After:

   ```
   host:
       cmp001:
         address: ${_param:single_address}
         names:
          - cmp001.mk.slab.local
          - cmp001
   ```

   > **Note**
   >
   > If you have added nodes after deployment, you need to add their definitions as described above. Then, run the reclass state on the Salt Master node to add them under the Salt management.

2. On the Salt Master node, sync the Salt data to each of the nodes:

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

3. Refresh the pillar data. This is not covered by sync_all and needs to be done every time you run the reclass state on the cfg node:

   ```
   salt '*' saltutil.refresh_pillar
   ```

4. Run the reclass.storage state to refresh the deployed pillar data:

   ```
   salt '*cfg*' state.sls reclass.storage
   ```

5. Verify that the new machines have connectivity with the Salt Master node:

   ```
   salt '*' test.ping
   ```

6. On the Salt Master node, run the Salt states for linux, ntp, openssh, and salt for the compute nodes:

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

7. Run the Salt highstate for the compute nodes:

```
salt 'cmp*' state.highstate
```

> Note
>
> You may need to run the highstate multiple times to get a successful deployment. If after two runs you still have errors, reboot the compute nodes and run the highstate again.

> Note
>
> You may have an error stating that iptables is down. Ignore this error.

Install StackLight

StackLight is the Logging, Metering, Alerting (LMA) toolchain, the operational health and response monitoring solution. It monitors nodes, services, and clusters health, as well as business KPIs. Moreover, StackLight provides rich operational insights out-of-the-box for the OpenStack, OpenContrail, Kubernetes, and Calico services deployed on MCP. Cloud operators can easily prevent service downtime using StackLight for monitoring as Stacklight quickly notifies users of critical conditions that may occur in the system.

Prerequisites

Before you start installing the StackLight components, complete the following steps:

1. Configure StackLight for installation.

   The configuration of StackLight for installation is defined in the Reclass model. See stacklight-salt-model as an example of the Reclass model to install StackLight 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, such as the IP address and port number of the Elasticsearch and InfluxDB servers.

- The cluster level model defines the configuration of StackLight for a particular deployment. A user Reclass model to install OpenStack with StackLight monitoring must be created. This is where you typically define the name of the InfluxDB database, username, password of the InfluxDB admin, and others.

2. Verify that you have Internet access to download several packages from external repositories that are not included in the standard Ubuntu distribution. If there is no Internet access, these repositories must be mirrored on MCP.

Install StackLight back ends

StackLight integrates several back-end servers to visualize an environment monitoring and health statuses. This section describes how to to install various back-end visualization solutions, including: InfluxDB and Grafana, Elasticsearch and Kibana, Sensu and Uchiwa.

Install InfluxDB and Grafana

The InfluxDB server must be installed on the monitoring cluster of Mirantis Cloud Platform (MCP).

Hardware requirements

InfluxDB and Grafana have the following requirements:

Requirement	Comments
Disk space	InfluxDB requires at least 15 GB of disk space for the system and 10 GB for the logs. We highly recommend installing the InfluxDB database on a dedicated disk partition. The size of the partition depends on many factors including the size of the deployment and the retention period. It has been measured that a partition of 100 GB is necessary for a deployment of about 200 compute nodes with three controller nodes and a retention period of one month.
Hardware specification	The hardware specification required for InfluxDB and Grafana depends on the size of the deployment and other factors like the retention period. A basic setup requires a quad-core CPU with 8 GB of RAM and a fast disk of 500-1000 IOPS. See the InfluxDB Hardware Sizing Guide for additional sizing information.
MCP supported versions	MCP 1.0

Limitations

StackLight has the following limitations in regards to InfluxDB and Grafana:

- InfluxDB is deployed on three nodes in the active/passive failover mode (only one node is storing data at a time).
- The current implementation of StackLight for MCP uses the latest version of the community edition, InfluxDB 1.2. However, InfluxDB 1.2 does not support clustering. If you want to

deploy InfluxDB in cluster mode for HA and scale-out, you need to use the InfluxEnterprise version.

Configure InfluxDB and Grafana

The configuration parameters of the InfluxDB and Grafana dashboard are defined in the Salt formula. For details and the configuration examples, see the GitHub projects for SaltStack InfluxDB formula and SaltStack Grafana formula.

Deploy InfluxDB and Grafana

This section describes how to deploy the InfluxDB cluster and Grafana.

To 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 will be 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, run the following command for the Salt minion nodes with the influxdb:server role defined in the Salt Pillar:

```
salt -C 'I@influxdb:server' state.sls influxdb
```

The deployment of Grafana consists of the server, the collector, and the client deployment.

To deploy Grafana:

1. Deploy the server:

   ```
   salt -C 'I@grafana:server' state.sls grafana.server -b 1
   ```

2. The collector is used to retrieve dashboards provided by different services. Services that have Grafana support enabled will generate grains and those grains will be used to fill mines.

   Run the collector before the client:

   ```
   salt -C 'I@grafana:collector' state.sls grafana.collector
   salt -C 'I@grafana:collector' state.sls salt.minion.grains
   salt -C 'I@grafana:collector' saltutil.refresh_modules
   salt -C 'I@grafana:collector' mine.update
   ```

3. Run the client that will configure Grafana:

   1. Verify that the Salt Minion is properly configured to connect to the Grafana server:

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

2. Perform the configuration:

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

Verify InfluxDB and Grafana after deployment

Depending on the number of nodes and deployment setup, deploying InfluxDB and Grafana may take up to several hours.

