MCP Operations 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

This book provides information about designing and operating MCP clouds.

Definitions

MCP is a deployment and lifecycle management (LCM) tool that enables DevOps engineers to deploy Mirantis OpenStack and then update software and configuration through continuous integration and continuous delivery.

SaltStack is an orchestration and configuration platform that implements the model-driven architecture (MDA) that you can use to turn services configuration described using Reclass models and Salt Formulas into actual services running on nodes.

Reclass (recursive external node classifier) is a cluster level representation of the configuration metadata model.

Seealso

SaltStack Components

Provision hardware

MCP installations can use the Ubuntu's Metal-as-a-Service (MAAS) or Ironic service to handle physical node lifecycle management. The guide describes only MAAS as a bare metal provisioning service.

As such, MAAS requires an IPMI user to manage power state. This should be configured as part of the installation process.

MaaS provides DHCP to the network(s) on which compute nodes reside. Compute nodes then perform PXE boot from the network and you can configure MaaS how to provision those PXE booted nodes.

Reinstall MAAS

If your MAAS instance is lost or broken, you can reinstall it. This section describes how to install MAAS from the Ubuntu Server 16.04 ISO image.

To reinstall MAAS:

1. Download a Ubuntu Server 16.04 LTS ISO image.
2. Create a VM with virt-install to boot from the Ubuntu ISO image. For example:

   ```
   virt-install -n maas-vm -r 2048 --os-type=linux \
             --disk /path/to/images/maas.img,device=disk,bus=virtio,\
             size=40,sparse=true,format=raw -w bridge=br0,model=virtio \
             --vnc --noautoconsole \
             -c /path/to/iso/ubuntu-16.04.2-server-amd64.iso
   ```

3. Install MAAS using one of the options:

   - Select the Install MAAS Region Controller option in the boot menu that will install an entire MaaS environment.
   - Select Install Ubuntu Server to install the Ubuntu 16.04 server first, then install MAAS from packages:

     ```
     sudo apt install maas
     ```

4. Configure MAAS as required to complete the installation.
5. Verify the installation by opening the MAAS web UI:

   ```
   http://<MAAS-IP-ADDRESS>:5240/MAAS
   ```

6. If you have installed MAAS from the packages, create an initial (administrative) user first to log in to the MAAS web UI:

   ```
   sudo maas createadmin --username=<PROFILE> --email=<EMAIL_ADDRESS>
   ```

Seealso

MAAS installation

Add an SSH key

To simplify access to provisioned nodes, add an SSH key that MAAS will deliver when provisioning these nodes.

To add an SSH key:

1. In the MAAS web UI, open the user page by clicking on your user name at the top-right conner.
2. Find the SSH keys section.
3. From the Source drop-down menu, specify the source of an SSH key using one of the options:
    - Launchpad, specifying your Launchpad ID
    - Github, specifying your Github ID
    - Upload, placing an existing public key to the Public key edit box
4. Click Import.

Seealso

MAAS Configuration

Add a boot image

You can select images with appropriate CPU architectures that MAAS will import, regularly sync, and deploy to the managed nodes.

To add a new boot image:

1. In the MAAS web UI, open the Images page.
2. Select the releases you want to make available, as well as any architecture.
3. Click Save selection.

Seealso

MAAS Configuration

Add a subnet

You can add new networking elements in MAAS such as fabrics, VLANs, subnets, and spaces. MAAS should detect new network elements automatically. Otherwise, you can add them manually.

To add a new subnet:

1. In the MAAS web UI, open the Subnets page.
2. Select Subnet in the Add drop-down menu at the top-right corner.
3. Specify Name, CIDR, Gateway IP, DNS servers, Fabric & VLAN, and Space.
4. Click Add subnet.

Seealso

MAAS Networking

Enable DHCP on a VLAN

Before enabling DHCP, ensure that you have the MAAS node network interface properly configured to listen to VLAN.

You can use external DHCP or enable MAAS-managed DHCP. Using an external DHCP server for enlistment and commissioning may work but is not supported. High availability also depends upon MAAS-managed DHCP.

To enable MAAS-managed DHCP on a VLAN:

1. In the MAAS web UI, open the Subnets page.
2. Click on the VLAN you want to enable DHCP on.
3. In the VLAN configuration panel, you find DHCP Disabled.
4. From the Take action drop-down menu at the top-right corner, select the Provide dhcp item.
5. In the Provide DHCP panel, verify or change the settings for Rack controller, Subnet, Dynamic range start IP, Dynamic range end IP.
6. Click Provide dhcp.
7. In the VLAN configuration panel, verify that DHCP is enabled.

Seealso

- Network interface configuration
- DHCP in MAAS

Enable device discovery

MAAS provides passive and active methods of device discovery.

Passive methods include:

- Listening to ARP requests
- DNS advertisements

To enable passive device discovery:

1. In the MAAS web UI, open the MAAS dashboard.
2. On the MAAS dashboard, turn on the Discovery enabled switch.
3. Verify if you can see the discovered devices on the MAAS dashboard.

Active methods include active subnet mapping that forces MAAS to discover nodes on all the subnets with enabled active mapping using an active subnet mapping interval value.

To enable active subnet mapping:

1. In the MAAS web UI, open the Settings page.
2. Go to the Device Discovery section.
3. From the drop-down menu, select the value for Active subnet mapping interval.
4. Open the Subnets page.
5. Click the subnet you want to enable active mapping on.
6. In the Subnet summary section, turn on the Active mapping switch.

Seealso

Device discovery

Add a new node

Using MAAS, you can add new nodes in an unattended way called enlistment or manually, when enlistment does not work.

MAAS enlistment uses a combination of DHCP with TFTP and PXE technologies.

Note

To boot a node over PXE, enable netboot or network boot in BIOS.

Note

For KVM virtual machines, specify the boot device as network in the VM configuration file and add the node manually. You need to configure the Virsh power type and provide access to the KVM host as described in BMC Power Types.

To add a new node manually:

1. In the MAAS web UI, open the Nodes page.
2. Click the Add hardware drop-down menu at the top-right corner and select Add machine.
3. Set Machine name, Domain, Architecture, Minimum Kernel, Zone, MAC Address, Power type.

   Note

   See Configure power management for more details on power types.

4. Click Save machine.

MAAS will add the new machine to the list of nodes with the status Comissioning.

Seealso

- Add nodes
- BMC Power Types

Configure power management

MAAS supports many types of power control, from standard IPMI to non-standard types such as virsh, VMWare, Nova, or even completely manual ones that require operator intervention. While most servers may use their own custom vendor management, for example, iLO or DRAC, standard IPMI controls are also supported, and you can use IPMI as shown in the following example.

To configure IPMI node power management type:

1. In the MAAS web UI, open the Nodes page.
2. In the list of nodes, select the one you want to configure.
3. In the machine configuration window, go to the Power section and click the Edit button at the top-right conner.
4. Select IPMI for Power type from the drop-down menu.

5. Specify parameters for the IPMI power type: Power driver, IP address, Power user, Power password, and Power MAC.
6. Click Save changes.

After saving the changes, MAAS will verify that it can manage the node through IPMI.

Seealso

BMC Power Types

Commission a new node

When you add a new node, MAAS automatically starts commissioning the node once configuration is done. Also, you can commission a node manually.

To commision a new node manually:

1. In the MAAS web UI, open the Nodes page.
2. In the list of nodes, click the node you want to configure.
3. In the node configuration window, click Take action at the top-right conner and select Commission.
4. Select additional options by setting the appropriate check boxes to allow SSH access and prevent machine from powering off, retain network and storage configuration if you want to preserve data on the node. For example, if a node comes from an existing cloud with an instance on it.
5. Click Go to start commissioning.
6. Verify that the node status has changed to Ready and hardware summary in the node configuration window has been filled with values other than zeros, which means commisionining was successful.

Note

Use MAAS CLI to commission a group of machines.

Seealso

- MAAS CLI
- Commission nodes
- Add a new node

Deploy a node

Once a node has been commissioned, you can deploy it.

The deployment operation includes installing an operating system and copying the SSH keys imported to MAAS. As a result, you can access the deployed node through SSH using the default user account ubuntu.

To deploy a node:

1. In the MAAS web UI, open the Nodes page.
2. In the list of nodes, verify that the commisioned node is in the Ready state.
3. Click on the node to open the configuration window.
4. In the node configuration window, click the Take action button at the top-right conner and select the Deploy item.
5. Specify the OS, release, and kernel options.
6. Click Go to start the deployment.

Once the deployment is finished, MAAS will change the node status to Deployed.

Seealso

- Deploy nodes
- Add an SSH key

Redeploy a node

To redeploy a node:

1. In the MAAS web UI, open the Nodes page.
2. In the list of nodes, select the node you want to redeploy.
3. In the node configuration window, click Take action at the top-right coner and select Release.
4. Verify that the node status has changed to Ready.
5. Redeploy the node as described in Deploy a node.

Delete a node

Warning

Deleting a node in MAAS is a permanent operation. The node will be powered down, and removed from the MAAS database. All existing configuration done on the node such as name, hardware specs, and power control type will be permanently lost. The node can be readded again, however, it will be unrecognized by MAAS. In such case, you will need to add the node as a new one and reconfigure from scratch.

To delete a node:

1. In the MAAS web UI, open the Nodes page.
2. In the list of nodes, select the node you want to delete.
3. In the node configuration window, click the Take action button at the top-right coner and select Delete.
4. Click Go to confirm the deletion.

SaltStack operations

SaltStack is an orchestration and configuration platform that implements the model-driven architecture (MDA) that you can use to turn services configuration described using Reclass models and Salt Formulas into actual services running on nodes.

Salt Minion operations

Run a command on a node

The Salt integrated cmd.run function is highly flexible and enables the operator to pass nearly any bash command to a node or group of nodes and functions as a simple batch processing tool.

To run a command on a node, execute:

```
salt '[node]' cmd.run '[cmd]'
```

List services on a node

The Salt integrated service.get_all function shows available services on the node.

To list services on a node, run:

```
salt '<NODE_NAME>' service.get_all
```

Restart a service on a node

You can use the Salt integrated service.restart function to restart services.

To restart a service on a node, run:

```
salt '<NODE_NAME>' service.restart <SERVICE_NAME>
```

Note

If you do not know the name of the service or unsure which services are available, see the List services on a node section.

Verify Minions have joined the Master

If you are not sure whether or not a Minion has been joined the Master, verify the output of salt-key. The output of the command lists all known keys in the following states:

- **Accepted**

  Nodes in this state have successfully joined the Salt Master

- **Denied**

  Nodes in this state have not successfully joined because of a bad, duplicate, or rejected key. Nodes in this state require additional user action to join the Master.
- **Rejected**

  Nodes in this state have been explicitly rejected by an administrator.

To verify Minions have joined the Master, run:

```
salt-key
```

Example of a system response:

```
Accepted Keys:
<NODE_NAME>.domain.local
... [snip] ...
Execute salt-key:
Denied Keys:
Unaccepted Keys:
Rejected Keys:
```

Ping a Minion from the Master

You can ping all properly running Salt Minion nodes from the Salt Master. To verify that you have network availability between Salt Minion nodes and the Salt Master node, use the test.ping command.

To ping a Minion from the Master:

```
salt '<NODE_NAME>.domain.local' test.ping
```

Example of a system response:

```
<NODE_NAME>
 True
```

Salt States operations

Salt State is a declarative or imperative representation of a system state.

List available States of a Minion

A Salt Minion node can have different States.

To list available States of a Minion, execute on a node:

```
salt-call state.show_top
```

Example of a system response:

```
local:
 ----------
 base:
     - linux
     - ntp
     - salt
     - collectd
     - sensu
     - heka
     - openssh
     - nova
     - opencontrail
     - ceilometer
```

Apply a State to a Minion

You can apply changes to a Minion's State from the Salt Master.

To apply a State to a Minion, run:

```
salt '<NODE_NAME>' state.sls <STATE_NAME>
```

Salt Formula operations

Salt Formula is a declarative or imperative representation of a system configuration.

Verify and validate a Salt Formula

You can verify and validate a new Salt Formula before applying it by running a quick test for invalid Jinja, YAML, and a Salt state.

To verify a SLS file in a Salt Formula, run:

```
salt '*' state.show_sls <SLS_FILE_NAME>
```

To validate the Salt Formula, run in the test-only (dry-run) mode using the test option:

```
salt '*' state.apply test
```

Seealso

Salt Command Line Reference

Apply a Salt Formula

This section covers how you can test and apply a Salt Formula.

To apply all configured states (highstate) from a Salt Formula to all Minions, run on the Salt Master:

```
salt '*' state.apply
```

Note

This command is equal to:

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

To apply individual SLS files in a Salt Formula, run:

```
salt '*' state.apply <SLS_FILE_1>,<SLS_FILE_2>
```

Warning

Applying Salt Formulas on more than 100 nodes may result in numerous failures.

Note

SaltStack runs new states in parallel leading to temporary out of service that may affect end users. To avoid taking down services on all the nodes at the same time, you can stagger highstates in a batch mode.

To apply a Salt Formula on a big number of nodes, for example, more than 100 nodes, follow one of the approaches below.

- Use the --batch-size or -b flags to specify the number of nodes to have Salt apply a state in parallel:

  ```
  salt --batch-size <NUMBER_OF_NODES> '*' state.apply
  ```

- Specify a percentage of nodes to apply a highstate on:

  ```
  salt -b <PERCENTAGE> '*' state.apply
  ```

- Use node name conventions in the form of <GROUP>.<NODE_TYPE_NAME><NUM> to run a highstate by a pattern. For example: group1.cmp001:

  ```
  salt 'group1.cmp*' state.highstate
  ```

- Use Node Groups that you can define in the Salt Master configuration file /etc/salt/master. To run a highstate on nodes within a Node Group, run:

  ```
  salt -N <GROUP_NODE> state.apply
  ```

- Use Grains for grouping nodes specifying a grain variable in the /etc/salt/grains configuration file and then specify the grain value in the Salt command to apply a highstate for the nodes that have this grain value assigned:

  ```
  salt -G <GRAIN_NAME>:<GRAIN_VALUE> state.apply
  ```

Note

You can use --batch-size flag together with Node Groups and Grains. For example:

```
salt --batch-size 10% -N computes1 state.apply
salt -b 5 -N compute:compute1 state.apply
```

Seealso

- Salt Node Groups
- Salt Grains
- Salt Command Line Reference

Seealso

SaltStack components

OpenStack operations

This section describes reprovisioning of OpenStack Controllers and Compute nodes.

Reprovision a controller node

If you have lost a controller node and recovery is not possible, you can recreate a controller node from scratch.

To reprovision a controller node:

1. Verify MAAS works properly and provides the DHCP service to assign IP addresses and bootstrap instances.
2. Run the salt.control state on a KVM node:

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

   Salt virt takes the name of a virtual machine and registers the virtual machine into the Salt Master node.

   Once created, the instance picks up an IP address from the MAAS DHCP service and the key will be seen as accepted in the Salt Master node.
3. Run the Salt highstate:

   ```
   salt '<CONTROLLER_NAME>' state.highstate -l info
   ```

Reprovision a compute node

Provisioning compute nodes is relatively straightforward in that you can run all states at once. Though, you need to run and reboot it multiple times for network configuration changes to take effect.

Note

Multiple reboots are needed because the ordering of dependencies is not yet orchestrated.

To reprovision a compute node:

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

> Note
>
> Create as many hosts as you have compute nodes in your environment within this file.

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

   ```
   salt ‘*cfg*’ state.sls reclass.storage
   ```

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

   ```
   salt ‘*cmp[<NUM>]*’ test.ping
   ```

4. Run the Salt highstate for the compute node(s):

   ```
   salt ‘*cmp[<NUM>]*’ state.highstate
   ```

   > Note
   >
   > Failures may occur during the first run of highstate.

5. Reboot to apply network configuration changes.
6. Rerun the Salt highstate for the node(s):

   ```
   salt ‘*cmp[<NUM>]*’ state.highstate
   ```

7. Provision the vRouter on the compute node using CLI or the Contrail web UI. Example of the CLI command:

   ```
   salt ‘*cmp[<NUM>]*’ cmd.run '/usr/share/contrail-utils/provision_vrouter.py \
       --host_name <CMP_HOSTNAME> --host_ip <CMP_IP_ADDRESS> --api_server_ip <CONTRAIL_VIP> \
       --oper add --admin_user admin --admin_password <PASSWORD> \
       --admin_tenant_name admin --openstack_ip <OPENSTACK_VIP>'
   ```

MCP Control Plane

The MCP Control Plane consists of:

- The continuous integration and continuous delivery (CI/CD) pipeline
- The DevOps Portal

The CI/CD pipeline enables you to run an up-to-date cloud platform by delivering software updates and configuration changes in a smoothly and reliable manner.

Using CI/CD pipeline results in stable software that is consistently tested, shared, and promoted across all your MCP clusters.

CI/CD pipeline has the following components in the basement of the MCP CI/CD process:

- Jenkins - a job server
- Gerrit - a gate for reviewing changes coming to a Git repository
- Aptly - Debian repository management
- Artifactory - a registry for artifacts such as Docker images and deb packages

The CI/CD process for MCP cloud cluster can be divided into three main steps:

- Development and configuration
- Staging (verification and validation)
- Deployment to production

To manage CI/CD pipeline, visit the DevOps Portal.