To verify InfluxDB:

1. Log in to the Salt Master node.
2. Run the following command:

```
/usr/bin/influx -database lma -password lmapass \
--username root -host mon -port 8086
```

Example of the system response:

```
Connected to http://mon:8086 version 1.2.0
InfluxDB shell version: 1.2.0
>
```

This example shows that executing /usr/bin/influx starts an interactive CLI and automatically connects to the InfluxDB server.

3. To see a dump of all the time-series collected, run:

```
> show series
```

Example of the system response fragment:

```
[...] cluster_status,cluster_name=cinder-control,
environment_label=lucky,member=cinder_control,nagios_host=00-top-clusters
cluster_status,cluster_name=compute,environment_label=lucky,
member=compute_nodes,nagios_host=01-node-clusters cluster_status,
cluster_name=contrail-compute,environment_label=lucky,
member=contrail_compute,nagios_host=00-top-clusters cluster_status,
cluster_name=contrail-control,environment_label=lucky,
member=contrail_control,nagios_host=00-top-clusters [...]
```

To verify Grafana:

1. Log in to the Salt Master node.
2. Enter the monitoring VIP on port 3000 by default.

3. Authenticate using your credentials following the Connect to Grafana instructions.

   You should be redirected to the Grafana Home page with a list of available dashboards sorted by name.

Install Elasticsearch and Kibana

The Elasticsearch and Kibana servers must be installed on the monitoring cluster of the Mirantis Cloud Platform.

Hardware requirements

Elasticsearch and Kibana have the following requirements:

Requirement	Comments
Disk space	Elasticsearch requires at least 15 GB of disk space for the system and 10 GB for the logs. We highly recommend installing the Elasticsearch database on a dedicated disk partition. The size of the partition depends on many factors including the size of the deployment, the retention period, and the log level. Logging at the DEBUG level requires 10 times more space than logging at the INFO level. It has been measured that a partition of 500 GB is necessary for a deployment of about 200 compute nodes with three controller nodes and a retention period of one month.
Hardware specification	The hardware specification required for Elasticsearch depends on the size of the deployment and other factors like the retention period and logging activity. A basic setup requires a quad-core CPU with 4 GB of RAM and a fast disk of 500-1000 IOPS. For larger deployments, we recommend having 8 GB of RAM or 50% of the available memory up to 32 GB maximum for the JVM heap.
MCP supported versions	MCP 1.0

Limitations

StackLight has the following limitations in regards to Elasticsearch and Kibana:

- The cluster must be installed on minimum three nodes to avoid split-brain issues.
- The advanced cluster operations may require manual steps. For details, see Advanced operations.

Configure Elasticsearch and Kibana

The configuration parameters of the Elasticsearch engine and Kibana dashboard are defined in the Salt formula. For details and the configuration examples, see the GitHub projects for SaltStack Elasticsearch formula and SaltStack Kibana formula.

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 server that will install Elasticsearch and Kibana:

```
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 client that will configure the server:

    1. Configure the client to communicate with the server:

    ```
    salt -C 'I@elasticsearch:client' state.sls elasticsearch.client.service
    salt -C 'I@kibana:client' state.sls kibana.client.service
    ```

    2. Restart the Minion on the nodes where clients are running:

    ```
    salt -C 'I@elasticsearch:client or I@kibana:client' --async service.restart salt-minion
    ```

    3. Configure the server itself:

    ```
    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 the Salt Master node.
2. Run the following command:

```
curl http://mon:9200
```

Example of the system response:

```
curl http://mon:9200
{
  "name" : "mon01",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "KJM5s5CkTNKGkhd807gcCg",
  "version" : {
    "number" : "2.4.4",
    "build_hash" : "fcbb46dfd45562a9cf00c604b30849a6dec6b017",
    "build_timestamp" : "2017-01-03T11:33:16Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.2"
  },
  "tagline" : "You Know, for Search"
}
```

To verify the Kibana dashboard

To verify that the Kibana dashboard is working, use the monitoring VIP and the default port 5601. No credentials are required.

You should be redirected to the Kibana Logs Analytics dashboard with six sections of logs:

1. Log messages over time per source
2. Log messages over time per severity
3. Number of log messages per severity
4. Top 10 sources
5. Top 10 programs
6. Top 10 hosts

Install Sensu

The Sensu server must be installed on the monitoring cluster of Mirantis Cloud Platform (MCP).

Hardware requirements

Sensu has the following requirements:

Requirement	Comments
Disk space	Sensu requires at least 15 GB of disk space for the system, 10 GB for the logs, and 20 GB for Sensu itself.
Hardware specification	A basic setup at least requires a quad-core CPU with 8 GB of RAM and a fast disk.
MCP supported versions	MCP 1.0

Configure Sensu

The configuration parameters of the Sensu engine and Uchiwa dashboard are defined in the Salt formula. For details and configuration examples, see the GitHub project for SaltStack Sensu formula.

Deploy Sensu

Sensu is an infrastructure and application monitoring and telemetry solution that is widely used across the industry. Sensu provides a general purpose framework for monitoring infrastructure, service and application health, and business KPIs.

In the StackLight architecture for MCP, the use of Sensu is dual:

- It monitors the components of the StackLight operational insight pipeline itself (collectd, Local Metric Collector, Remote Metric Collector, and Aggregator).
- It handles the StackLight health status events for alerting and escalation.

The Sensu cluster must be installed on minimum three nodes to avoid split-brain issues.

To deploy Sensu

Run the following command for the Salt Minions:

```
salt -C 'I@sensu:server and I@rabbitmq:server' state.sls rabbitmq
salt -C 'I@sensu:server and I@rabbitmq:server' cmd.run "rabbitmqctl cluster_status"
salt -C 'I@redis:cluster:role:master' state.sls redis
salt -C 'I@redis:server' state.sls redis
salt -C 'I@sensu:server' state.sls sensu -b 1
salt -C 'I@sensu:client' state.sls sensu
```

This will install the Sensu cluster on the monitoring cluster and the Sensu client on all nodes.

Verify Sensu and Uchiwa

After you deploy Sensu, verify that it up and running using the steps below.

To verify Sensu:

1. Log in to the Salt Master node.
2. Run the following command:

   ```
   salt -C 'I@sensu:server' service.status sensu-server
   ```

To verify the Uchiwa dashboard:

To verify that the Uchiwa dashboard is working, use the monitoring VIP and the default port 3001. The Sensu dashboard username is admin. The password can be retrieved from Salt Pillar data:

```
salt ‘mon*’ pillar.data _param:sensu_dashboard_password
```

Seealso

Sensu official documentation <https://sensuapp.org/docs/latest/overview/what-is-sensu.html>

Install StackLight operational insights pipeline

After you install back ends for StackLight as described in the Install StackLight back ends section, proceed with installing the StackLight operational insights pipeline (SOIP) that includes the Log Collector, the Local Metric Collector, the Remote Metric Collector, and the Aggregator.

Install SOIP

This section describes how to install the StackLight operational insights pipeline (SOIP). All steps in this section must be executed from the Salt Master node.

To install SOIP:

1. Log in to the Salt Master node.
2. Restart salt-minion to make sure that it uses the latest Jinja library:

```
salt '*' --async service.restart salt-minion
```

3. Clean the Salt mine:

```
salt "*" mine.flush
```

4. Clean the grains files to make sure that you start from a clean state:

```
salt "*" file.remove /etc/salt/grains.d/collectd
salt "*" file.remove /etc/salt/grains.d/grafana
salt "*" file.remove /etc/salt/grains.d/heka
salt "*" file.remove /etc/salt/grains.d/sensu
salt "*" file.remove /etc/salt/grains
```

5. Install the collectd and heka services on the nodes:

```
salt "*" state.sls collectd
salt "*" state.sls heka
```

6. Update the Salt mine:

```
salt "*" state.sls salt.minion.grains
salt "*" saltutil.refresh_modules
salt "*" mine.update
```

7. Update Heka:

```
salt -C 'I@heka:aggregator:enabled:True or \
I@heka:remote_collector:enabled:True' state.sls heka
```

8. Update collectd:

```
salt -C 'I@collectd:remote_client:enabled:True' state.sls collectd
```

9. Update Sensu:

```
salt -C 'I@sensu:server' state.sls sensu
```

10. Retrieve the StackLight monitoring VIP from Salt pillar:

```
salt-call pillar.data _param:stacklight_monitor_address --out key
```

Example of system response:

```
Local Keys:
_param:stacklight_monitor_address: 172.16.10.253
```

11 Restart the services that are bound to the monitoring VIP:

.

```
export vip=<MONITORING_VIP>
salt -G "ipv4:$vip" service.restart remote_collectd
salt -G "ipv4:$vip" service.restart remote_collector
salt -G "ipv4:$vip" service.restart aggregator
```

Seealso

- Verify SOIP

Verify SOIP

After you install the StackLight operational insights pipeline (SOIP) components, verify that they are up and running on the targeted Salt Minion nodes.

To verify the Log Collector:

1. Log in to the Salt Master node.
2. Verify that the Log Collector status is True on all the Salt Minion nodes with names that start with ctl:

   ```
   salt 'ctl*' service.status log_collector
   ```

   Example of the system response:

   ```
   ctl03.mcp-lab-advanced.local:
     True
   ctl01.mcp-lab-advanced.local:
     True
   ctl02.mcp-lab-advanced.local:
     True
   ```

To verify the Metric Collector:

1. Log in to the Salt Master node.
2. Verify that the Metric Collector status is True on all the Salt Minion nodes with names that start with ctl:

   ```
   salt 'ctl*' service.status metric_collector
   ```

   Example of the system response:

```
ctl03.mcp-lab-advanced.local:
  True
ctl01.mcp-lab-advanced.local:
  True
ctl02.mcp-lab-advanced.local:
  True
```

To verify the Remote Metric Collector:

1. Log in to the Salt Master node.
2. Verify that the Remote Metric Collector status is True on the Salt Minion node with the monitoring VIP:

   ```
   salt 'mon*' service.status remote_collector
   ```

   Only the node with the monitoring VIP should have the status True.

   Example of the system response:

   ```
   mon-01.mcp-lab-advanced.local:
     False
   mon-02.mcp-lab-advanced.local:
     False
   mon-03.mcp-lab-advanced.local:
     True
   ```

   In this example, the Salt Minion node with the monitoring VIP mon-03 successfully runs the Remote Metric Collector since its status is True.

To verify the Aggregator:

1. Log in to the Salt Master node.
2. Verify that the Aggregator status is True on the Salt Minion node with the monitoring VIP:

   ```
   salt 'mon*' service.status aggregator
   ```

   > Note
   >
   > Only the node with the monitoring VIP should have the status True.

   Example of the system response:

   ```
   mon-01.mcp-lab-advanced.local:
     False
   mon-02.mcp-lab-advanced.local:
     False
   ```