Logging, metering, alerting

Using StackLight for Mirantis Cloud Platform, cloud operators can monitor an OpenStack environment and be quickly notified of critical conditions that may occur in the system so that they can prevent service downtimes.

This section describes how StackLight works and how to use the components integrated with the StackLight operational insights pipeline.

Restart Log Collector

The Log Collector starts automatically when the heka Salt state is applied. To manually start and stop it, use the Salt commands.

The following example shows how to restart the Log Collector from the Salt Master node on all Salt Minion nodes with names that start with ctl:

```
# salt 'ctl*' service.restart log_collector
```

Alternatively, you can SSH to the node and use the service manager (systemd or upstart) to restart the service. For example:

```
# ssh ctl01.mcp-lab-advanced.local
# service log_collector restart
```

See the salt.modules.service documentation for more information on how to use the Salt service execution module.

Restart Metric Collector

The Local Metric Collector runs the metric_collector and collectd services on all monitoring nodes.

The following example shows how to restart the Metric Collector from the Salt Master node on all Salt minion nodes with the names that start with ctl:

```
# salt 'ctl*' service.restart metric_collector
# salt 'ctl*' service.restart collectd
```

The Remote Metric Collector is active on the node with the monitoring VIP. It runs the remote_collector and the remote_collectd services.

Note

The IP address of the monitoring VIP is stored in the Salt Pillar.

To find the IP address value, use the following command (where cfg01.mcp-lab-advanced.local is the Salt Master node):

```
# reclass --nodeinfo cfg01.mcp-lab-advanced | grep stacklight_monitor_address
```

To find the node with the VIP, use the following command (assuming that the monitoring VIP is 172.16.10.253):

```
# salt -G 'ipv4:172.16.10.253' test.ping
```

The following example shows how to restart the Remote Metric Collector (assuming that the mon01 node has the monitoring VIP):

```
# salt 'mon01.mcp-lab-advanced' service.restart remote_collector
# salt 'mon01.mcp-lab-advanced' service.restart remote_collectd
```

Warning

Do not attempt to start remote_collector or remote_collectd on a monitoring node that does not have the monitoring VIP. It may lead to multiple Remote Metric Collectors running at the same time, resulting in unexpected behavior.

Alternatively, use the following Salt command to restart the remote_collector service on the node with the monitoring VIP (assuming that the monitoring VIP is 172.16.10.253):

```
# salt -G 'ipv4:172.16.10.253' service.restart remote_collector
```

Restart Aggregator

The Aggregator runs on one of the monitoring nodes. For example, the Aggregator runs on the monitoring node named mon01.mcp-lab-advanced. Use the following command to restart the Aggregator on that node:

```
# salt 'mon01.mcp-lab-advanced.local' service.restart aggregator
```

Warning

Do not attempt to start the Aggregator on a monitoring node that does not have the monitoring VIP. It may lead to multiple Aggregators running at the same time, resulting in unexpected consequences.

For example, the monitoring VIP is 172.16.10.253. Use the following command to restart the Aggregator on the node with the monitoring VIP:

```
# salt -G 'ipv4:172.16.10.253' service.restart aggregator
```

InfluxDB and Grafana operations

After you install and configure InfluxDB and Grafana, you can start using a collection of predefined dashboards to visualize the time-series of your OpenStack environment.

Connect to Grafana

You can access Grafana through the monitoring VIP on port 3000 by default. To access the Grafana dashboard, enter the credentials that can be retrieved from the Salt Pillar as follows (where cfg01.mcp-lab-advanced.local is the Salt Master node):

```
# reclass --nodeinfo cfg01.mcp-lab-advanced.local | grep grafana_user
   grafana_user: admin
# reclass --nodeinfo cfg01.mcp-lab-advanced.local | grep grafana_password
   grafana_password: password
```

Restart Grafana

The Grafana service is called grafana-server. Use the following steps to restart the Grafana service on all nodes of the monitoring cluster:

1. Log in to the Salt Master node.
2. Restart Grafana:

   ```
   # salt -C 'I@grafana:server' service.restart grafana-server
   ```

   You can use the test.ping command to test the node selection. For example:

   ```
   # salt -C 'I@grafana:server' test.ping
   ```

3. Verify the status of the Grafana server on all nodes of the monitoring cluster:

   ```
   # salt -C 'I@influxdb:server' service.status grafana-server
   ```

   Example of system response:

   ```
   mon01.mcp-lab-advanced.local:
       True
   ```

   In this example, the Grafana server runs on one node called mon01.mcp-lab-advanced.local on the monitoring cluster.

Restart InfluxDB

The InfluxDB service is called influxdb. Use the following steps to restart the InfluxDB server on the Salt minion nodes with the influxdb:server role in the Salt Pillar:

1. Log in to the Salt Master node.
2. Restart InfluxDB:

   ```
   # salt -C 'I@grafana:server' service.restart influxdb
   ```

3. Verify that the InfluxDB server successfully runs on the monitoring cluster node with the monitoring VIP:

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

   Example of system response:

   ```
   mon01.mcp-lab-advanced.local:
       True
   ```

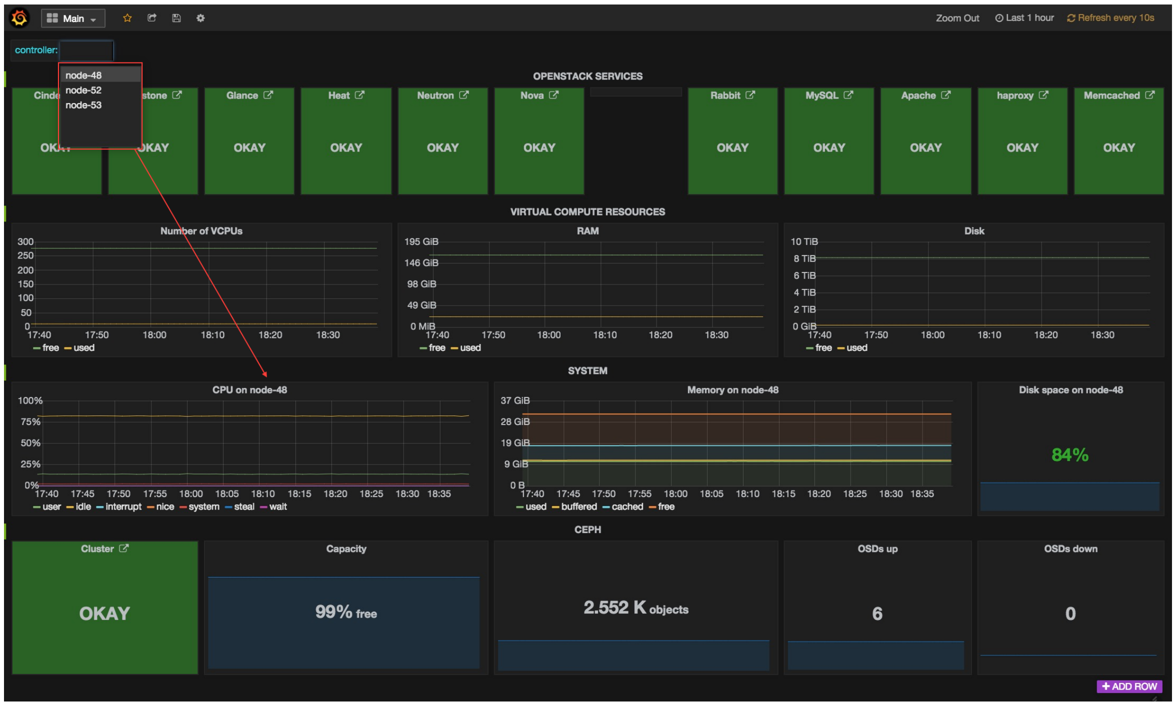
   In this example, the InfluxDB server runs on one node named mon01.mcp-lab-advanced.local on the monitoring cluster.

Manage Grafana dashboards

Grafana provides a collection of predefined dashboards to visualize the time-series of your MCP cluster.

Main dashboard

Mirantis recommends that you start with the Main dashboard as an entry to other dashboards. It provides a single pane of glass from where you can visualize the overall health status of your OpenStack services such as Nova, Cinder, HAProxy, MySQL, RabbitMQ, and others.



The Main dashboard, like most dashboards, provides a drop-down menu in the upper left corner from where you can pick a particular metric dimension such as the controller name or the device name you want to select.

In the example above, the dashboard displays the system metrics of node-48. The following sections are displayed in the dashboard:

- The OPENSTACK SERVICES section where each of the represented services can be assigned five different statuses.

  Note

  The precise determination of a service health status depends on the correlation policies implemented for that service by a Global Status Evaluation (GSE) plugin. See Aggregation and correlation.

  The service health statuses can be as follows:

Status	Description
DOWN	When one or several primary functions of a service cluster has failed. For example, all API endpoints of a service cluster like Nova or Cinder failed.
CRITICAL	When one or several primary functions of a service cluster are severely degraded. The quality of service delivered to the end user is severely impacted.

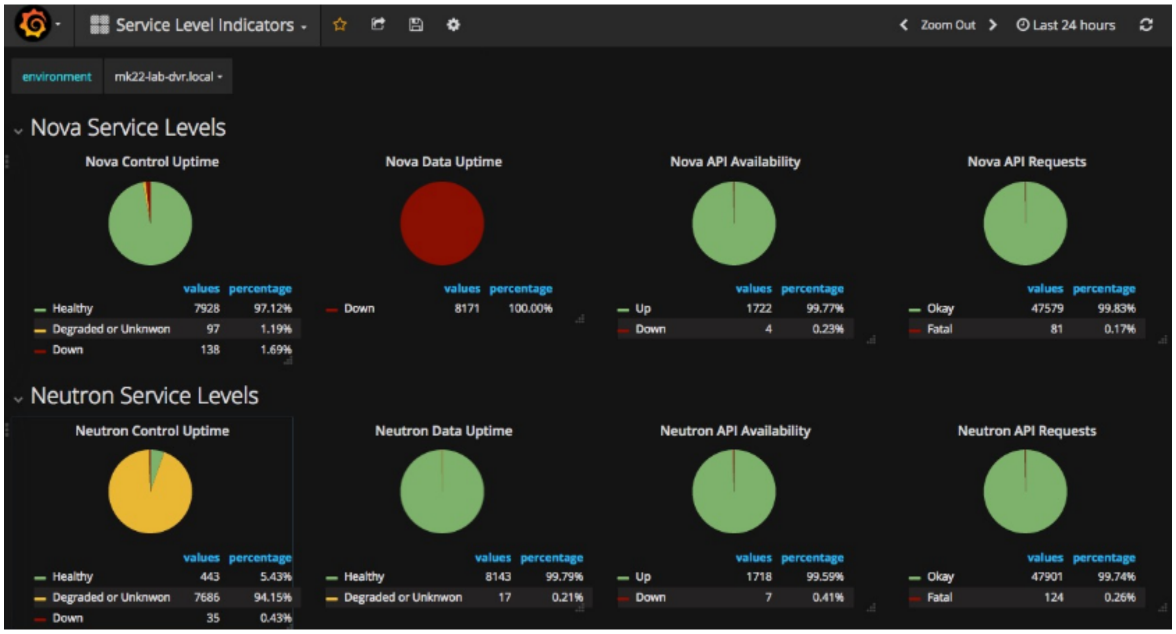
WARNING	When one or several primary functions of a service cluster are slightly degraded. The quality of service delivered to the end user is slightly impacted.
UNKNOWN	When there is not enough data to infer the actual health status of a service cluster.
OKAY	When none of the above was found to be true.

- The VIRTUAL COMPUTE RESOURCES section provides an overview of the number of virtual resources being used by the compute nodes including the following information:
    - Number of virtual CPUs
    - Memory and disk space being used
    - Virtual resources that remain available to create new instances
- The SYSTEM section provides an overview of the number of physical resources being used on the control plane (the controller cluster). You can select a specific controller using the controller drop-down list in the left corner of the toolbar.
- The Ceph section provides an overview of the resources usage and current health status of the Ceph cluster when it is deployed in the OpenStack environment.
- The Main section is also an entry point to access more detailed dashboards for each of the OpenStack services that are monitored. For example, if you click the Nova box, the Nova dashboard is displayed. For details, see Nova dashboard.

Service Level dashboard

The Service Level dashboard is another single pane of glass entry point in Grafana to visualize service uptime statistics. Service uptime statistics shows in percentage the health status of each OpenStack top-level cluster as reported by StackLight for to the chosen time-picker interval. Using that dashboard, you can easily verify whether your service-level objectives were met during the past period.

The screen capture below shows an example of indicators displayed on the Service Level dashboard:



Nova Service Levels

Nova Control Uptime

	values	percentage
🟩 Healthy	7928	97.12%
🟨 Degraded or Unknwon	97	1.19%
🟥 Down	138	1.69%

Nova Data Uptime

	values	percentage
🟥 Down	8171	100.00%

Nova API Availability

	values	percentage
🟩 Up	1722	99.77%
🟥 Down	4	0.23%

Nova API Requests

	values	percentage
🟩 Okay	47579	99.83%
🟥 Fatal	81	0.17%

Neutron Service Levels

Neutron Control Uptime

	values	percentage
🟩 Healthy	443	5.43%
🟨 Degraded or Unknwon	7686	94.15%
🟥 Down	35	0.43%

Neutron Data Uptime

	values	percentage
🟩 Healthy	8143	99.79%
🟨 Degraded or Unknwon	17	0.21%

Neutron API Availability

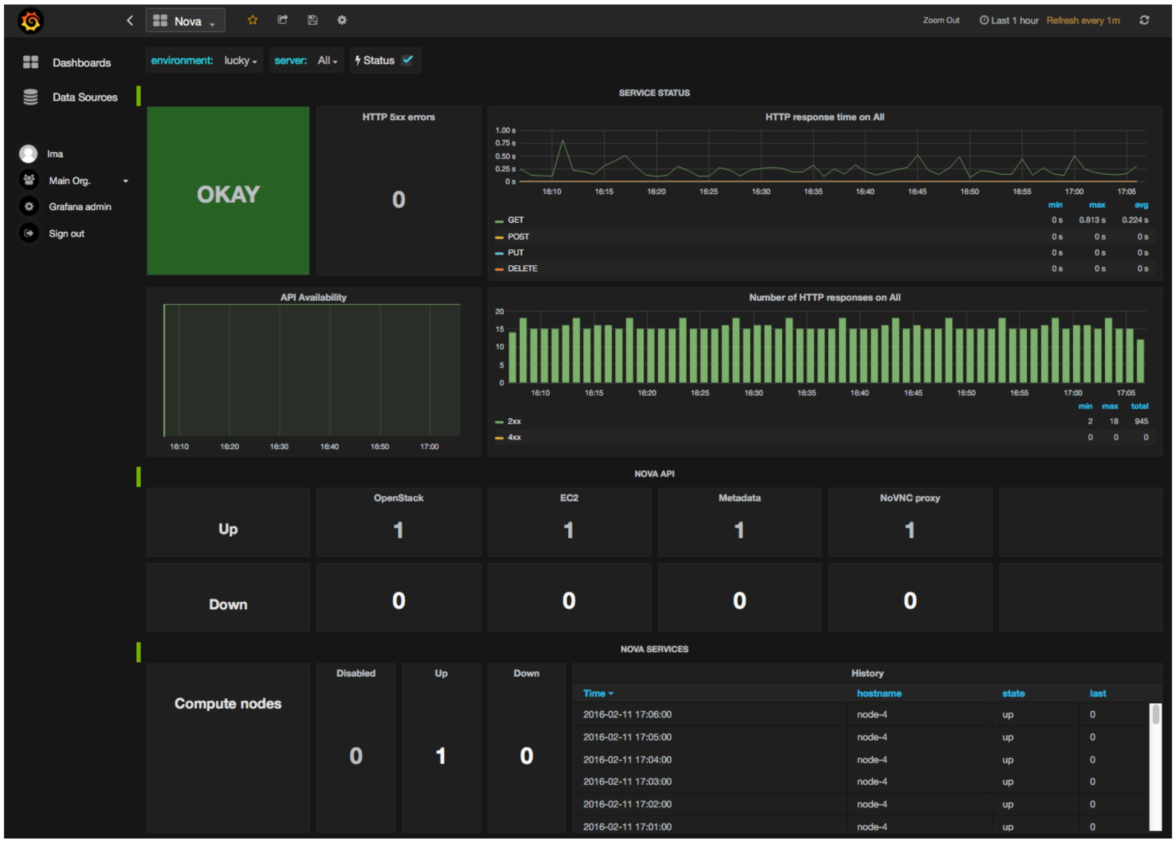
	values	percentage
🟩 Up	1718	99.59%
🟥 Down	7	0.41%

Neutron API Requests

	values	percentage
🟩 Okay	47901	99.74%
🟥 Fatal	124	0.26%

Nova dashboard

The Nova dashboard provides a detailed view of the Nova-related metrics.



	min	max	avg
GET	0 s	0.813 s	0.224 s
POST	0 s	0 s	0 s
PUT	0 s	0 s	0 s
DELETE	0 s	0 s	0 s

	min	max	total
2xx	2	18	945
4xx	0	0	0

	OpenStack	EC2	Metadata	NoVNC proxy	
Up	1	1	1	1	
Down	0	0	0	0	

	Disabled	Up	Down
Compute nodes	0	1	0

History

Time	hostname	state	last
2016-02-11 17:06:00	node-4	up	0
2016-02-11 17:05:00	node-4	up	0
2016-02-11 17:04:00	node-4	up	0
2016-02-11 17:03:00	node-4	up	0
2016-02-11 17:02:00	node-4	up	0
2016-02-11 17:01:00	node-4	up	0