```
mon-03.mcp-lab-advanced.local:
    True
```

In the example above, the Salt Minion node with the monitoring VIP mon-03 successfully runs the Aggregator since its status is True.

Install Horizon plugins

The Horizon Telemetry and Monitoring plugins must be installed on the proxy cluster of Mirantis Cloud Platform (MCP).

The Horizon Telemetry and Monitoring plugins are deployed and configured by the Horizon formula. To deploy plugins, deploy Horizon following the Deploy Horizon procedure.

To verify the Horizon plugins after deployment

After you successfully deploy Horizon with the Telemetry and Monitoring plugins, verify that the Telemetry and Monitoring tabs are present in the Horizon dashboard.

To connect to the Horizon plugins dashboard

The Horizon dashboard with the Telemetry and Monitoring plugins can be reached through the proxy VIP using the HTTPS protocol. To access the Horizon dashboard, use your OpenStack credentials.

Seealso

The Logging, metering, alerting section in the MCP Operations Guide

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 OpenContrail

This section includes the workarounds for the following issues:

Troubleshoot Cassandra

Example of system response of the contrail-status command indicating the issue:

```
== Contrail Database ==
supervisor-database:         active
contrail-database            active
contrail-database-nodemgr     initializing (Cassandra state detected
                         DOWN.Disk space for analytics db not
                         retrievable.)
kafka                        active
```

Workaround:

1. Remove all files from the /var/lib/cassandra/ folder:

```
rm -rf /var/lib/cassandra/*
```

2. Restart the service:

```
service supervisor-database restart
```

3. Check the status:

```
contrail-status
```

Troubleshoot the database connectivity

Example of system response of the contrail-status command indicating the issue:

```
== Contrail Analytics ==
supervisor-analytics:         active
contrail-alarm-gen            active
contrail-analytics-api        active
contrail-analytics-nodemgr    active
contrail-collector            initializing (Database:ctl02:Global connection down)
contrail-query-engine         timeout
contrail-snmp-collector       active
contrail-topology             active
```

Workaround:

1. Restart the supervisor-analytics service:

   ```
   service supervisor-analytics restart
   ```

2. Check the neutron API:

   ```
   neutron net-list
   +----------------------+------------------------+---------+
   | id                   | name                   | subnets |
   +----------------------+------------------------+---------+
   | d6638b91-4e1d-4214-... | ip-fabric              |         |
   | ce66ee12-71b4-44ea-... | __link_local__         |         |
   | d452af3a-3b9f-442e-... | default-virtual-network |         |
   +----------------------+------------------------+---------+
   ```

3. Restart nova-api to reflect installed OpenContrail dependencies:

   ```
   salt 'ctl*' service.restart nova-api
   ctl02.workshop.cloudlab.cz:
       True
   ctl03.workshop.cloudlab.cz:
       True
   ctl01.workshop.cloudlab.cz:
       True
   ```

4. Check the nova list command system response:

   ```
   nova list
   ```

Now, OpenStack and OpenContrail controllers are properly deployed.

To troubleshoot OpenContrail, verify the OpenContrail status using the contrail-status command.

> Note
>
> Ignore the vRouter failure. The vRouter agent is located on the compute node only.

Example of system response:

```
contrail-status
vRouter is NOT PRESENT

== Contrail vRouter ==
supervisor-vrouter: unrecognized service
supervisor-vrouter:           inactive
contrail-vrouter-agent        failed

== Contrail Control ==
supervisor-control:           active
contrail-control              active
contrail-control-nodemgr      active
contrail-dns                  active
contrail-named                active

== Contrail Analytics ==
supervisor-analytics:         active
contrail-analytics-api        active
contrail-analytics-nodemgr    active
contrail-collector            active
contrail-query-engine         active
contrail-snmp-collector       active
contrail-topology             active

== Contrail Config ==
supervisor-config:            active
contrail-api:0                active
contrail-config-nodemgr       active
contrail-device-manager       active
contrail-discovery:0          active
contrail-schema               initializing
contrail-svc-monitor          initializing
ifmap                         active

== Contrail Web UI ==
supervisor-webui:             active (disabled on boot)
contrail-webui                active
contrail-webui-middleware     active

== Contrail Database ==
supervisor-database:          active
```

```
contrail-database           active
contrail-database-nodemgr   active
```

Deploy a Kubernetes cluster manually

Kubernetes is the system for containerized applications automated deployment, scaling, and management. This section guides you through the deployment procedure for a Kubernetes cluster.

Prerequisites

Generic hardware requirements for a Kubernetes cluster deployment include six nodes:

1x Configuration node
: A host for the Salt Master node. Can be a virtual machine

3x Controller nodes
: Hosts for the Kubernetes control plane components and ETCD

2x Compute nodes
: Hosts for the Kubernetes pods, groups of containers that are deployed together on the same host

For easier deployment and testing, the following usage of 3 NICs is recommended:

- 1x NIC as a PXE/DHCP/Salt network (PXE and DHCP is a third-party service in DC) unmanaged by SaltStack
- 2x NICs as bond active-passive or active-active with two 10Gbit slave interfaces

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

Generate a Kubernetes cluster metadata model

This section covers how to generate the metadata model for the MCP Kubernetes cluster deployment.

To generate the Kubernetes metadata model:

1. Create a Git repository that will store your Kubernetes cluster configurations. To prepare the Git repository, select one of the options below:

   - Create a project repository manually.
   - Prepare a Git repo automatically using the Jenkins pipeline.

2. Clone the kubernetes_mk template from the MCP templates repository as described in get-cookiecutter-templates.

3. Define etcd members under Salt formulas level metadata.

   In the majority of cases, the Kubernetes cluster deployment includes 3 etcd members for Kubernetes. The cluster_node_address and cluster_node_hostname parameters are commonly used through all Salt formulas. Define them once to use for all roles through your deployment:

   ```
   parameters:
     kubernetes:
       master:
         etcd:
           members:
             - host: ${_param:cluster_node01_address}
               name: ${_param:cluster_node01_hostname}
             - host: ${_param:cluster_node02_address}
               name: ${_param:cluster_node02_hostname}
             - host: ${_param:cluster_node03_address}
               name: ${_param:cluster_node03_hostname}
   ```

4. Define the networking plugin under the system level metadata.

   The default networking plugin for the MCP deployment is Calico. To define Calico, edit the reclass-system-salt-model/kubernetes/master/cluster.yml:

   ```
   parameters:
     kubernetes:
       master:
         network:
           engine: calico
           private_ip_range: ${_param:kubernetes_private_net_cidr}
   ```

5. Define the cluster level metadata for your Kubernetes deployment.

6. Copy the cookiecutter.json file from the template repository:

   ```
   cd $RECLASS_REPO
   cp cookiecutter.$ENV_NAME.json cookiecutter.json
   ```

7. Update new environment definition in cookiecutter.json to fit your deployment.

8. Generate the Kubernetes cluster level metadata definition:

```
cookiecutter ~/.cookiecutters/kubernetes_mk --output-dir \
    ./classes/cluster --no-input -f
```

The name of the cluster is defined by the project_name parameter.

9. Push the changes to the Git repository as described in Publish the deployment model to a project repository.

Now, the Kubernetes metadata model is ready to be used on the Salt Master node.

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, add the following example definition to the {{ cookiecutter.cluster_name }}/kubernetes/init.yml file:

```
parameters:
...
 _param:
   deploy_nic: eth0
   primary_first_nic: eth1
   primary_second_nic: eth2
 linux:
   ...
   network:
   ...
     interface:
       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

This section explains how you can deploy your Kubernetes cluster.

To deploy the Kubernetes cluster:

1. Set up the Salt Master node as described in Install the Salt Master node.
2. Update modules and states on all Minions:

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

3. Create and distribute SSL certificates for services using the salt state:

   ```
   salt "*" state.sls salt
   ```

4. Install Keepalived:

   ```
   salt -C 'I@keepalived:cluster' state.sls keepalived -b 1
   ```

5. Install HAProxy:

   ```
   salt -C 'I@haproxy:proxy' state.sls haproxy
   salt -C 'I@haproxy:proxy' service.status haproxy
   ```

6. Install Docker:

```
salt -C 'I@docker:host' state.sls docker.host
salt -C 'I@docker:host' cmd.run "docker ps"
```

7. Install etcd:

```
salt -C 'I@etcd:server' state.sls etcd.server.service
salt -C 'I@etcd:server' cmd.run "etcdctl cluster-health"
```

   Install etcd with the SSL support:

```
salt -C 'I@etcd:server' state.sls salt.minion.cert,etcd.server.service
salt -C 'I@etcd:server' cmd.run '. /var/lib/etcd/configenv && etcdctl cluster-health'
```

8. Install Kubernetes and Calico:

```
salt -C 'I@kubernetes:master' state.sls kubernetes.master.kube-addons
salt -C 'I@kubernetes:pool' state.sls kubernetes.pool
salt -C 'I@kubernetes:pool' cmd.run "calicoctl node status"
```

9. Set up NAT for Calico:

```
salt -C 'I@kubernetes:master' state.sls etcd.server.setup
```

10. Run master to check consistency:

```
salt -C 'I@kubernetes:master' state.sls kubernetes exclude=kubernetes.master.setup
```

11. Register addons:

```
salt -C 'I@kubernetes:master' --subset 1 state.sls kubernetes.master.setup
```

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.

NFV for MCP compatibility matrix

Type	Host OS	Kernel	HugePages	DPDK	SR-IOV	NUMA	CPU pinning	Multiqueue
OVS	Xenial	4.8	Yes	No	Yes	Yes	Yes	Yes
Kernel vRouter	Xenial	4.8	Yes	No	Yes	Yes	Yes	Yes
DPDK vRouter	Trusty	4.4	Yes	Yes	No	Yes	Yes	No (version 3.2)
DPDK OVS	Xenial	4.8	Yes	Yes	No	Yes	Yes	Yes

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-pci
```

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
```

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.

To enable OVS DPDK:

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 dpdk0 interface. Select from the following options:

   - Single interface NIC dedicated for DPDK:

```
...
- 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

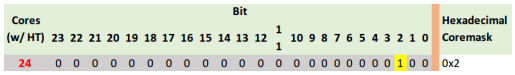
```
...
- system.neutron.compute.nfv.dpdk
...
parameters:
  linux:
    network:
     interfaces:
        ...
        # other interface setup
        ...
       dpdk0:
          name: ${_param:dpdk0_name}
          pci: ${_param:dpdk0_pci}
          driver: igb_uio
          bond: dpdkbond1
```

```
      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
    dpdkbond1:
      enabled: true
      bridge: br-prv
      type: dpdk_ovs_bond
      mode: active-backup
    br-prv:
      enabled: true
      type: dpdk_ovs_bridge
```

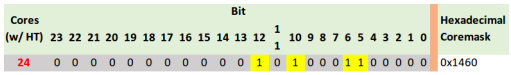
3. Calculate the hexadecimal coremask.

   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 lcores 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

```
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. 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: xenial
    dpdk0_name: enp5s0f1
    dpdk1_name: enp5s0f2
    dpdk0_pci: '"0000:05:00.1"'
    dpdk1_pci: '"0000:05:00.2"'
```

6. If the VXLAN neutron tenant type is selected, set the local IP address on br-prv for VXLAN tunnel termination:

```
...
- 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
```

7. 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.

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: xenial
    dpdk0_name: enp5s0f1
    dpdk1_name: enp5s0f2
    dpdk0_pci: '"0000:05:00.1"'
    dpdk1_pci: '"0000:05:00.2"'
    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"
```

Enable OpenContrail DPDK

OpenContrail 3.1.1 uses DPDK libraries version 2.1. Therefore, it is supported only on Ubuntu Trusty with kernel 3.13 =< 4.4.

> 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

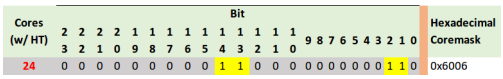
  Set of amount HugePages in Megabytes to be used by vRouter-DPDK taken for each NUMA node. Set size is equal to NUMA nodes count, elements are divided by comma

- 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.)



Cores (w/ HT)	Bit 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Hexadecimal Coremask
24	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0	0x6006

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 Mirantis 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:

```
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

```
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:

```
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

```
...
  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:
      kernel:
        sriov: True
        unsafe_interrupts: False
      rc:
        local: |
          #!/bin/sh -e
          # Enabling 7 VFs on eth1 PF
          echo 7 > /sys/class/net/eth1/device/sriov_numvfs; sleep 2; ip link set eth1 up
          # Enabling 15 VFs on eth2 PF
          echo 15 > /sys/class/net/eth2/device/sriov_numvfs; sleep 2; ip link set eth2 up
          # Enabling 15 VFs on eth3 PF
          echo 15 > /sys/class/net/eth3/device/sriov_numvfs; sleep 2; ip link set eth3 up
          # Enabling 7 VFs on eth4 PF
          echo 7 > /sys/class/net/eth4/device/sriov_numvfs; sleep 2; ip link set eth4 up
          exit 0
```

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 sriovnicswitch 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:

```
- system.neutron.control.openvswitch.sriov
```

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
   ```

Seealso

- Using SR-IOV functionality in the official OpenStack documentation

Seealso

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 exaplains how to configure the support for the Huge Pages feature in your MCP depoyment.

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:

```
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
```

Seealso

- 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

```
+----------------------------+-----------------------------+
| Property                   | Value                       |
+----------------------------+-----------------------------+
| OS-FLV-DISABLED:disabled   | False                       |
| OS-FLV-EXT-DATA:ephemeral  | 0                           |
| disk                       | 4                           |
| extra_specs                | {"hw:mem_page_size": "2048"} |
| id                         | 7                           |
| name                       | huge                        |
| os-flavor-access:is_public | True                        |
| ram                        | 1024                        |
| rxtx_factor                | 1.0                         |
| swap                       |                             |
| vcpus                      | 1                           |
+----------------------------+-----------------------------+
```

4. 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:

   ```
   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:

```
salt "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

```
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:

```
. 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:

```
numactl -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 pining:

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:

```
| extra_specs  | {"hw:numa_cpus.0": "0,2", "hw:numa_cpus.1": "1,3", \
"hw:numa_nodes": "2", "hw:numa_mem.1": "700", "hw:numa_mem.0": "324"} |
```

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:

   ```
   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, run:

      ```
      ssh-keygen
      ```

   3. Add the key pair to Nova:

      ```
      nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp
      ```

5. Boot a new instance using the created flavor:

   ```
   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:

   ```
   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
```

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: 1 3
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 vrouters**

  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:

```
...
- 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:

```
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>
```

Install Decapod

Decapod is a tool that simplifies the deployment and lifecycle management of Ceph clusters. This section guides you through the process of installation and configuration of Decapod.

Prerequisites

You can build Decapod on any commodity node that has Linux or OS X. However, prior to installing Decapod, verify that your software configurations meet the following requirements:

1. Install git and make.
2. Install Docker Engine as described in Install Docker Engine. Pay attention to the DNS configuration.
3. Install Docker Compose version 1.6 or later as described in Install Docker Compose.
4. Verify that your machine has access to the external network.

Install Decapod

The installation procedure contains the following steps:

1. Building the Decapod images.
2. Moving the Docker images to the target node.
3. Configuring Docker Compose.
4. Running the Docker containers.
5. Running migrations. If you run Decapod for the first time or upgrade from a previous version, apply migrations. This operation is idempotent and you may execute it safely at any time. Decapod does not reapply already applied migrations. On the first boot, migrations are required to obtain the root user. Otherwise, Decapod starts with an empty database and, therefore, without the capability to perform any operations.

Before you install Decapod, verify that you have completed the tasks described in Prerequisites.

To install Decapod:

1. Clone the source code repository:

```
$ git clone --recurse-submodules \
  https://github.com/Mirantis/ceph-lcm.git decapod
$ cd decapod
```

2. In the repository, check the available versions using Git tag. To select a specific version:

```
git checkout {tag} && git submodule update --init --recursive
```

3. Build the Decapod images.

    1. Copy the repository files to the top level directory:

```
make copy_example_keys
```

> Note
>
> The copy_example_keys target allows the build process to override the default Ubuntu and Debian repositories.

2. Build the images:

```
$ make build_containers
```

> **Important**
>
> If you have no access to the private repository to fetch base images, use the community repository:
>
> ```
> $ make docker_registry_use_dockerhub
> ```
>
> To switch back:
>
> ```
> $ make docker_registry_use_internal_ci
> ```

4. Move the Docker images to the target node.

> Note
>
> In this release, only one machine with Docker and Docker compose is supported. There may be one build machine and another production one. If you have such a diversity, use the Docker registry to manage Decapod images or dump/load them manually.