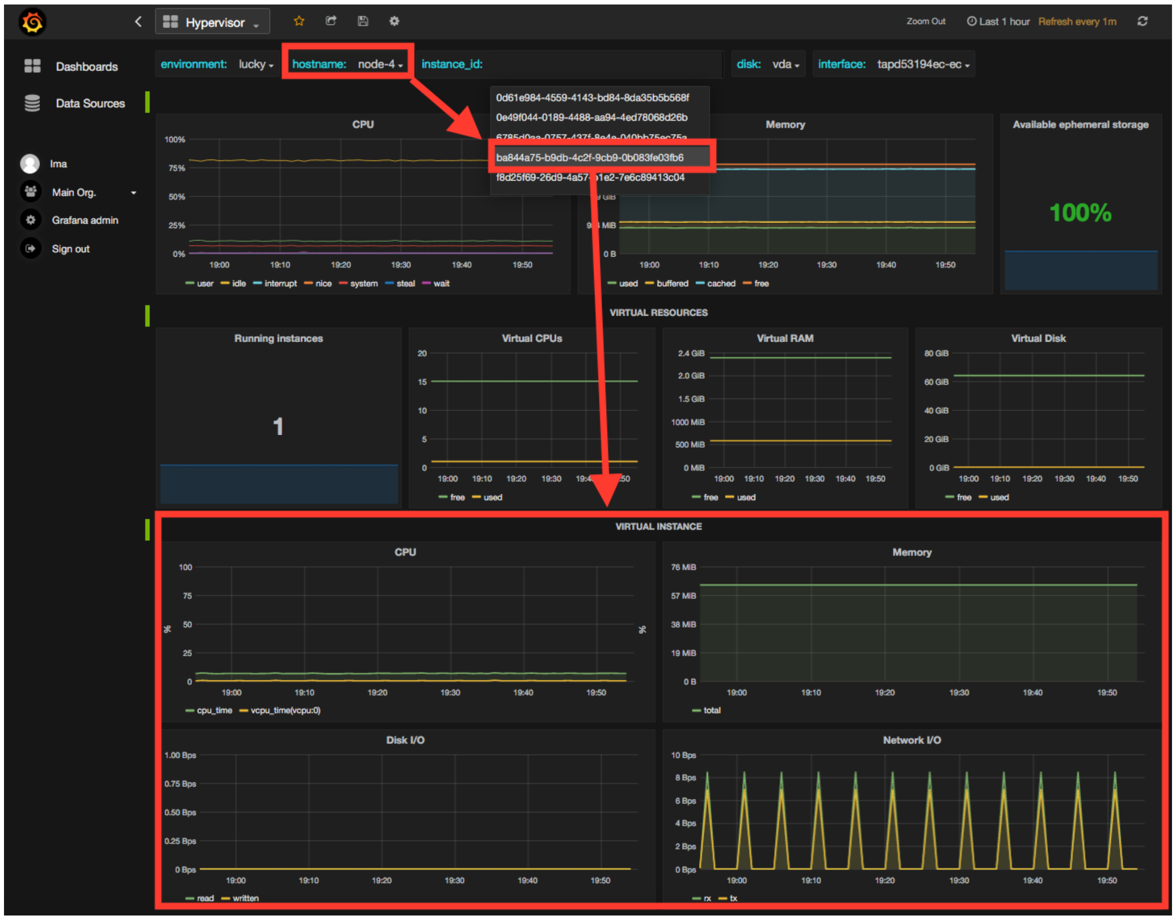
The Nova dashboard consists of the following sections:

- The SERVICE STATUS section displays the overall health status of the Nova service cluster, including the following information:
  - The status of the API front end (the HAProxy public VIP)
  - A counter of HTTP 5xx errors
  - The HTTP requests response time
  - The status code
- The NOVA API section displays the current health status of the API back ends, for example, nova-api, ec2-api, and others.
- The NOVA SERVICES section displays the current and historical status of the Nova workers.
- The INSTANCES section displays information about the number of active instances in error state and statistics about the creation time of instances.
- The RESOURCES section displays various indicators of the virtual resources usage.

Hypervisor dashboard

The Hypervisor dashboard brings operational insights about the virtual instances managed through libvirt.

The Hypervisor dashboard assembles various libvirt metrics. Use the drop-down menu to choose a particular instance UUID running on a particular node. The example below shows the metrics for the instance ID ba844a75-b9db-4c2f-9cb9-0b083fe03fb7 running on node-4.



Additional dashboards

Grafana provides four additional dashboards for Kubernetes, etcd, Docker, and Calico.

Grafana also provides 20 dashboards for OpenStack that you can use to explore different time-series facets of your OpenStack environment. You may want to use the following OpenStack additional dashboards:

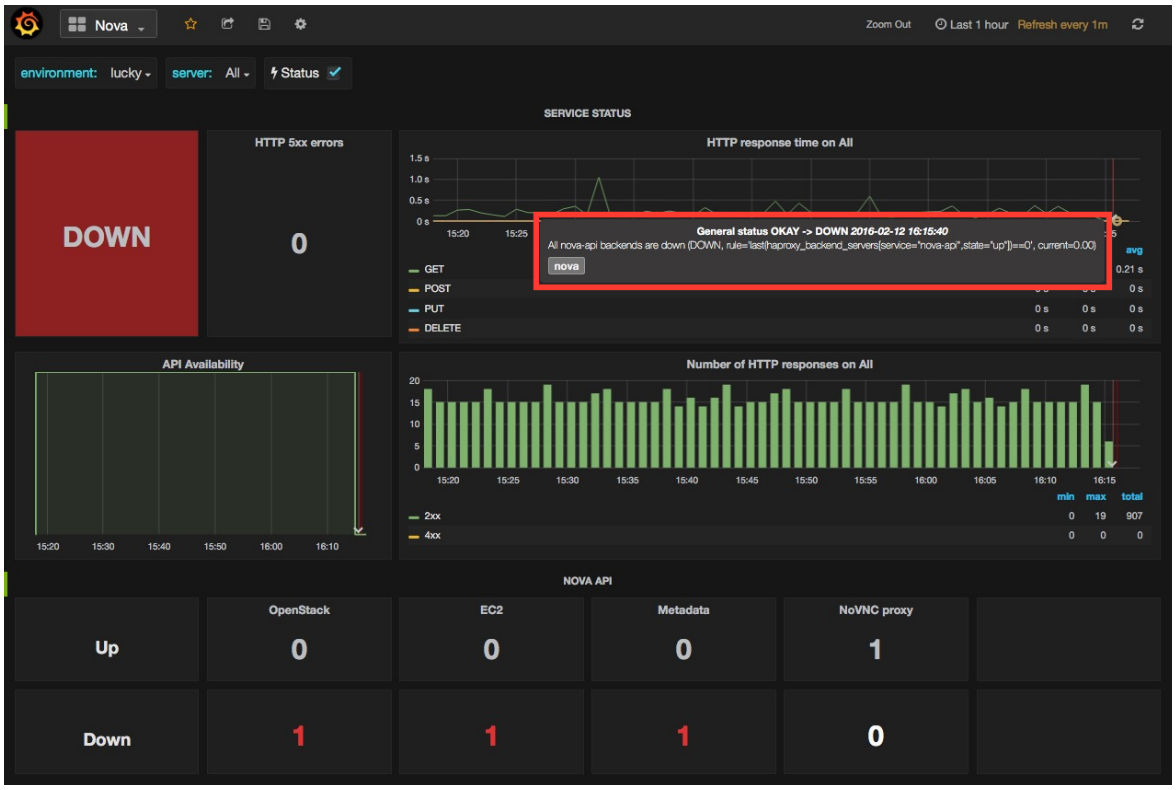
- The Elasticsearch Cluster dashboard monitors the overall health status of the Elasticsearch cluster including the state of the shards, the number of pending tasks, and various resources usage metrics.
- The InfluxDB Cluster dashboard provides statistics about the InfluxDB processes running in the InfluxDB cluster including various metrics of the resources usage.

Viewing faults and anomalies

The LMA Toolchain can detect a number of service-affecting conditions such as the faults and anomalies that occurred in your OpenStack environment. These conditions are reported in annotations that are displayed in Grafana.

The Grafana annotations contain a textual representation of the alarm (or set of alarms) that were triggered by the Collectors for a service. In other words, the annotations contain valuable insights that you can use to diagnose and troubleshoot issues.

On the screen capture below, an annotation shows that the health status of Nova is DOWN because there is no nova-api service back end (viewed from HAProxy) that is UP.



General status OKAY -> DOWN *2016-02-12 16:15:40*

```
All nova-api backends are down (DOWN, rule='last(haproxy_backend_servers[service="nova-api",state="up"])==0', current=0.00)
```

Hiding nodes from dashboards

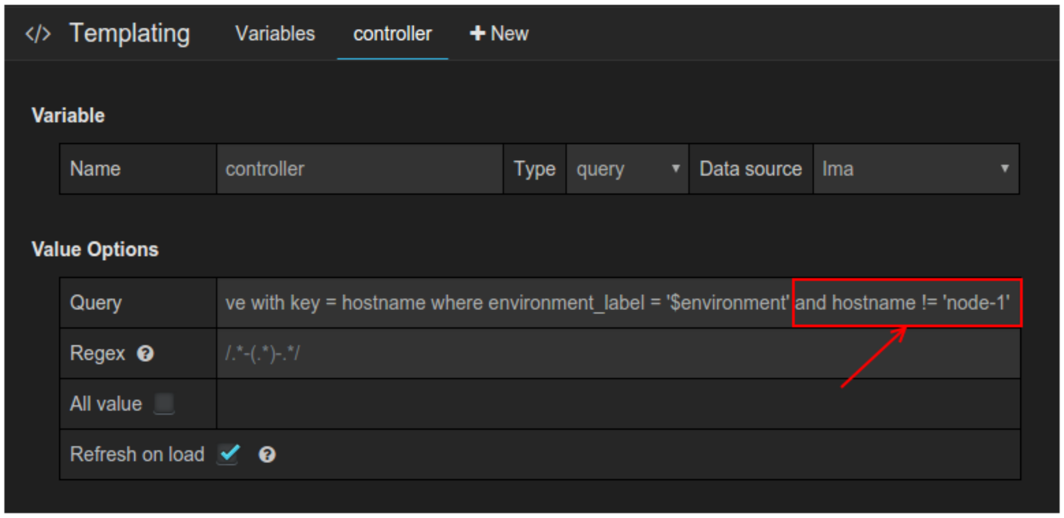
When you remove a node from the environment, it is still displayed in the server and controller drop-down lists of the Grafana dashboards.

To hide a node from the list:

1. Edit the associated InfluxDB query in the Templating section. For example:

   - To remove node-1, add the following condition to the where clause:

     ```
     and hostname != 'node-1'
     ```



Variable

Name	controller	Type	query	Data source	lma

Value Options

Query	ve with key = hostname where environment_label = '$environment' and hostname != 'node-1'
Regex	/.*-(.*)-.*/
All value	
Refresh on load	✓

- To hide more than one node, add more conditions:

```
and hostname != 'node-1' and hostname != 'node-2'
```

2. Repeat step 1 for all dashboards that display the deleted node.
3. Save the changes.

Seealso

The InfluxDB and Grafana official documentation:

- InfluxDB
- Grafana

Elasticsearch and Kibana operations

After you install and configure Elasticsearch and Kibana, you can start using the Kibana dashboards to monitor and diagnose issues in your OpenStack environment.

Connect to Kibana

Kibana can be reached through the monitoring VIP on port 5601 by default. No credentials are required to connect to the Kibana dashboard.

Restart Kibana

The Kibana service is called kibana.

To restart Kibana:

1. Log in to the Salt Master node.

2. Run the following command:

```
# salt -C 'I@kibana:server' service.restart kibana
```

Restart Elasticsearch

The Elasticsearch service is called elasticsearch.

To restart the Elasticsearch service:

1. Log in to the Salt master node.
2. Run the following command to restart the Elasticsearch service on all monitoring nodes one by one:

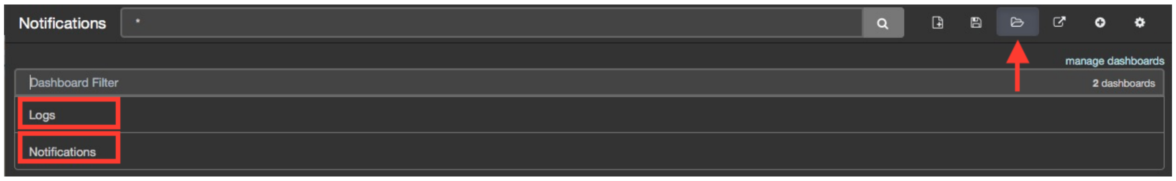
```
# salt -C 'I@elasticsearch:server' service.restart elasticsearch -b
```

Manage Kibana dashboards

The StackLight Kibana contains two built-in dashboards:

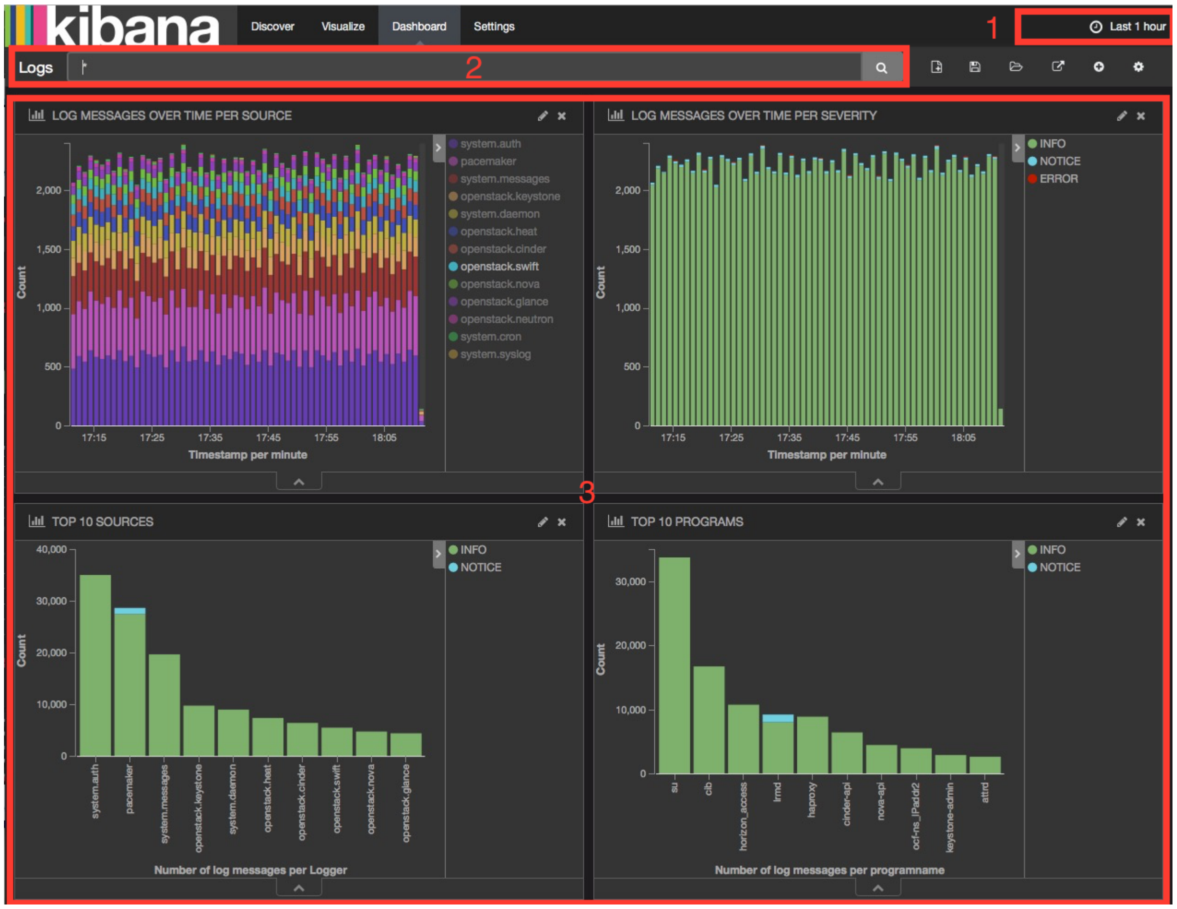
- The Logs analytics dashboard that is used to visualize and search the logs.
- The Notifications analytics dashboard that is used to visualize and search the OpenStack notifications. This dashboard is available if you enable the feature in the Collector settings.
- The Audit analytics dashboard that is used to visualize and search for the OpenStack CADF notifications.

You can switch from one dashboard to another by clicking on the top-right Load icon on the toolbar, as shown on the screen capture below:



Each dashboard provides a single pane of glass for visualizing and searching for the logs and notifications of your OpenStack environment.

The Kibana dashboard for logs is divided into several sections as follows:



On the screen capture above, the following items are highlighted:

1. A time-picker control to choose the required time period and refresh frequency.
2. A text box to enter search queries.
3. The logs analytics with six different panels showing the following stack graphs:
    1. All logs per source
    2. All logs per severity
    3. All logs for top 10 sources
    4. All logs for top 10 programs
    5. All logs for top 10 hosts
    6. The number of logs per severity
    7. The number of logs per role

    The table of log messages is sorted in the reverse chronological order.