Use the following commands to dump Docker images, copy to a remote host, and load them:

```
$ make dump_images
$ rsync -a output/images/ <remote_machine>:images/
$ scp docker-compose.yml <remote_machine>:docker-compose.yml
$ ssh <remote_machine>
```

```
$ cd images
$ for i in $(\ls -1 *.bz2); do docker load -i "$i"; done;
$ cd ..
$ docker-compose up
```

5. Configure Docker Compose as described in Configure Docker Compose and Configuration files subsection in the Manage Ceph clusters using Decapod section of MCP Operations Guide.
6. Run Docker Compose:

   ```
   $ docker-compose up
   ```

   To daemonize the process:

   ```
   $ docker-compose up -d
   ```

   To stop the detached process:

   ```
   $ docker-compose down
   ```

   For details, see Overview of the Docker Compose CLI.
7. If you run Decapod for the first time or upgrade from a previous version, run migrations.

   Example:

```
$ docker-compose exec admin decapod-admin migration apply
2017-02-06 11:11:48 [DEBUG   ] (       lock.py:118 ): Lock applying_migrations was acquire by locker 76
2017-02-06 11:11:48 [DEBUG   ] (       lock.py:183 ): Prolong thread for locker applying_migrations of lo
2017-02-06 11:11:48 [INFO    ] (  migration.py:123 ): Run migration 0000_index_models.py
2017-02-06 11:11:48 [INFO    ] (  migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:11:53 [DEBUG   ] (       lock.py:164 ): Lock applying_migrations was proloned by locker 7
2017-02-06 11:11:56 [INFO    ] (  migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:11:56 [INFO    ] (  migration.py:277 ): Save result of 0000_index_models.py migration (r
2017-02-06 11:11:56 [INFO    ] (  migration.py:123 ): Run migration 0001_insert_default_role.py
2017-02-06 11:11:56 [INFO    ] (  migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:11:58 [INFO    ] (  migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:11:58 [INFO    ] (  migration.py:277 ): Save result of 0001_insert_default_role.py migrati
2017-02-06 11:11:58 [INFO    ] (  migration.py:123 ): Run migration 0002_insert_default_user.py
2017-02-06 11:11:58 [INFO    ] (  migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:11:58 [DEBUG   ] (       lock.py:164 ): Lock applying_migrations was proloned by locker 7
2017-02-06 11:11:59 [INFO    ] (  migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:11:59 [INFO    ] (  migration.py:277 ): Save result of 0002_insert_default_user.py migrat
2017-02-06 11:11:59 [INFO    ] (  migration.py:123 ): Run migration 0003_native_ttl_index.py
2017-02-06 11:11:59 [INFO    ] (  migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:12:00 [INFO    ] (  migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
```

```
2017-02-06 11:12:00 [INFO    ] (   migration.py:277 ): Save result of 0003_native_ttl_index.py migration
2017-02-06 11:12:00 [INFO    ] (   migration.py:123 ): Run migration 0004_migrate_to_native_ttls.py
2017-02-06 11:12:00 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:12:02 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:12:02 [INFO    ] (   migration.py:277 ): Save result of 0004_migrate_to_native_ttls.py mig
2017-02-06 11:12:02 [INFO    ] (   migration.py:123 ): Run migration 0005_index_cluster_data.py
2017-02-06 11:12:02 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:12:03 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:12:03 [INFO    ] (   migration.py:277 ): Save result of 0005_index_cluster_data.py migrati
2017-02-06 11:12:03 [INFO    ] (   migration.py:123 ): Run migration 0006_create_cluster_data.py
2017-02-06 11:12:03 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:12:03 [DEBUG   ] (       lock.py:164 ): Lock applying_migrations was proloned by locker 7
2017-02-06 11:12:04 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:12:04 [INFO    ] (   migration.py:277 ): Save result of 0006_create_cluster_data.py migrat
2017-02-06 11:12:04 [INFO    ] (   migration.py:123 ): Run migration 0007_add_external_id_to_user.py
2017-02-06 11:12:04 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.5/dist-packages/decapo
2017-02-06 11:12:06 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/dist-packages/decapod_ad
2017-02-06 11:12:06 [INFO    ] (   migration.py:277 ): Save result of 0007_add_external_id_to_user.py m
2017-02-06 11:12:06 [DEBUG   ] (       lock.py:202 ): Prolong thread for locker applying_migrations of lo
2017-02-06 11:12:06 [DEBUG   ] (       lock.py:124 ): Try to release lock applying_migrations by locker 7
2017-02-06 11:12:06 [DEBUG   ] (       lock.py:140 ): Lock applying_migrations was released by locker 7
```

For a list of applied migrations, use the list all option.

Example:

```
$ docker-compose exec admin decapod-admin migration list all
[applied]    0000_index_models.py
[applied]    0001_insert_default_role.py
[applied]    0002_insert_default_user.py
[applied]    0003_native_ttl_index.py
[applied]    0004_migrate_to_native_ttls.py
[applied]    0005_index_cluster_data.py
[applied]    0006_create_cluster_data.py
[applied]    0007_add_external_id_to_user.py
```

For details on a certain migration, use the show option.

Example:

```
$ docker-compose exec admin decapod-admin migration show 0006_create_cluster_data.py
Name:          0006_create_cluster_data.py
Result:        ok
Executed at:    Mon Feb  6 11:12:04 2017
SHA1 of script: 73eb7adeb1b4d82dd8f9bdb5aadddccbcef4a8b3

-- Stdout:
Migrate 0 clusters.
```

```
-- Stderr:
```

8. Reset the root user password.

    1. Obtain the user ID:

    ```
    $ docker-compose exec admin decapod user get-all
    ```

    Example:

    ```
    $ docker-compose exec admin decapod user get-all
    [
      {
        "data": {
          "email": "noreply@example.com",
          "full_name": "Root User",
          "login": "root",
          "role_id": "4ca555d3-24fd-4554-9b4b-ca44bac45062"
        },
        "id": "e6f28a01-ee7f-4ac8-b1ee-a1a21c3eb182",
        "initiator_id": null,
        "model": "user",
        "time_deleted": 0,
        "time_updated": 1488279856,
        "version": 1
      }
    ]
    ```

    2. Change the password:

    ```
    $ decapod-admin password-reset USER_ID
    ```

Configure Docker Compose

To configure Docker Compose, modify the docker-compose.override.yml file or set the environment variables. Use the official Docker documentation and the information below.