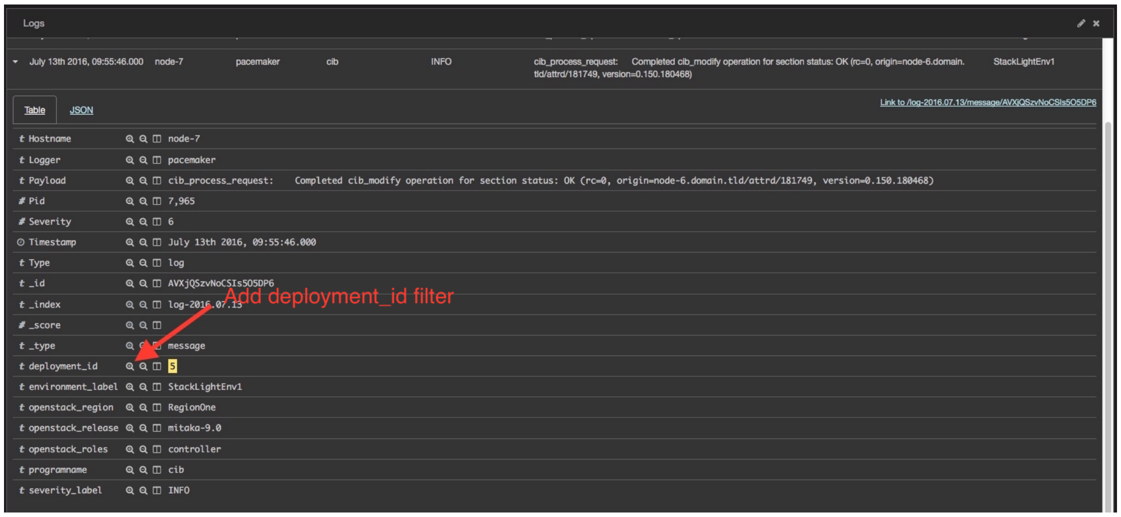
Use Kibana filters and queries

Filters and queries have similar syntax but they are used for different purposes:

- The filters are used to restrict what is displayed in the Kibana dashboard.
- The queries are used for free-text search.

You can combine multiple queries and compare their results. You can also further filter the log messages. For example, to select the deployment_id filter:

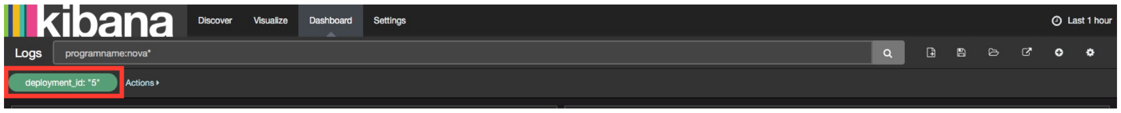
1. Expand a log entry.
2. Select the deployment_id field by clicking on the magnifying glass icon as follows:



```
July 13th 2016, 09:55:46.000   node-7   pacemaker   cib   INFO   cib_process_request:   Completed cib_modify operation for section status: OK (rc=0, origin=node-6.domain.tld/attrd/181749, version=0.150.180468)   StackLightEnv1

Hostname            node-7
Logger              pacemaker
Payload             cib_process_request:    Completed cib_modify operation for section status: OK (rc=0, origin=node-6.domain.tld/attrd/181749, version=0.150.180468)
Pid                 7,965
Severity            6
Timestamp           July 13th 2016, 09:55:46.000
Type                log
_id                 AVXjQSzvNoCSIs5O5DP6
_index              log-2016.07.13
_score
_type               message
deployment_id       5
environment_label   StackLightEnv1
openstack_region    RegionOne
openstack_release   mitaka-9.0
openstack_roles     controller
programname         cib
severity_label      INFO
```

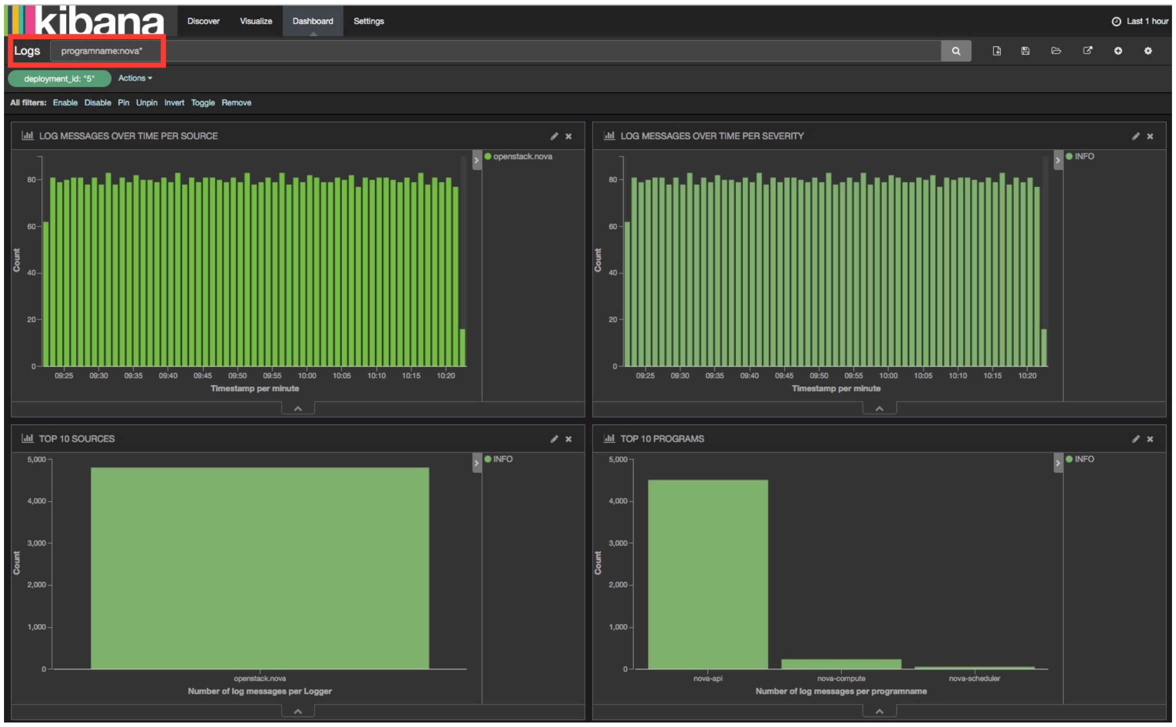
This will apply a new filter in the Kibana dashboard:



Filtering works for any field that has been indexed for the log entries that are present in the Kibana dashboard.

Filters and queries can also use wildcards that can be combined with the field names like in programname: <name>*.

For example, to display only the Nova logs, enter programname:nova* in the query text box as follows:



Seealso

The Elasticsearch and Kibana official documentation:

- Elasticsearch
- Kibana

Sensu operations

After you install and configure Sensu and Uchiwa, you can start using the Uchiwa dashboard to visualize the health status of all nodes and services running on your OpenStack environment.

Connect to Sensu

The Sensu dashboard supported by Uchiwa can be reached through the monitoring VIP on port 3001 by default. The Uchiwa credentials can be retrieved from Pillar using the following command:

```
salt -C 'I@sensu:server' pillar.data sensu:dashboard:admin
```

Example of system response:

```
mon01.mcp-lab-advanced:
    ----------
    sensu:dashboard:admin:
        ----------
        password:
            password
        username:
            admin
```

Restart Sensu

The Sensu services run on the MCP monitoring cluster.

To restart the Sensu services:

1. Log in to the Salt Master node.
2. Restart the Sensu services:

   ```
   salt -C 'I@sensu:server' service.restart sensu-server
   salt -C 'I@sensu:server' service.restart sensu-api
   salt -C 'I@sensu:server' service.restart uchiwa
   ```

3. Restart the Sensu client:

   ```
   salt -C 'I@sensu:client' service.restart sensu-client
   ```

Manage Sensu dashboard

The Sensu dashboard is supported by Uchiwa. The Uchiwa dashboard contains two main panels that are described in this section.

The EVENTS panel

Once you authenticate to Uchiwa as described in the Connect to Sensu section, you are automatically redirected to the EVENTS panel that shows the list of current Sensu events. If the cluster status is up and running, no events are displayed on this panel.

The CLIENTS panel

The CLIENTS panel shows the list of the Sensu clients that are registered with Sensu.

A client is typically a Sensu client, but it can also be a proxy client. In the StackLight integration with Sensu, the Aggregator is a proxy client that registers with Sensu.

Three virtual clients represent the aggregated health status of top-clusters, node-clusters, and service-clusters:

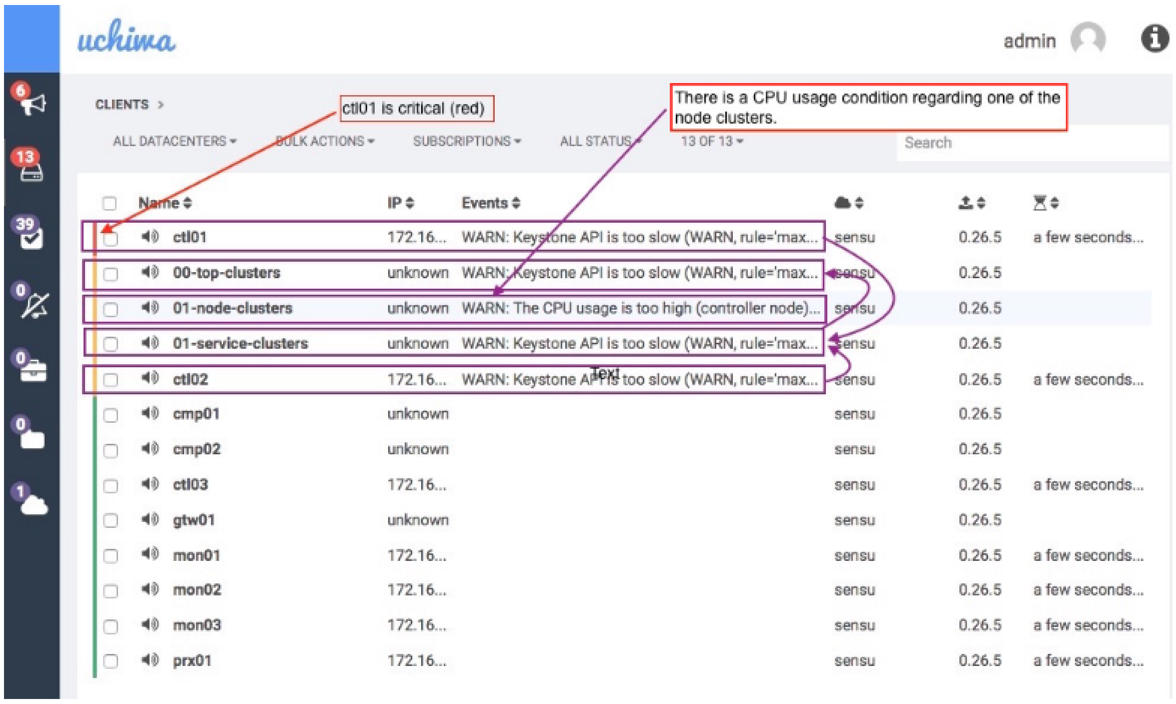
- 00-top-clusters represents the aggregated health status of the top-level clusters such as Nova, Keystone, RabbiMQ, GlusterFS, and so on.
- 01-node-clusters represents the aggregated health status of node clusters such as controller and compute clusters.

- 01-service-clusters represents the aggregated health status of service clusters such as cinder_api, cinder_logs, cinder_scheduler, and so forth.

These virtual clients have no IP address assigned to them.

When an anomaly or fault is detected by the StackLight Operational Insights pipeline, the state of the clients changes and its details are displayed on the panel. The green vertical line also changes its color depending on the state.

Example:



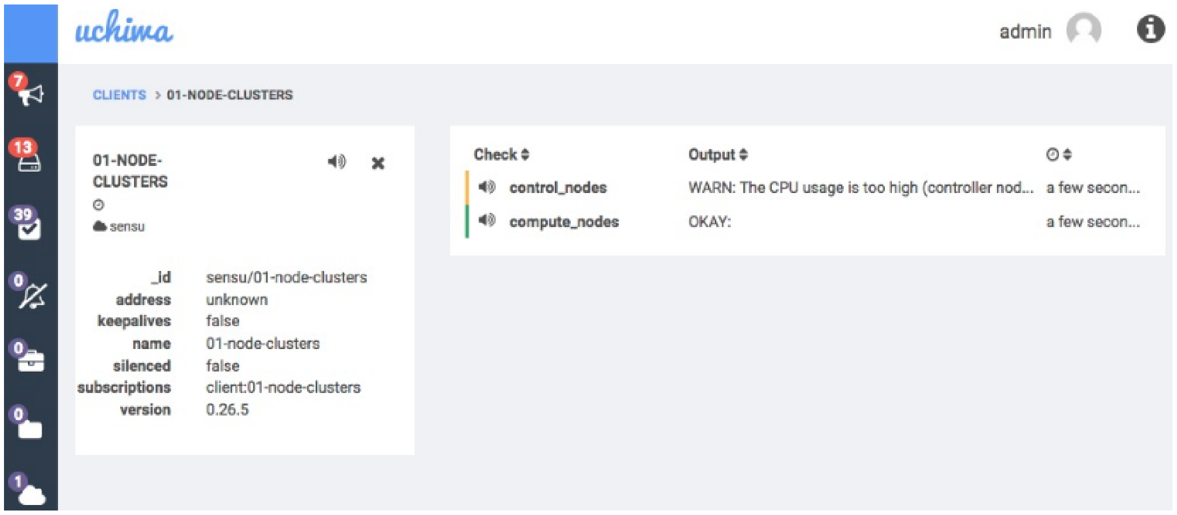
ctl01 is critical (red)

There is a CPU usage condition regarding one of the node clusters.

Name	IP	Events			
ctl01	172.16...	WARN: Keystone API is too slow (WARN, rule='max...	sensu	0.26.5	a few seconds...
00-top-clusters	unknown	WARN: Keystone API is too slow (WARN, rule='max...	sensu	0.26.5	
01-node-clusters	unknown	WARN: The CPU usage is too high (controller node)...	sensu	0.26.5	
01-service-clusters	unknown	WARN: Keystone API is too slow (WARN, rule='max...	sensu	0.26.5	
ctl02	172.16...	WARN: Keystone API is too slow (WARN, rule='max...	sensu	0.26.5	a few seconds...
cmp01	unknown		sensu	0.26.5	
cmp02	unknown		sensu	0.26.5	
ctl03	172.16...		sensu	0.26.5	a few seconds...
gtw01	unknown		sensu	0.26.5	
mon01	172.16...		sensu	0.26.5	a few seconds...
mon02	172.16...		sensu	0.26.5	a few seconds...
mon03	172.16...		sensu	0.26.5	a few seconds...
prx01	172.16...		sensu	0.26.5	a few seconds...

In the example above, the status of ctl01 is CRITICAL (red) and the status of ctl02 is WARNING (yellow) displaying the last received event stating that the Keystone API is too slow.

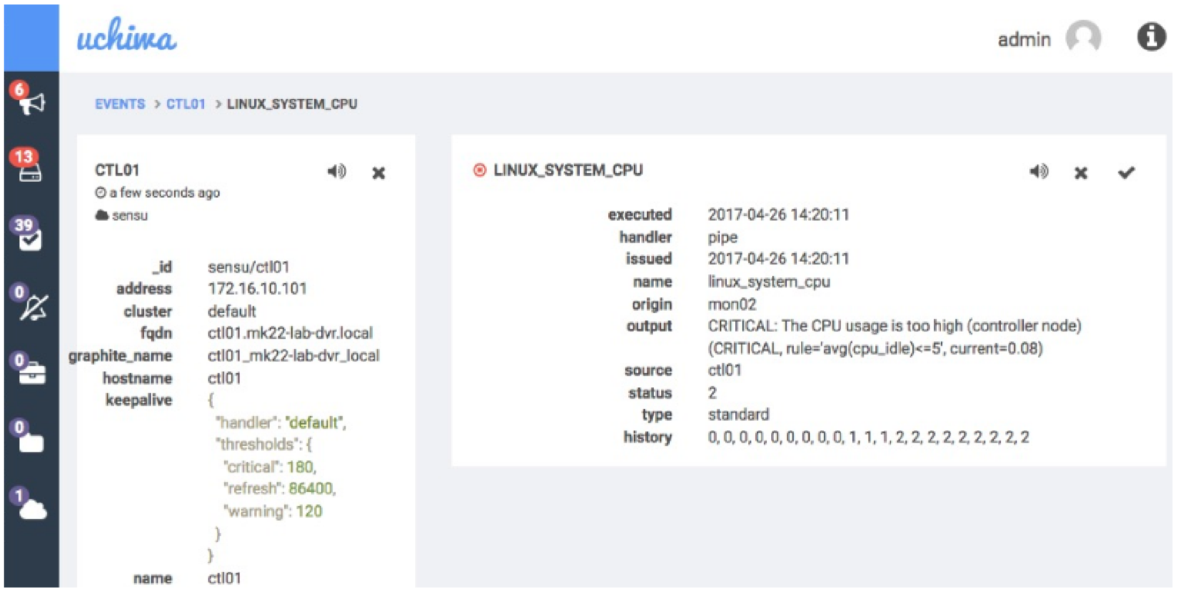
As a result, the status of the 01-service-clusters and 00-top-clusters virtual clients have also changed to WARNING. The status of 01-node-clusters is WARNING that is related to the CRITICAL status of ctl01 as follows:



Check	Output	
control_nodes	WARN: The CPU usage is too high (controller nod...	a few secon...
compute_nodes	OKAY:	a few secon...

_id	sensu/01-node-clusters
address	unknown
keepalives	false
name	01-node-clusters
silenced	false
subscriptions	client:01-node-clusters
version	0.26.5

The status of control_nodes is WARNING because the CPU usage on ctl01 is too high, and hence, ctl01 has the CRITICAL status.



```
_id            sensu/ctl01
address        172.16.10.101
cluster        default
fqdn           ctl01.mk22-lab-dvr.local
graphite_name  ctl01_mk22-lab-dvr_local
hostname       ctl01
keepalive      {
                 "handler": "default",
                 "thresholds": {
                   "critical": 180,
                   "refresh": 86400,
                   "warning": 120
                 }
               }
name           ctl01
```

```
executed  2017-04-26 14:20:11
handler   pipe
issued    2017-04-26 14:20:11
name      linux_system_cpu
origin    mon02
output    CRITICAL: The CPU usage is too high (controller node)
          (CRITICAL, rule='avg(cpu_idle)<=5', current=0.08)
source    ctl01
status    2
type      standard
history   0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2
```

However, the overall status of the control_nodes_clusters client is WARNING because only one of the cluster controller nodes is affected. This is the result of the aggregation and correlation policies in action. The overall status of the cluster of nodes is WARNING, despite that one of its members has the CRITICAL status when the majority of its members is healthy.

Seealso

Sensu official documentation

Use Horizon monitoring dashboards

After you deploy Horizon along with the Telemetry and Monitoring plugins, you can start auditing your MCP cluster using the Horizon monitoring dashboards.

The Telemetry dashboard for Horizon provides access to the telemetry data stored in InfluxDB for the control plane and data plane nodes.

With the admin role, you have access to the Telemetry dashboard of Horizon.

To access the control and data plane nodes telemetry:

1. Log in to Horizon as admin.
2. On the top-right of the Horizon web UI, click Telemetry.
3. In the Telemetry tab, click Control nodes or Compute nodes.

   The tabs contain diagrams and tables displaying the control plane and data plane nodes telemetry with the disk, CPU, and RAM usage on a per-node basis.

With the non-admin role, you have access to the Telemetry data overview of instances on a per-tenant basis.

To access the telemetry overview of instances:

1. Log in to Horizon.
2. On the top-right of the Horizon web UI, click Telemetry.
3. In the Telemetry tab, click Overivew.

   The table displays the interim statistics on CPU utilization, network transmit/receive, and disk write/read operations on a per-instance basis.

The Monitoring dashboard displays the StackLight health status events stored in Sensu. A user with the admin role can visualize those events the same way as in Uchiwa.

To access the Monitoring dashboard:

1. Log in to Horizon as admin.
2. On the top-right of the Horizon web UI, click Monitoring.

The Monitoring dashboard contains several tabs:

- The Current Events tab displays the details of the warnings and critical events, if any, stored in Sensu. The tab contains the Resolve Events and Recheck Events options.
- The Event Stashes tab provides access to the Sensu stash data stored in the Redis key/value store. In this store, you can create and access arbitrary JSON data (the value) indexed by an arbitrary path (the key). For details on the Sensu key/value store, see Sensu documentation.

- The Service Checks tab provides access to the Sensu checks. From this tab, you can issue a check request, silence or unsilence a check. For details on the Sensu checks, see Sensu documentation.
- The Monitored Clients tab provides access to the list of registered Sensu clients along with their service checks subscription. A Sensu client is materialized by either a Sensu monitoring agent running on every node or a Sensu proxy client. Typically, the StackLight Aggregator plays the role of a Sensu proxy client. It registers both the 01-node-clusters and the 00-top-clusters virtual clients, which represent the MCP service clusters, and the MCP top-level clusters respectively. For details on the Sensu client, see Sensu documentation.
- The Aggregates tab provides access to the list of the Sensu check aggregates. By default, StackLight does not have a configuration for the Sensu check aggregates. For details on the Sensu check aggregates, see Sensu documentation.
- The Monitoring Status tab provides status information on the Sensu monitoring framework itself.

Seealso

The Install Horizon plugins subsection of the Install StackLight section in the MCP Deployment Guide

StackLight alerts

The alerting system of StackLight for Mirantis Cloud Platform (MCP) is conceptually similar to the alerting system of StackLight for Mirantis OpenStack but has several implementation changes that were required to accommodate the constraints imposed by the MCP Configuration Management system based on Salt and Reclass:

- The declarative definition of the alarms, aggregation rules, and correlation policies are now collated in the heka.yml file of the Salt formulas. We globally refer to them as built-in alarms. For example:
    - The built-in alarms for Nova are stored in heka.yml of the salt-formula-nova GitHub project.
    - The built-in alarms for RabbitMQ are stored in heka.yml of the salt-formula-rabbitmq GitHub project.
- The YAML structure of the alarms, correlation, and aggregation rules has changed. The new structure is described below.

The built-in alarms defined in the formulas are somewhat considered immutable by the engineering. In principle, they should always be relevant regardless of the deployment model and as such, should not be altered even though it is technically possible to override them in the user Reclass model. The built-in alarms are expected to detect all the critical conditions occurring on the MCP and as such, should always be there and not tampered.

In addition to the built-in alarms, it is possible to define custom alarms in the user Reclass model that would be more relevant to a specific deployment. The severity level attached to the built-in alarms is generally for the most critical conditions. A deployer may decide to create alarms in the Reclass model to track conditions of a lower severity level (like warning) that he would like a cloud operator to be informed of.

Both the built-in alarms defined in the formulas and the custom alarms defined in the user Reclass model use the same declarative YAML structure.

Seealso

StackLight alarm structure

StackLight alarm structure

The most effective way to explain the structure of an alarm in StackLight for Mirantis Cloud Platform is to use an example. Consider the alarms defined for Nova within the heka.yml file of the Nova formula. This example illustrates the alerting capabilities of StackLight for Mirantis Cloud Platform because it has a combination of local and remote checks and measurements as well as different levels of aggregation up to the control-plane and data-plane top-level clusters.

Alarms for the Local Metric Collector

The alarms for the metrics collected locally to the node are defined in the metric_collector section. They are evaluated by the Local Metric Collector.

The definition of an alarm begins with the definition of triggers that are defined in the trigger subsection which is a dictionary of alarm triggers.

A trigger executes the statistical function defined by the function key against metric time-series defined by the metric key for those metrics that match the field key-value pairs. The resulting value is compared to a threshold defined by the threshold key using a relational operator defined by the relational_operator key. The metric time-series are stored in the memory of Heka. Their size is proportional to a sliding window defined by the window key.

The following example describes the triggers set in the metric_collector section:

```
metric_collector:
 trigger:
   nova_logs_error:
     description: 'Too many errors have been detected in Nova logs'
     severity: warning
     no_data_policy: okay
     rules:
     - metric: log_messages
       field:
         service: nova
         level: error
       relational_operator: '>'
       threshold: 0.1
       window: 70
       periods: 0
```

```
        function: max
    {%- if pillar.nova.controller is defined %}
    nova_api_local_endpoint:
      description: 'Nova API is locally down'
      severity: down
      rules:
      - metric: openstack_check_local_api
        field:
          service: nova-api
        relational_operator: '=='
        threshold: 0
        window: 60
        periods: 0
        function: last
    {%- endif %}
```

The following is the description of parameters described in the example above:

nova_logs_error:
: Trigger used to monitor the rate of errors we have in the logs for Nova.

severity: warning
: Severity of the trigger.

no_data_policy: okay
: Defines the severity level to apply if no metrics are received during the ticker interval. In this example, the policy says that it is okay. By default, the severity is unknown.

metric: log_messages
: Metric obtained from the connection between the Log Collector and the Local Metric Collector. It is ingested in the Heka pipeline the same way as the collectd metrics.

field:
: The trigger is evaluated only if the fields of the Heka message match. In this example, service must be nova and severity level must be error.

threshold: 0.1
: The trigger is evaluated to true if the rate of error exceeds 0.1 that is more than 6 errors per minute.

{%- if pillar.nova.controller is defined %}
: Jinja templating conditional branch allows specifying the triggers that only apply to controller nodes.

nova_api_local_endpoint:
: The second alarm trigger to check the value of openstack_check_local_api metric for the nova-api service. The trigger is evaluated to true if the value is 0 meaning that the service is down.

The alarm subsection contains a dictionary of alarms. The alarms are composed of an array of triggers and a dictionary of dimensions. The example below shows several alarms defined for the Nova formula:

- The nova_logs_compute alarm is created in Heka if it is a compute node.
- The nova_logs alarm and the nova_api_endpoint alarms are created if it is a controller node.

Important

An alarm is turned into a health status metric (an AFD) decorated with dimensions. The use of dimensions by the correlation and aggregation process is shown below. Finally, an alarm has an alerting property defined by the alerting key. The available values for this property can be either enabled, disabled, or enabled_with_notification:

- An alarm that is enabled is turned into an alert (passive check) sent to Nagios.
- An alarm that is enabled_with_notification is turned into an alert notification using any of notification schemes defined for Nagios.

The following example describes the alarm subsection for the triggers described above:

```
alarm:
  {%- if pillar.nova.compute is defined %}
  nova_logs_compute:
    alerting: enabled
    triggers:
    - nova_logs_error
    dimension:
      service: nova-logs-compute
  {%- endif %}
  {%- if pillar.nova.controller is defined %}
  nova_logs:
    alerting: enabled
    triggers:
    - nova_logs_error
    dimension:
      service: nova-logs
  nova_api_endpoint:
    alerting: enabled
    triggers:
    - nova_api_local_endpoint
    dimension:
      service: nova-api-endpoint
  {%- endif %}
```

The following is the description of parameters described in the example above:

nova_logs_compute:**,** nova_logs_error
: These alarms have only one trigger but there could be more.

service: nova-logs-compute

The resulting status metric will be decorated with a dimension service set to nova-logs-compute. This dimension will be used to aggregate alarms.

Alarms for the Remote Metric Collector

The alarms for the metrics collected remotely are defined in the remote_collector section of the service formula. They are evaluated by the Remote Metric Collector.

The following example describes the remote_collector section structure example that is identical to the structure of the local metric_collector section:

```
remote_collector:
 trigger:
   {%- if pillar.nova.controller is defined %}
   nova_api_check_failed:
     description: 'Endpoint check for nova-api is failed'
     severity: down
     rules:
     - metric: openstack_check_api
       field:
         service: nova-api
       relational_operator: '=='
       threshold: 0
       window: 60
       periods: 0
       function: last
   {%- for nova_service in ('cert', 'consoleauth', 'conductor', 'scheduler') %}
   nova_{{ nova_service }}_two_up:
     description: 'There is one or more {{ nova_service }} down'
     severity: warning
     rules:
     logical_operator: 'and'
     - metric: openstack_nova_services
       field:
         service: {{ nova_service }}
         state: up
       relational_operator: '>='
       threshold: 2
       window: 60
       periods: 0
       function: last
     - metric: openstack_nova_services
       field:
         service: {{ nova_service }}
         state: down
       relational_operator: '>'
       threshold: 0
       window: 60
       periods: 0
```

```
      function: last
  nova_{{ nova_service }}_one_up:
    description: 'There is only one Nova {{ nova_service }} up left'
    severity: critical
    rules:
    logical_operator: 'and'
    - metric: openstack_nova_services
      field:
        service: {{ nova_service }}
        state: up
      relational_operator: '=='
      threshold: 1
      window: 60
      periods: 0
      function: last
    - metric: openstack_nova_services
      field:
        service: {{ nova_service }}
        state: down
      relational_operator: '>'
      threshold: 0
      window: 60
      periods: 0
      function: last
  nova_{{ nova_service }}_zero_up:
    description: 'All Nova {{ nova_service }}s are down'
    severity: down
    rules:
    - metric: openstack_nova_services
      field:
        service: {{ nova_service }}
        state: up
      relational_operator: '=='
      threshold: 0
      window: 60
      periods: 0
      function: last
  {%- endfor %}
```

The following is the description of parameters described in the example above:

{%- if pillar.nova.controller is defined %}

This trigger applies only to the controller nodes. However, the resulting Heka plugins configuration is exported to the remote metric collector using Salt Mine.

{%- for nova_service in ('cert', 'consoleauth', 'conductor',**'scheduler') %}**``

Jinja templating allows defining the same trigger for a collection of services in a loop.

logical_operator:

A logical operator can be used to describe a composition of triggering rule conditions. In this example, the trigger evaluates to true if 2 rule conditions are met.

Likewise, the remote_collector section has the alarm subsection for the triggers defined in the remote_collector subsection. In this subsection, the nova_{{ nova_service }}: alarm has three triggers. The triggers are evaluated in the order they are listed in the array. The alarm evaluates to true and returns when a trigger evaluates to true. The resulting status metrics will contain a description of what triggered the alarm.

Example:

```
alarm:
   {%- if pillar.nova.controller is defined %}
   nova_api_check:
     alerting: enabled
     triggers:
     - nova_api_check_failed
     dimension:
       service: nova-api-check
   {%- for nova_service in ('cert', 'consoleauth', 'conductor', 'scheduler') %}
   nova_{{ nova_service }}:
     alerting: enabled
     triggers:
     - nova_{{ nova_service }}_zero_up
     - nova_{{ nova_service }}_one_down
     - nova_{{ nova_service }}_two_up
     dimension:
       service: nova-{{ nova_service }}
   {%- endfor %}
   nova_free_vcpu:
     alerting: enabled
     triggers:
     - nova_total_free_vcpu_warning
     dimension:
       service: nova-free-vcpu
   nova_free_memory:
     alerting: enabled
     triggers:
     - nova_total_free_memory_warning
     dimension:
       service: nova-free-memory
   nova_compute:
     alerting: enabled
     triggers:
     - nova_compute_all_down
     - nova_compute_majority_down
     - nova_compute_some_down
     dimension:
       service: nova-compute
```

```
  {%- endif %}
  {%- endif %}
```

Aggregation and correlation

The Aggregator section defines how the health status metrics (AFD metrics) are aggregated together to infer a health status at the cluster level. Each aggregation is turned into a cluster health status metric (GSE metric) decorated with dimensions. The aggregation process is reentrant. Therefore, there can be several levels of aggregation and several metrics representing the health status of a hierarchy of clusters.

Like an alarm, an aggregation has an alerting property defined by the alerting key. The value of the alerting key can be either enabled, disabled, or enabled_with_notification. The health status aggregation is performed using different correlation policies depending on the characteristics of the clusters. The correlation policies are defined in the Heka formula independently of the services formulas. Like for the alarms, it should be possible to define new policies in the user Reclass model, if needed.

The following example describes the aggregations defined for Nova within the heka.yml file of the Nova formula.

```
aggregator:
 alarm_cluster:
   nova_logs_compute:
     policy: majority_of_node_members
     group_by: hostname
     alerting: enabled
     match:
       service: nova-logs-compute
     members:
     - nova_logs_compute
     dimension:
       service: nova-data
       nagios_host: 01-service-clusters
   nova_logs:
     policy: availability_of_members
     group_by: hostname
     alerting: enabled
     match:
       service: nova-logs
     members:
     - nova_logs
     dimension:
       service: nova-control
       nagios_host: 01-service-clusters
   nova_api_endpoint:
     policy: availability_of_members
     group_by: hostname
     alerting: enabled
```

```
    match:
      service: nova-api-endpoint
    members:
    - nova_api_endpoint
    dimension:
      service: nova-control
      nagios_host: 01-service-clusters
  nova_api_check:
    policy: availability_of_members
    alerting: enabled
    match:
      service: nova-api-check
    members:
    - nova_api_check
    dimension:
      service: nova-control
      nagios_host: 01-service-clusters
  {%- for nova_service in ('cert', 'consoleauth', 'conductor', 'scheduler') %}
  nova_{{ nova_service }}:
    policy: availability_of_members
    alerting: enabled
    match:
      service: nova-{{ nova_service }}
    members:
    - nova_{{ nova_service }}
    dimension:
      service: nova-control
      nagios_host: 01-service-clusters
  {%- endfor %}
  nova_free_vcpu:
    policy: majority_of_node_members
    alerting: enabled
    match:
      service: nova-free-vcpu
    members:
    - nova_free_vcpu
    dimension:
      service: nova-data
      nagios_host: 01-service-clusters
  nova_free_memory:
    policy: highest_severity
    alerting: enabled
    match:
      service: nova-free-memory
    members:
    - nova_free_memory
    dimension:
      service: nova-data
      nagios_host: 01-service-clusters
  nova_compute:
    policy: highest_severity
    alerting: enabled
    match:
      service: nova-compute
    members:
    - nova_compute
    dimension:
      service: nova-data
      nagios_host: 01-service-clusters
  nova_control:
    policy: highest_severity
    alerting: enabled_with_notification
    match:
      service: nova-control
    members:
    - nova_logs
    - nova_api_endpoint
    - nova_api_check
    {%- for nova_service in ('cert', 'consoleauth', 'conductor', 'scheduler') %}
    - nova_{{ nova_service }}
    {%- endfor %}
    dimension:
      cluster_name: nova-control
      nagios_host: 00-top-clusters
  nova_data:
    policy: highest_severity
    alerting: enabled_with_notification
    match:
      service: nova-data
    members:
    - nova_logs_compute
    - nova_free_vcpu
    - nova_free_memory
    - nova_compute
    dimension:
      cluster_name: nova-data
      nagios_host: 00-top-clusters
```

The following is the description of parameters described in the example above:

alarm_cluster:

This subsection contains a dictionary of aggregation rules.

policy: majority_of_node_members

The policy applied to the aggregation. In this case, the majority_of_node_members policy is applied. The severity level applied to this aggregation is the result of applying this policy to the nova_logs_compute AFDs traversing the Aggregator pipeline. In practice, if there a majority of compute nodes that are warning because too many errors are found in the logs, then the resulting GSE metric will be warning too.

group_by: hostname

  The incoming AFDs are bucketed by hostname. The policy is applied across the buckets as opposed to the entire flow of AFDs regardless of the nodes they were emitted. Each host bucket stores only the highest severity level.

match: service: nova-logs-compute

  Only the AFDs that match the service dimension and the nova-logs-compute value are matched.

members: - nova_logs_compute

  The members list is a mere implementation artifact that may be removed in the future. For now, it contains the list of all the AFDs that must be aggregated.

dimension:

  The dimension(s) that decorates the GSE metric resulting from this aggregation. The GSE dimensions are used as classifiers for higher level aggregations.

nagios_host: 01-service-clusters

  The dimension nagios_host is used to configure the Nagios dashboard. The passive check for the GSE resulting from that aggregation will be exposed under the 01-service-clusters virtual host.

nova_control:

  It is an aggregation of aggregations (an aggregation of GSEs). The resulting GSE represents the health status of a so-called top-level cluster.

policy: highest_severity

  This policy is applied to the top-level cluster. The resulting GSE will inherit the highest severity level of its members. For example, if nova_api_endpoint is critical and it is the highest severity level, then nova-control is critical too.

alerting: enabled_with_notification

  A Nagios alert notification should be sent.