The Decapod Docker Compose configuration supports a number of environment variables. For a list of variables, see the .env file at the top level of the repository. The defaults are applicable for a development environment built on a local machine and have to be modified to run in production:

Environment variable	Default value	Description

DECAPOD_HTTP_PORT	9999	The port to bind the HTTP endpoint of Decapod.
DECAPOD_HTTPS_PORT	10000	The port to bind the HTTPS endpoint of Decapod.
DECAPOD_REGISTRY_URL		By default, Decapod tries to access local images. To take images from a private registry, point it here.
DECAPOD_NAMESPACE		In private registries, Decapod images are not always created without a prefix, sometimes the organization name, like mirantis, is present. The variable sets this prefix.
DECAPOD_VERSION	latest	The Decapod version to use. This is the image tag that is set in the registry. The latest tag means developer version.
DECAPOD_SSH_PRIVATE_KEY	$(pwd)/containerization/files/devconfigs/ansible_ssh_keyfile.pem	A full path to the SSH private key that Ansible uses to access Ceph nodes.

Default configuration example:

```
networks: {}
services:
  api:
    image: decapod/api:latest
    links:
    - database
```

```
    restart: on-failure:5
  controller:
    image: decapod/controller:latest
    links:
    - database
    restart: on-failure:5
    volumes:
    - /vagrant/containerization/files/devconfigs/ansible_ssh_keyfile.pem:/root/.ssh/id_rsa:ro
  cron:
    image: decapod/cron:latest
    links:
    - database
    restart: on-failure:3
  database:
    image: decapod/db:latest
    restart: always
    volumes_from:
    - service:database_data:rw
  database_data:
    image: decapod/db-data:latest
    volumes:
    - /data/db:rw
  frontend:
    image: decapod/frontend:latest
    links:
    - api
    - cron
    ports:
    - 10000:443
    - 9999:80
    restart: always
version: '2.0'
volumes: {}
```

For example, to set docker-prod-virtual.docker.mirantis.net as a registry and mirantis/ceph as a namespace and run version 0.2, execute docker compose with the following environment variables:

```
$ DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/ \
DECAPOD_NAMESPACE=mirantis/ceph/ DECAPOD_VERSION=0.2 docker-compose config
networks: {}
services:
  api:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/api:0.2
    links:
    - database
    restart: on-failure:5
  controller:
```

```
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/controller:0.2
    links:
    - database
    restart: on-failure:5
    volumes:
    - /vagrant/containerization/files/devconfigs/ansible_ssh_keyfile.pem:/root/.ssh/id_rsa:ro
  cron:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/cron:0.2
    links:
    - database
    restart: on-failure:3
  database:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db:0.2
    restart: always
    volumes_from:
    - service:database_data:rw
  database_data:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db-data:0.2
    volumes:
    - /data/db:rw
  frontend:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/frontend:0.2
    links:
    - api
    - cron
    ports:
    - 10000:443
    - 9999:80
    restart: always
version: '2.0'
volumes: {}
```

Important

The trailing slash in DECAPOD_REGISTRY_URL and DECAPOD_NAMESPACE is required due to the limitations of the Docker Compose configuration file.

Note

Docker Compose supports reading the environment variables from the .env file, which should be placed in the same directory as the docker-compose.yml file. For more information, see the Docker documentation.

Example:

Configuration:

- The default Mirantis registry for Decapod and the latest version of Decapod
- The private SSH key for Ansible is placed in /keys/ansible_ssh_keyfile.pem
- The Decapod HTTP port is 80 and the HTTP port is 443

The .env file should look as follows:

```
DECAPOD_NAMESPACE=mirantis/ceph/
DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/
DECAPOD_VERSION=latest
DOCKER_HTTP_PORT=80
DOCKER_HTTPS_PORT=443
DOCKER_SSH_PRIVATE_KEY=/keys/ansible_ssh_keyfile.pem
```

Alternatively, set the environment variables explicitly:

```
$ export DECAPOD_NAMESPACE=mirantis/ceph/
$ export DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/
$ export DECAPOD_VERSION=latest
$ export DOCKER_HTTP_PORT=80
$ export DOCKER_HTTPS_PORT=443
$ export DOCKER_SSH_PRIVATE_KEY=/keys/ansible_ssh_keyfile.pem
$ docker-compose config
networks: {}
services:
  api:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/api:latest
    links:
    - database
    restart: on-failure:5
  controller:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/controller:latest
    links:
    - database
    restart: on-failure:5
    volumes:
    - /keys/ansible_ssh_keyfile.pem:/root/.ssh/id_rsa:ro
  cron:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/cron:latest
    links:
    - database
    restart: on-failure:3
  database:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db:latest
    restart: always
    volumes_from:
```

```
    - service:database_data:rw
  database_data:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db-data:latest
    volumes:
    - /data/db:rw
  frontend:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/frontend:latest
    links:
    - api
    - cron
    ports:
    - 443:443
    - 80:80
    restart: always
version: '2.0'
volumes: {}
```

Seealso

Configuration files in the Manage Ceph clusters using Decapod section of MCP Operations Guide