Create or modify an alarm trigger

The built-in alarms of StackLight are provided with relevant defaults that in most cases can be applied to any deployment. But in practice, you may want to customize some alarms. For example, a threshold defined in an alarm trigger could be too high or too low for your deployment, hence resulting in flapping alerts. For this reason, the alerting system of StackLight allows changing the definition of the triggers of the built-in alarms within the cluster Reclass model.

Example

If you detect that the keystone_response_time alarm is flapping, the keystone_response_time_duration trigger may be defined improperly.

By default, the following parameters are defined for Keystone in /etc/salt/grains.d/heka on the controller nodes:

- For the keystone_response_time alarm:

```
keystone_response_time:
  alerting: enabled
  dimension:
    service: keystone-response-time
  triggers:
  - keystone_response_time_duration
```

- For the keystone_response_time_duration trigger:

```
keystone_response_time_duration:
  description: Keystone API is too slow
  no_data_policy: okay
  rules:
  - field:
      http_method: == GET || == POST
      http_status: == 2xx
    function: max
    metric: openstack_keystone_http_response_times
    periods: 0
    relational_operator: '>'
    threshold: 0.3
    value: upper_90
    window: 60
  severity: warning
```

The threshold value is 0.3 by default. It means that every time the upper_90 value of the openstack_keystone_http_response_times metric is greater than 300 milliseconds, the alarm evaluates to true which results in an alert in Sensu and/or Nagios.

> Note
>
> In this example, we use the 90 percentile value to discard the 10% outliers values from the trigger evaluations.

To change the threshold value to 500 milliseconds, you must override the trigger in the Reclass model and reapply the configuration of the alarms on the controller nodes where this trigger is evaluated.

To create an alarm trigger:

1. On the Salt Master node, create a Reclass file under the cluster Reclass model with a new definition of the trigger:

```
# pwd
/srv/salt/reclass/classes/cluster/mcp-lab-advanced/stacklight

# less ./custom_alarms.yml
```

```
parameters:
  heka:
    metric_collector:
      trigger:
        # Override the openstack_keystone_http_response_times
        # trigger to relax the threshold value
        keystone_response_time_duration:
          description: 'Keystone API is too slow'
          severity: warning
          no_data_policy: okay
          rules:
          - metric: openstack_keystone_http_response_times
            field:
              http_method: '== GET || == POST'
              http_status: '== 2xx'
            relational_operator: '>'
            threshold: 0.5
            window: 60
            periods: 0
            value: upper_90
            function: max
```

2. Add the custom_alarms class to the control class:

```
# pwd
/srv/salt/reclass/classes/cluster/mcp-lab-advanced/openstack

# less ./control.yml
classes:
- system.linux.system.lowmem
[...]
- system.heka.alarm.openstack_control
- cluster.mcp-lab-advanced
- cluster.mcp-lab-advanced.custom_alarms
```

3. Remove the Heka grains on the controller nodes:

```
# salt "ctl*" file.remove /etc/salt/grains.d/heka
```

4. Recreate the Heka grains:

```
# salt "ctl*" state.sls heka
```

5. Restart the metric collectors on the controller nodes.

Now, the Heka grains in etc/salt/grains.d on the controller nodes should be updated with the new trigger definition as follows:

```
[...]
heka:
  [...]
  metric_collector:
    [...]
    trigger:
      [...]
      keystone_response_time_duration:
        description: Keystone API is too slow
        no_data_policy: okay
        rules:
        - field:
            http_method: == GET || == POST
            http_status: == 2xx
          metric: openstack_keystone_http_response_times
          periods: 0
          relational_operator: '>'
          threshold: 0.5
          value: upper_90
          window: 60
        severity: warning
```

Seealso

Create or modify an alarm

Create or modify an alarm

In StackLight, the alarms can contain a dictionary of triggers. You can add or remove triggers to existing alarms as required.

Example

By default, the keystone_response_time alarm contains only one keystone_response_time_duration trigger. To raise the criticality of the alarm when Keystone stops responding within an acceptable delay, you can either create a new alarm called custom_keystone_response_time or add a new trigger to the existing alarm. In this example, the acceptable delay must be below 1 second in the 90 percentile of all the response time samples.

To create a custom_keystone_response_time alarm:

1. Log in to the Salt Master node.
2. Create the custom_alarms.yml Reclass file under the cluster Reclass model with a new definition of the trigger as described in the Create or modify an alarm trigger section.
3. On every controller node, change the directory to /etc/salt/grains.d/heka.
4. Verify that the custom_alarms.yml file for Keystone contains the following parameters:

```
parameters:
  heka:
    metric_collector:
      alarm:
        # Override the keystone_response_time alarm
        # with a new trigger
        custom_keystone_response_time:
        alerting: enabled
        dimension:
          service: keystone-response-time
        triggers:
        - keystone_response_time_duration_critical
        - keystone_response_time_duration_warning
       trigger:
         # Create 2 triggers. One a critical level and
         # one at warning level.
         keystone_response_time_duration_critical:
           description: 'Keystone API is too slow'
           severity: warning
           no_data_policy: okay
           rules:
           - metric: openstack_keystone_http_response_times
             field:
               http_method: '== GET || == POST'
               http_status: '== 2xx'
             relational_operator: '>='
             threshold: 1
             window: 60
             periods: 0
             value: upper_90
             function: max
         keystone_response_time_duration_warning:
           description: 'Keystone API is too slow'
           severity: warning
           no_data_policy: okay
           rules:
           - metric: openstack_keystone_http_response_times
             field:
               http_method: '== GET || == POST'
               http_status: '== 2xx'
             relational_operator: '>='
             threshold: 0.5
             window: 60
             periods: 0
             value: upper_90
             function: max
```

5. Execute the stacklight_monitor_install.sh script to verify that the configuration change is properly applied to the entire alerts processing system of StackLight including Sensu and/or

Nagios. This procedure is necessary since creating a new alarm has a larger configuration impact on the alerting system of StackLight than simply creating or modifying a trigger.

1. Log in to the Salt Master node.
2. Change the directory to /srv/salt/reclass/scripts.
3. Run the stacklight_monitor_install.sh script.

The script updates the grains and the pillars used for the configuration of the Collectors, the Aggregator, Sensu, and/or Nagios and restarts the StackLight services with the proper configuration.

Seealso

Create or modify an alarm trigger

Create or modify an aggregation

To activate a new alarm in a service-level and/or top-level cluster, update the aggregation rules. The aggregation rules are executed by the Aggregator that runs on the monitoring cluster.

The modification of the built-in aggregation rules is similar to the modification of a built-in trigger or alarm. You can either create a new aggregation rule or override an existing one.

Example

The following example shows how to override the Keystone service cluster aggregation rule to incorporate the new alarm created in the Create or modify an alarm section:

1. Log in to the Salt Master node.
2. Create the custom_aggregation file:

```
# pwd
/srv/salt/reclass/classes/cluster/mcp-lab-advanced/stacklight

# cat ./custom_aggregation.yml
parameters:
  heka:
    aggregator:
      alarm_cluster:
        keystone:
          alerting: enabled_with_notification
          dimension:
            cluster_name: keystone
            nagios_host: 00-top-clusters
          match:
            service: keystone
          members:
          - custom_keystone_response_time
          - keystone_logs
          - keystone_public_api_endpoint
          - keystone_public_api_check
          policy: highest_severity

In this file, the new ``custom_keystone_response_time`` alarm is added
to the list of alarm members of the Keystone cluster. For this cluster,
the Aggregator will correlate the new alarm with other alarms of the
cluster using the ``highest_severity`` policy.
```

3. Add the new custom_aggregation class to the server class under StackLight since the Aggregator runs on the StackLight monitoring cluster:

```
# pwd
/srv/salt/reclass/classes/cluster/mcp-lab-advanced/stacklight

# less ./server.yml
classes:
- system.linux.system.repo.grafana
[...]
- system.nagios.server.single
- cluster.mcp-lab-advanced
- cluster.mcp-lab-advanced.custom_aggregation
```

4. On the Salt Master node, run the stacklight_monitor_install.sh script to apply changes.

Seealso

- Aggregation and correlation
- Create or modify an alarm trigger
- Create or modify an alarm

Seealso

- StackLight alerts

Create or modify a metric

StackLight provides a large collection of metrics for the monitoring of the entities supported out of the box. StackLight can also be easily extended to support new service check and measurement metrics without programming Lua plugins. A special Lua decoder plugin of the Metric Collector allows turning all the telemetry data samples it receives from collectd into standard metric messages. These samples have a JSON payload that is formatted by the Write HTTP plugin of collectd. An enhancement of the Lua decoder plugin allows handling transparently all the samples that comply with clearly defined mapping conventions.

Note

A special Lua decoder for collectd is defined in the support metadata of the Heka formula (meta/heka.yml file) as follow:

```
metric_collector:
  decoder:
    collectd:
      engine: sandbox
      module_file: /usr/share/lma_collector/decoders/collectd.lua
```

Mapping conventions of StackLight metric data model

The samples injected by a collectd plugin can be mapped automatically into standard metric messages as long as they comply with the StackLight metric data model conventions as described in the table below.

StackLight metric data model attributes	Mapping of collectd samples attributes
Timestamp	The timestamp of the metric built out of the time key value in nanoseconds.
Logger	Always collectd.
Type	Always metric.
Severity	Always 6.
Hostname	The hostname where the metric is collected. It is built out of the host key value.
Payload	The original JSON payload.
Fields[name]	The name of the metric built out of the plugin key value and a concatenation of optional key values (if not null) separated by the _ delimiter. sample['plugin']_<'sample['plugin_instance']>_<sample['type']>_ <sample['type_instance']>
Fields[Type]	Type of metric that can be either gauge, counter, derive, or absolute built out of the dstypes array of values. One metric is created per value of the dstypes array.
Fields[Value]	The value of the metric built out of the the values array of values. One metric is created per value of the values array.

Fields[Interval]	The sampling interval of the metric built out of the interval key value.
Fields[Region]	The OpenStack region for metrics collected in OpenStack clusters.
Fields [Environment_label]	An MCP environment name.
Fields[Tag_fields]	An array of tags built out of the meta key/value pairs array.
Fields['hostname']	The hostname that the metric applies to built out of the host key value. If the sample contains a meta with a discard_hostname key value that is not null, then Fields['hostname'] is reset to null.

Example

When you create a new collectd plugin that you integrate with the Metric Collector, this plugin complies with the mapping conventions described above. The following example shows the resulting metric message in the reStructuredText format. For the illustration purposes, the collectd plugin generates samples with a random number.

```
:Timestamp: 2017-03-16 10:46:03.349999872 +0000 UTC
:Type: metric
:Hostname: localhost
:Pid: 19884
:Uuid: caf25c89-9a28-4899-a993-2747a9631441
:Logger: collectd
:Payload: {"type":"gauge","values":[27001],"type_instance":"",\
"meta":{"foo":"bar"},"dsnames":["value"],"plugin":"collectd_random",\
"time":1489661163.35,"interval":10,"host":"localhost",\
"dstypes":["gauge"],"plugin_instance":"random_value"}
:EnvVersion:
:Severity: 6
:Fields:
    | name:"type" type:string value:"gauge"
    | name:"source" type:string value:"collectd_random"
    | name:"tag_fields" type:string value:["foo","hostname"]
    | name:"name" type:string value:"collectd_random_random_value"
    | name:"hostname" type:string value:"localhost"
    | name:"value" type:double value:27001
    | name:"environment_label" type:string value:"mk22-lab-dvr.local"
    | name:"interval" type:double value:10
    | name:"foo" type:string value:"bar"
```

The Payload attribute contains the original collectd JSON payload (highlighted in the example) received by the Metric Collector. This message is injected into the Heka pipeline every 10 seconds ticker interval.

Now, you can create a new alarm trigger as described in Create or modify an alarm trigger. It will raise a warning health status when the maximum value of the samples goes above 1000 within the last 60 seconds window.

This example is fictitious but it shows that custom metrics can be evaluated exactly the same way as regular metrics.

```
dumb_random_value_trigger:
        description: monitor max value of random number
        rules:
        - field:
            foo: == bar
          function: max
          metric: collectd_random_random_value
          periods: 0
          relational_operator: '>'
          threshold: 1000
          value: value
          window: 60
        severity: warning
```

Register a custom collectd plugin in Metric Collector

Changing the configuration of the Metric Collector in MCP requires modifications in the Salt formulas and/or Reclass models.

The following example shows how to add a new collectd plugin written in Python to the Local Metric Collector. This operation requires changing the support metadata in the following Salt formulas:

1. The collectd formula.
2. The formula of the service that you want the new metric to be collected for.

In the following example, the new metric is to be collected for the Linux operating system, since this type of metric is useful for the monitoring of the nodes. To complete this example, we therefore need to modify the support metadata of both collectd and Linux formulas. After the change, the new collectd plugin should be installed on all the nodes of the MCP environment where the Linux formula is applied when collectd is enabled. The configuration changes described below work exactly the same way as for other formulas than Linux.

Caution!

In an MCP environment provisioned by the MCP bootstrap, all formulas are extracted from the MCP packages under the /srv/salt/env directory of the Salt Master node. However, those formulas are not supposed to be modified directly, because all changes are erased every time the formula is updated. Instead, the customization is supposed to be done in a versioned fork of the community formula as a site customization.

Getting into the details of managing site formulas customization is beyond the scope of this document.

To apply the configuration changes for the Metric Collector:

1. Apply the changes in the Salt Linux formula.
2. Apply the changes in the Salt collectd formula.
3. Re-run the collectd state.

Seealso

Apply changes to the Linux and collectd formulas

Apply changes to the Linux and collectd formulas

Changing the configuration of the Metric Collector requires modifications in the Salt formulas and/or the Reclass models. After you register your custom collectd plugin as described in Register a custom collectd plugin in Metric Collector, apply the changes to the corresponding Salt formulas.

To apply changes to the Linux and collectd formulas:

1. Log in to the Salt Master node.
2. In the support metadata directory, append the collectd_random fragment to the collectd.yml file.

   ```
   root@cfg01:/srv/salt/env/prd/linux/meta# cat << EOF >> collectd.yml
     collectd_random:
       plugin: python
       template: linux/files/collectd_random.conf
   EOF
   ```

3. In /srv/salt/env/prd/linux/files, create the new collectd plugin configuration file:

   ```
   root@cfg01:/srv/salt/env/prd/linux/files# cat << EOF > collectd_random.conf
   Import "collectd_random"

   <Module "collectd_random">
   </Module>
   EOF
   ```

4. In files/plugin, add the new collectd plugin to the Salt collectd formula.
5. Apply the collectd state. The resulting configuration changes that should be applied by Salt are as follows:

   ```
   # salt "*" state.sls collectd true=false
   cfg01.mk22-lab-dvr.local:
   ----------
   [...]
   ----------
             ID: collectd_client_grain
   ```

```
    Function: file.managed
        Name: /etc/salt/grains.d/collectd
      Result: True
     Comment: File /etc/salt/grains.d/collectd updated
     Started: 15:48:36.653025
    Duration: 16.887 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -26,6 +26,9 @@
                           metric_collector:
                             match: heka.*metric_collector
                         template: collectd/files/collectd_processes.conf
                  +    collectd_random:
                  +      plugin: python
                  +      template: linux/files/collectd_random.conf
                       linux_network_netlink:
                         ignore_selected: false
                         interfaces:
----------
[...]
----------
          ID: /etc/collectd/conf.d/collectd_python.conf
    Function: file.managed
      Result: True
     Comment: File /etc/collectd/conf.d/collectd_python.conf updated
     Started: 15:48:38.134810
    Duration: 53.823 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -14,5 +14,9 @@
                     Url "http://localhost:8888"
                   </Module>

                  +  Import "collectd_random"
                  +
                  +<Module "collectd_random">
                  +</Module>

                   </Plugin>
----------
[...]
----------
```

```
      ID: collectd_service
Function: service.running
    Name: collectd
  Result: True
 Comment: Service restarted
 Started: 15:48:38.418323
Duration: 333.999 ms
 Changes:
          ----------
          collectd:
              True
```

6. Restart the collectd service.

Now, the new metric should be available in the operational insights pipeline of StackLight.

Seealso

- Mapping conventions of StackLight metric data model
- Register a custom collectd plugin in Metric Collector

Seealso

- Create or modify an alarm trigger
- Create or modify an alarm
- Create or modify an aggregation

Seealso

The Install StackLight section in the MCP Deployment Guide

Manage Ceph clusters using Decapod

Using Decapod, you can deploy Ceph clusters and manage their lifecycle. All the management functionality is distributed using configurable plugins, called playbooks.

This section describes how to deploy a Ceph cluster, perform various operations using Decapod plugins, back up and restore Decapod, generate a diagnostic snapshot, and others.

Configuration files

Decapod supports a number of configuration files you may want to propagate into a container. This section describes these files.

SSH private key

The ansible_ssh_keyfile.pem file is an SSH private key used by Ansible to connect to Ceph nodes. Decapod uses Ansible to configure remote machines. Ansible uses SSH to connect to remote machines. Therefore, it is required to propagate SSH private key to Decapod. If you do not have a prepared SSH private key, generate a new one as described in Create SSH keys.

After you generate the key, copy it to the top level of the source code repository. The file name must be ansible_ssh_keyfile.pem and the format of the file must be PEM.

Warning

Keep the key private.

SSL certificates

The following files are the SSL certificates:

- ssl.key - Private key for SSL/TLS certificate. Used by web UI.
- ssl.crt - Signed certificate for SSL/TLS. Used by web UI.
- ssl-dhparam.pem - Diffie-Hellman ephemeral parameters for SSL/TLS. This enables perfect forward secrecy for secured connection.

If you do not have such certificates, generate new ones as described in OpenSSL Essentials and Forward Secrecy & Diffie Hellman Ephemeral Parameters. All SSL keys should be in the PEM format. Place the SSL files to the top level of your source code repository.

Warning

Keep the key private. Do not use self-signed certificates for a production installation.

Decapod configuration file

Decapod configuration is performed within the config.yaml file in YAML format. Decapod searches for the configuration file in several locations in the following order:

- $(pwd/decapod.yaml
- $XDG_CONFIG_HOME/decapod/config.yaml
- :`$HOME/.decapod.yaml
- /etc/decapod/config.yaml
- Default configuration file of the decapod_common package

If a configuration file was found and parsed before, other alternatives will not be used. Therefore, if you have the default configuration file in /etc/decapod/config.yaml then placing the configuration to $XDG_CONFIG_HOME/decapod/config.yaml will override the default one. For details, see XDG Base Directory Specification.

Default configuration in containerized Decapod stack is placed in /etc/decapod/config.yaml.

Decapod config.yaml example

The following is an example of the default Decapod configuration file for containers:

```
---
common:
  password:
    length: 10
    time_cost: 10
    memory_cost: 2048
    parallelism: 3
    hash_len: 32
    salt_len: 16
  password_reset_ttl_in_seconds: 86400  # 1 day
  email:
    enabled: false
    from: "noreply@mirantis.com"
    host: "localhost"
    port: 25
    login: ""
    password: ""

# Options here are Flask options so please check
# http://flask.pocoo.org/docs/0.11/config/#builtin-configuration-values
api:
  debug: false
  testing: false
  logger_name: "decapod.decapod_api.wsgi"
  logger_handler_policy: "never"
  json_sort_keys: faluse
  jsonify_prettyprint_regular: false
  json_as_ascii: false
  pagination_per_page: 25
  server_discovery_token: "26758c32-3421-4f3d-9603-e4b5337e7ecc"
  reset_password_url: "http://127.0.0.1/password_reset/{reset_token}/"
  token:
    ttl_in_seconds: 1800
  logging:
    propagate: true
    level: "DEBUG"
    handlers:
      - "stderr_debug"
  auth:
    type: native
    parameters: {}
    # type: keystone
    # parameters:
    #   auth_url: http://keystone:5000/v3
    #   username: admin
    #   password: nomoresecret
    #   project_domain_name: default
    #   project_name: admin
    #   user_domain_name: default

controller:
  pidfile: "/tmp/decapod-controller.pid"
  daemon: false
  ansible_config: "/etc/ansible/ansible.cfg"
  # 0 worker_threads means that we will have 2 * CPU count threads
  worker_threads: 0
  graceful_stop: 10
  logging:
    propagate: true
    level: "DEBUG"
    handlers:
      - "stderr_debug"

cron:
  clean_finished_tasks_after_seconds: 2592000  # 60 * 60 * 24 * 30; 30 days

db:
  uri: "mongodb://database:27017/decapod?ssl=true"
  connect: false
  connect_timeout: 5000  # ms, 5 seconds
  socket_timeout: 5000  # ms, 5 seconds
  pool_size: 50
  gridfs_chunk_size_in_bytes: 261120  # 255 kilobytes

plugins:
  alerts:
    enabled: []
    email:
      enabled: false
      send_to:
        - "bigboss@example.com"
      from: "errors@example.com"
  playbooks:
    disabled:
      - hello_world

# Default Python logging is used.
# https://docs.python.org/2/library/logging.config.html#dictionary-schema-details
logging:
  version: 1
  incremental: false
  disable_existing_loggers: true
  root:
    handlers: []
  filters: {}
  formatters:
    stderr_default:
      format: "%(asctime)s [%(levelname)-8s]: %(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
    stderr_debug:
      format: "%(asctime)s [%(levelname)-8s] (%(filename)15s:%(lineno)-4d): %(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
    syslog:
      format: "%(name)s %(asctime)s [%(levelname)-8s]: %(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
  handlers:
    stderr_debug:
      class: "logging.StreamHandler"
      formatter: "stderr_debug"
      level: "DEBUG"
    stderr_default:
      class: "logging.StreamHandler"
      formatter: "stderr_default"
      level: "DEBUG"
    syslog:
      class: "logging.handlers.SysLogHandler"
      formatter: "syslog"
      level: "DEBUG"
```

Seealso

Decapod config.yaml settings description

Settings description

The following tables describe the config.yaml configuration file settings:

Common settings

Setting	Description
common	Defines generic Decapod settings not related to API or controller. You can specify the following parameters: password Describes settings for Decapod key derivation function. Decapod does not store user passwords in plain text. Instead, it uses key derivation functions to calculate a cryptographic secure hash from the password. To do so, it uses the Argon2 key derivation function that is similar to the scrypt key derivation function but has a property for defense against concurrent attacks with GPUs. To change the default settings, follow the Argon2 documentation. password_reset_ttl_in_seconds Sets the TTL value in seconds for the password reset token. When resetting the password, the user gets a secret token. Consuming this token performs the actual password reset. This parameter sets the TTL of such token. The token is valid only for the specified amount of time and expires after. email Defines how to send emails from Decapod. The from parameter defines the email to set in the From field. The enabled parameter (boolean) enables or disables the email sending. If disabled, all other fields in this section are ignored.

The api section contains settings specific to the API service only. Some parameters propagate directly to Flask. For Flask settings description, see Flask documentation. The following parameters are related to Flask:

- DEBUG
- TESTING
- LOGGER_NAME
- LOGGER_HANDLER_POLICY

- JSON_SORT_KEYS
- JSON_AS_ASCII
- JSONIFY_PRETTYPRINT_REGULAR

If you are not sure which parameter to specify, use the default ones.

The following parameters of the api section are Decapod-related:

Decapod-related API settings

Setting	Description
pagination_per_page	Sets a default count of items per page in paginated listings. If the number of items is less than pagination_per_page, then fewer elements would be returned.
server_discovery_token	Defines the server discovery token. Servers found during the server discovery must have an authentication token to access the POST /v1/server API endpoint. This token does not refer to any certain user and allows accessing only the mentioned API endpoint. However, Ansible will access the remote host to gather facts and verify the access.
reset_password_url	Defines the template of the URL that will be used for generating the email during the password reset. The email sent to the user will contain this URL. Decapod will replace {reset_token} to a correct password reset token.
token	Contains configuration for authentication tokens. The ttl_in_seconds is the TTL value in seconds. This parameter applies only to newly generated tokens. This section is used only if native authentication back end is enabled. For example, Keystone integration will not use this parameter because Keystone manages its own tokens.
logging	Defines specific parameters for logging in API. Applies the parameters specified in the logging setting to API only.

auth	Configures the authentication back end used by Decapod. If not specified, the native authentication back end with default configuration is used. The type parameter defines the type of the back end to use and parameters define the back-end configuration. for details on available authentication back ends, see Authentication back ends.

Controller settings

Setting	Description

controller	Defines specific settings for the controller service. This service manages the task queue and runs Ansible for tasks. You can specify the following parameters: daemon Defines whether to run the controller service as a UNIX daemon. If you use systemd or Docker containers, set this to false. pidfile Defines the PIDFile for the daemon if the controller service is run as a daemon. ansible_config Defines the path to the default Ansible configuration to use. You can leave this parameter as is. worker_threads Defines the number of workers per controller. The controller service uses the worker pool to manage Ansible executions concurrently. The 0 value means to define this number automatically. By default, it is 2 * cpu_count. graceful_stop Defines the graceful stop for external processes in seconds. Since the controller service executes a number of processes, it cannot be stopped immediately, the processes should be correctly finished. Initially, the controller service sends the SIGTERM signal to the processes and if they do not stop after the amount of time specified in graceful_stop, the controller service stops them with SIGKILL. logging Defines specific parameters for logging in the controller service. Applies the parameters specified in the logging setting to the controller service only.

Cron settings

Setting	Description

cron	Defines Cron-related settings. You can specify the following parameters: clean_finished_tasks_after_seconds Defines the TTL for finished tasks. After the specified amount of time, the tasks will be purged from the database. This is related only to finished tasks that were completed or failed and is not related to not started tasks.

Database settings

Setting	Description

db	Defines the MongoDB-related settings, for example, how to connect to the database and some specifics of the database client configuration. You can specify the following parameters: uri Defines the URI to connect to MongoDB. For information on connection URIs, see the MongoDB documentation. connect Defines whether Decapod will connect to MongoDB immediately after initialization of a client or on the first request. We suggest that you keep this parameter value false. socket_timeout Defines the amount of time in milliseconds the driver will wait during server monitoring when connecting a new socket to a server before concluding that the server is unavailable. socket_timeout Defines the amount of time in milliseconds the driver will wait for a response after sending an ordinary (non-monitoring) database operation before concluding that a network error has occurred. pool_size Defines the maximum allowed number of concurrent connections to each connected server. Requests to a server will be blocked if there are more connections to the requested served than defined in pool_size. gridfs_chunk_size_in_bytes Defines the size of file chunk (a part of the file, stored in a separate document) for GridFS. It is 255 kilobytes by default.

Plugins and logs settings

Setting	Description

plugins	Describes what to do with plugins: disable, enable, and others. All plugins are split into 2 categories, alerts and plugins. • The alerts section contains a list of enabled alerts plugins responsible for issues alerting, for example, in case of a 500 error. Every parameter except enabled defines how to set up each alert plugin. • The playbooks section has only 1 parameter: disabled that lists the plugins that are disabled even if installed.
logging	Defines the configuration of Decapod logging. For more information on this setting and its options, see Python documentation.

Seealso

Decapod configuration file example

Authentication back ends

Decapod supports two authentication backends: the default native and the Keystone authentication back ends.

Native authentication back end

Native authentication back end uses Decapod MongoDB to store authentication tokens. Each time a user logs in to Decapod, it creates a new authentication token and stores it in the collection. Each time a user logs out, Decapod removes the corresponding token. Also, every token has a TTL value and when it expires, MongoDB deletes the token. This is performed using the MongoDB TTL indexes.

To set up the native authentication back end, place the following snippet to the api section of the config.yaml file:

```
auth:
  type: native
  parameters: {}
```

This type of back end does not require configuration.

Keystone authentication back end

Keystone authentication back end uses Keystone for authentication. This is option has a more complex setup than the default Native authentication back end.

Using the Keystone authentication back end, creating or deleting a user in Decapod will not affect Keystone and Decapod will not create or remove a user from Keystone. Decapod synchronizes the user list with Keystone every 10 minutes. So if you create, delete, or disable a user in Keystone, it will be also created, deleted, or disabled in Decapod.

To set up Keystone integration:

1. Place the following snippet to the api section of the config.yaml file:

   ```
   auth:
     type: keystone
     parameters:
       auth_url: {os_auth_url}
       username: {os_username}
       password: {os_password}
       project_domain_name: {os_project_domain_name}
       project_name: {os_project_name}
       user_domain_name: {os_domain_name}
   ```

   For details on these parameters, see the OpenStack command-line options. For the whole list of options, see v3.Password.

   Important

   Username and password should correspond to the user that has enough permissions to request tokens for other users and list them.

2. Perform initial synchronization using the admin service:

   ```
     $ docker-compose -p myprojectname exec admin decapod-admin keystone initial -h
     Usage: decapod-admin keystone initial [OPTIONS] ROLE [USER]...

       Initial Keystone sync.

       On initial sync it is possible to setup role for a user (users). If no
       usernames are given, then all users from Keystone would be synced and role
       will be applied to them.

     Options:
       -h, --help  Show this message and exit.

   Specify the role name (default is ``wheel``, which has the biggest number of
   permissions) and user login for this role.
   ```

As a result, you should be able to access Decapod and set required roles for users.

Note

Newly synchronized users from Keystone have no role.

Using the admin service, synchronization is performed by Cron, but you can execute it manually after the initial synchronization:

```
$ docker-compose -p myprojectname exec admin decapod-admin keystone sync
```

Seealso

Decapod configuration file

MongoDB certificate and key

The mongodb.pem file is the SSL/TLS pair of certificate and key, concatenated in one file. This is required for a secure connection to MongoDB. Generate this file as described in MongoDB documentation. To allow SSL/TLS on client side, verify that the configuration file has the ?ssl=true parameter in URI. For example, mongodb://database:27017/db will not use a secure connection, but mongodb://database:27017/db?ssl=true will.

Note

To use database authentication, see:

- MongoDB documentation
- MongoDB security
- Docker hub

After you have a MongoDB running with the required authentication, verify that the user and password pair is set in the configuration file. The URI should look like mongodb://user:password@database:27017/db?ssl=true.

By default, containers contain no information about users and their passwords.

Configuration files location

The table below provides the list of configuration files and their location in containers depending on the particular Docker Compose service. After changing the configuration, place the changed file into an appropriate container.

Configuration files location

Configuration file	Location
ansible_ssh_keyfile.pem	• Controller: /root/.ssh/id_rsa • Admin: /root/.ssh/id_rsa
ssl.key	• Front end: /ssl/ssl.key
ssl.crt	• Front end: /ssl/ssl.crt
ssl-dhparam.pem	• Front end: /ssl/dhparam.pem
config.yaml	• API: /etc/decapod/config.yaml • Controller: /etc/decapod/config.yaml • Admin: /etc/decapod/config.yaml
mongodb.pem	• Database: /certs/mongodb.pem
mongodb-ca.crt	• Database: /certs/mongodb-ca.crt

To specify custom files, use the docker-compose.override.yml file. For details, see Docker Compose documentation. An example of the docker-compose.override.yml file is placed in the top level of the repository.

Note

Provide the modified configuration for API, controller, and Cron services. There is no possibility to define it for all services in Docker Compose configuration version 2.

Seealso

- PEM
- YAML

Deploy an OS on a Ceph node

Warning

Decapod does not perform bare metal provisioning, OS deployment, and network setup.

The OS must support cloud-init. Also, it must be possible to run your own user data. For the available datasources for cloud-init, see Datasources. Alternatively, you can set user data using the kernel command line. For bare metal provisioning, try MAAS. This section covers the MAAS installation and OS deployment with this tool.

Generate user data for cloud-init

Prerequisites

Prior to generating the user data for cloud-init, complete the following steps:

1. Verify that your Decapod installation is up and running:

   ```
   $ docker-compose -p PROJECT ps
   ```

   All containers except decapod_database_data_1 should be in the Up state.
2. Obtain the server discovery token. Decapod uses automatic server discovery and cloud-init is required only for that. To access the Decapod API, servers will access it using an authentication token with limited capabilities (posting to the server discovery API endpoint). The server discovery token is set in the api.server_discovery_token section of the config.yaml file. Keep this token private. To obtain the token:

   ```
   $ grep server_discovery_token config.yaml
     server_discovery_token: "7f080dab-d803-4339-9c69-e647f7d6e200"
   ```
3. Generate an SSH public key. To generate the SSH public key from a private one, run:

   ```
   $ ssh-keygen -y -f ansible_ssh_keyfile.pem > ansible_ssh_keyfile.pem.pub
   ```

   Note

   The ansible_ssh_keyfile.pem file should have the 0600 permissions:

   ```
   $ chmod 0600 ansible_ssh_keyfile.pem
   ```

Generate user data

Verify that you have completed the steps described in Prerequisites.

To generate user data:

Run the following command:

```
$ decapod -u http://10.10.0.2:9999 cloud-config \
  7f080dab-d803-4339-9c69-e647f7d6e200 ansible_ssh_keyfile.pem.pub
```

Where the URL is the public URL of the Decapod machine with a correct port. The servers will send an HTTP request for server discovery using this URL. As a result, you will obtain a YAML-like user data.

Deploy OS using MAAS

Decapod does not provide MAAS deployment. This section describes one of the Ceph node OS deployment options that you may consider. To provision your Ceph nodes manually, skip this section.

Prerequisites

MAAS installation has the following requirements:

- MAAS has its own DHCP server. To avoid collisions, disable the default one.
- If you plan to run MAAS in a virtual network with libvirt, create a new network with disabled DHCP, but enabled NAT.

Install MAAS

To install MAAS:

To install MAAS, follow the steps described in:

1. Installing a single node MAAS.
2. Importing the boot images.
3. Logging in.

Deploy an OS using MAAS

To deploy an operating system using MAAS:

1. Encode the user data to base64 and send it to MAAS:

   ```
   $ decapod -u http://10.10.0.2:9999 cloud-config \
     7f080dab-d803-4339-9c69-e647f7d6e200 ansible_ssh_keyfile.pem.pub \
     | base64 -w 0 > user_data.txt
   ```

2. Deploy an OS using the required MAAS version.

   Note

   MAAS 2.0 has non-backward-compatible API changes.

   - MAAS 2.0:

     1. Obtain system_id of the machine to deploy:

        ```
        $ maas mymaas nodes read
        ```

     2. Deploy the OS:

```
$ maas mymaas machine deploy {system_id} user_data={base64-encoded of user-data}
```

Where mymaas is the profile name of the MAAS command line.

- MAAS prior to 2.0:

  1. Obtain system_id of the machine to deploy:

     ```
     $ maas prof nodes list
     ```

  2. Deploy the OS:

     ```
     $ maas mymaas node start {system_id} user_data={base64-encoded of \
     user-data} distro_series={distro series. Eg. trusty}
     ```

     Where mymaas is the profile name of the MAAS command line.

> Note
>
> If you do not want or cannot use server discovery, refer to Ansible playbooks to prepare a machine based on generated user data.

Manage users and roles

This section descibes how to manage users and roles in Decapod through the web UI.

Manage users

To add a new user:

1. Log in to the Decapod web UI.
2. Navigate to USERS MANAGEMENT.
3. Click the USERS tab.
4. Click CREATE NEW USER and type the required data.
5. Click SAVE. A new user has been created.

> Note
>
> The password is sent to the user email. This password can be changed.

After saving the changes, you will see that the CHANGELOG is updated. This CHANGELOG tracks all the results and it is possible to view the details about a user modifications. This is related not only to the user management. Decapod stores all changes and you can always obtain the entire log.

Clicking DELETE USER does not delete the user but archives it instead. You can still access the user through the Decapod CLI if you know the user ID.

Manage roles

Using the Decapod web UI you can create, edit, and delete roles as well as assign a role to a user.

To create a new role:

1. In the Decapod web UI. Navigate to USERS MANAGEMENT.
2. Click the ROLES tab.
3. Click CREATE NEW ROLE.
4. Type the role name and select the required permissions.
5. Click SAVE CHANGES.

To edit a role:

1. In the Decapod web UI, navigate to USERS MANAGEMENT.
2. Click the ROLES tab.
3. Click the pen icon near the required role name and edit the role as required.

To delete a role:

1. In the Decapod web UI, navigate to USERS MANAGEMENT.
2. Click the ROLES tab.
3. Click the trash can icon near the required role name.

> Note
>
> This will not completely delete the role but will archive it instead. You can access the role through the Decapod CLI if you know the role ID.

To assign a role to a user:

1. In the Decapod web UI, navigate to USERS MANAGEMENT.
2. Click the USERS tab.
3. Expand the required user.
4. Select the required role in the ROLE section.
5. Click SAVE.

Seealso

Ceph cluster deployed by Decapod in MCP Reference Architecture

Deploy a cluster

This section describes the cluster deployment workflow using the Decapod web UI.

Create a cluster

To create a cluster:

1. Log in to the Decapod web UI.
2. Navigate to CLUSTER.
3. Click CREATE NEW CLUSTER.
4. Type the cluster name and click SAVE.

A new cluster is empty and contains no servers. Discover servers as described in Discover a server.

View servers

Verify that you have discovered the required servers as described in Discover a server.

To view the discovered servers:

1. Log in to the Decapod web UI.
2. Navigate to SERVERS. The SERVERS page lists the servers accessible by Decapod.
3. Expand the required server to view its details.

Create a playbook configuration

To create a playbook configuration:

1. Log in to the Decapod web UI.
2. Navigate to PLAYBOOK CONFIGURATION.
3. Click CREATE NEW CONFIGURATION.
4. Type the configuration name and select a cluster, then click NEXT.
5. Select the required playbook and click NEXT.

   The table lists the plugins available for execution. Some playbooks require an explicit list of servers. For example, to purge a cluster, Decapod will use the servers in this cluster and you do not need to specify them manually.
6. In the SELECT SERVERS window, select all servers and click SAVE CHANGES. Once the new playbook configuration is created, you will see the PLAYBOOK CONFIGURATION window.

7. Edit the playbook configuration, if required, and click SAVE CHANGES.

Execute a playbook configuration

To execute a playbook configuration:

1. In the Decapod web UI, navigate to CONFIGURATIONS.
2. Click EXECUTE to execute the required configuration. Once the execution starts, its state changes to STARTED on the EXECUTIONS page.
3. To view the execution process, click LOGS.
4. Once the execution is finished, its status will change to COMPLETED. To download the entire execution log, click DOWNLOAD.

Playbook plugins

Decapod uses plugins to manage Ceph. These plugins support various tasks, such as cluster deployment, adding and removing of OSDs, and others. This section describes the available playbook plugins and the main options these plugins support.

Deploy Ceph cluster

The Deploy Ceph cluster playbook plugin allows you to deploy an initial Ceph cluster. The plugin supports all the capabilities and roles of ceph-ansible.

Note

The majority of configuration options described in this section match the ceph-ansible settings. For a list of supported parameters, see official list.

Overview

The following table shows the general information about the Deploy Ceph cluster plugin:

Property	Value
ID	cluster_deploy
Name	Deploy Ceph Cluster
Required Server List	Yes

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
dmcrypt	Use dmcrypted OSDs	False	Defines the dmcrypt usage for OSD devices.

collocation	Collocate OSD data and journal on same devices	False	Defines whether the journal and data have to be placed on the same devices.
rest_api	Setup Ceph RestAPI	False	Defines the RestAPI installation for Ceph.
mon_count	The number of monitors to deploy	3	Defines the number of monitors.

The Deploy Ceph cluster plugin is tightly coupled with ceph-ansible versions. The following table shows the mapping between the plugin version and the corresponding version of ceph-ansible.

Plugin version	version
>=0.1,<0.2	v1.0.8
>=0.2,<0.3	v2.1.9

Parameters and roles

The Deploy Ceph cluster plugin has the following parameters:

ceph_facts_template

The path to custom Ceph facts template. Decapod deploys the custom facts module on the nodes that collect the Ceph-related facts. Usually, you do not need to configure this parameter.

ceph_stable

Set to true if it is required to install Ceph from the stable repository.

ceph_stable_repo**,** ceph_stable_release**,** ceph_stable_distro_source

The options define the repository where to obtain Ceph. In case of Ubuntu Xenial, you will get the following repository string:

```
deb {{ ceph_stable_repo }} {{ ceph_stable_distro_source }} main
```

cluster

Defines the cluster name.

Important

Some tools require the ceph cluster name only. The default name allows executing the ceph utility without an explicit cluster name and with the --cluster option.

cluster_network

Defines the cluster network.

copy_admin_key

Copies the admin key to all nodes. This is required if you want to run the ceph utility from any cluster node. Keep this option as true. Otherwise, it may break some playbooks that maintain the lifecycle after deployment.

fsid

The unique identifier for your object store. Since you can run multiple clusters on the same hardware, you must specify the unique ID of the object store when bootstrapping a monitor.

journal_collocation

Defines if the OSD will place its journal on the same disk with the data. It is false by default.

If you want to have separate disks for journals (SSDs) and data (rotationals), set this to false. Also, set raw_multi_journal to true and list journal disks as raw_journal_devices.

raw_multi_journal

This option is the opposite to journal_collocation.

Note

The raw_multi_journal and journal_collocation options must have different values. For example, if journal_collocation is set to true, set raw_multi_journal to false.

dmcrypt_journal_collocation

This option has the same meaning as journal_collocation but both journal and data disks are encrypted by dmcrypt.

dmcrypt_dedicated_journal

This option has the same meaning as journal_collocation set to false. If dmcrypt_dedicated_journal is set to true, the journal and data will be placed on different disks and encrypted with dmcrypt.

journal_size

OSD journal size in megabytes.

max_open_files

Sets the number of open files to have on a node.

nfs_file_gw

Set to true to enable file access through NFS. Requires an MDS role.

nfs_obj_gw

Set to true to enable object access through NFS. Requires an RGW role.

os_tuning_params

Different kernels parameters. This is the list of dicts where name is the name of the parameter and value is the value.

public_network

Defines the public network.

monitor_interface

The option defines the NIC on the host that is connected to the public network.

devices
: Defines the disks where to place the OSD data. If collocation is enabled, then journal devices, raw_journal_devices, are not used.

raw_journal_devices
: Defines the disks where to place the journals for OSDs. If collocation is enabled, this option is not used.

The ceph-ansible project supports two deployment modes of a Ceph cluster: with journal collocation and on separate drives, and also with dmcrypt and without it. Therefore, there are four possible combinations.

The following table lists the possible combinations:

Setting	Combination 1	Combination 2	Combination 3	Combination 4
collocation	true	true	false	false
dmcrypt	true	false	true	false
journal_collocation	false	true	false	false
raw_multi_journal	true	false	false	true
dmcrypt_journal_collocation	false	false	false	false
dmcrypt_dedicated_journal	false	false	true	false
Data devices option name	devices	devices	devices	devices
Journal devices option name	--	--	raw_journal_devices	raw_journal_devices

Consider the different meaning of devices and raw_journal_devices in different modes: if no collocation is defined, then devices means disks with data. Journals are placed on raw_journal_devices disks. Otherwise, define devices only. In this case, the journal will be placed on the same device as the data.

Configuration example

The following is an example of the Deploy Ceph cluster plugin configuration:

```
{
 "global_vars": {
   "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/\
   decapod_common/facts/ceph_facts_module.py.j2",
   "ceph_stable": true,
   "ceph_stable_distro_source": "jewel-xenial",
   "ceph_stable_release": "jewel",
   "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
   "cluster": "ceph",
   "cluster_network": "10.10.0.0/24",
   "copy_admin_key": true,
   "dmcrypt_dedicated_journal": true,
   "dmcrypt_journal_collocation": false,
   "fsid": "e0b82a0d-b669-4787-8f4d-84f6733e45cd",
   "journal_collocation": false,
   "journal_size": 512,
   "max_open_files": 131072,
   "nfs_file_gw": false,
   "nfs_obj_gw": false,
   "os_tuning_params": [
     {
       "name": "kernel.pid_max",
       "value": 4194303
     },
     {
       "name": "fs.file-max",
       "value": 26234859
     }
   ],
   "public_network": "10.10.0.0/24",
   "raw_multi_journal": false
 },
 "inventory": {
   "_meta": {
     "hostvars": {
       "10.10.0.10": {
         "ansible_user": "ansible",
         "devices": [
           "/dev/vde",
           "/dev/vdb"
         ],
         "monitor_interface": "ens3",
         "raw_journal_devices": [
           "/dev/vdd",
           "/dev/vdc"
         ]
       },
       "10.10.0.11": {
         "ansible_user": "ansible",
         "devices": [
           "/dev/vde",
           "/dev/vdb"
         ],
         "monitor_interface": "ens3",
         "raw_journal_devices": [
           "/dev/vdd",
           "/dev/vdc"
         ]
       },
       "10.10.0.12": {
         "ansible_user": "ansible",
         "devices": [
           "/dev/vde",
           "/dev/vdb"
         ],
         "monitor_interface": "ens3",
         "raw_journal_devices": [
           "/dev/vdd",
           "/dev/vdc"
         ]
       },
       "10.10.0.8": {
         "ansible_user": "ansible",
         "devices": [
           "/dev/vde",
           "/dev/vdb"
         ],
         "monitor_interface": "ens3",
         "raw_journal_devices": [
           "/dev/vdd",
           "/dev/vdc"
         ]
       },
       "10.10.0.9": {
         "ansible_user": "ansible",
         "devices": [
           "/dev/vde",
           "/dev/vdb"
         ],
         "monitor_interface": "ens3",
         "raw_journal_devices": [
           "/dev/vdd",
           "/dev/vdc"
         ]
       }
     }
   },
   "clients": [],
   "iscsi_gw": [],
   "mdss": [],
   "mons": [
     "10.10.0.9"
   ],
   "nfss": [],
   "osds": [
     "10.10.0.10",
     "10.10.0.12",
     "10.10.0.11",
     "10.10.0.8"
   ],
   "rbdmirrors": [],
   "restapis": [
     "10.10.0.9"
   ],
   "rgws": []
  }
 }
```

Add OSD host

The Add OSD host playbook plugin allows you to add a new host with OSDs to a cluster. The plugin supports all the capabilities and roles of ceph-ansible.

Note

The majority of configuration options described in this section match the ceph-ansible settings. For a list of supported parameters, see official list.

Overview

The following table shows the general information about the Add OSD host plugin:

Property	Value
ID	add_osd
Name	Add OSD Host
Required server list	Yes

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
dmcrypt	Use dmcrypted OSDs	False	Defines the dmcrypt usage for OSD devices.
collocation	Collocate OSD data and journal on same devices	False	Defines whether the journal and data will be placed on the same devices.

The Add OSD host plugin is tightly coupled with ceph-ansible versions. The following table shows the mapping between the plugin version and the corresponding version of ceph-ansible.

Plugin version	version
>=0.1,<0.2	v1.0.8
>=0.2,<0.3	v2.1.9

Parameters and roles

The Add OSD host plugin parameters are mostly the same as the ones for the Deploy Ceph cluster plugin. However, the plugin has the following roles:

mons

Defines the nodes to deploy monitors.

osds

Defines the nodes to deploy OSDs.

Note

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

ceph_version_verify

A boolean setting that checks that strict mode is enabled. If set to false, no verification described above is performed.

ceph_version_verify_packagename

The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the Add OSD host plugin configuration:

```
{
  "data": {
    "cluster_id": "1597a71f-6619-4db6-9cda-a153f4f19097",
    "configuration": {
      "global_vars": {
        "ceph_facts_template": "/usr/local/lib/python3.5/\
        dist-packages/shrimp_common/facts/ceph_facts_module.py.j2",
        "ceph_stable": true,
        "ceph_stable_distro_source": "jewel-xenial",
        "ceph_stable_release": "jewel",
        "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
        "cluster": "ceph",
        "cluster_network": "10.10.0.0/24",
        "copy_admin_key": true,
        "fsid": "1597a71f-6619-4db6-9cda-a153f4f19097",
        "journal_collocation": true,
        "journal_size": 100,
        "max_open_files": 131072,
        "nfs_file_gw": false,
        "nfs_obj_gw": false,
        "os_tuning_params": [
            {
                "name": "kernel.pid_max",
                "value": 4194303
            },
            {
                "name": "fs.file-max",
                "value": 26234859
            }
        ],
        "public_network": "10.10.0.0/24"
      },
      "inventory": {
        "_meta": {
            "hostvars": {
                "10.10.0.2": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.3": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                }
            }
        },
        "mons": [
            "10.10.0.2"
        ],
        "osds": [
            "10.10.0.3",
        ],
      }
    },
    "name": "add_osd_name",
    "playbook_id": "add_osd"
```

```
    },
    "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1478174220,
    "version": 2
}
```

Remove OSD host

The Remove OSD host playbook plugin allows you to remove a host with OSDs from a cluster.

Overview

The following table shows the general information about the Remove OSD host plugin:

Property	Value
ID	remove_osd
Name	Remove OSD Host
Required Server List	Yes

Configuration example

The following is an example of the Remove OSD host plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.10.0.12": {
          "ansible_user": "ansible"
        },
        "10.10.0.9": {
          "ansible_user": "ansible"
        }
      }
    },
    "mons": [
      "10.10.0.9"
    ],
    "osds": [
      "10.10.0.12"
    ]
  }
}
```

This playbook has the simplest possible configuration. You only need to define the monitors and the OSD to remove.

Add monitor host

The Add monitor host playbook plugin allows you to add a new host with monitors to a cluster. The plugin supports all the capabilities and roles of ceph-ansible.

> Note
>
> The majority of configuration options described in this section match the ceph-ansible settings. For a list of supported parameters, see official list.

Overview

The following table shows the general information about the Add monitor host plugin:

Property	Value
ID	add_mon
Name	Add Monitor Host
Required Server List	Yes

The Add monitor host plugin is tightly coupled with ceph-ansible versions. The following table shows the mapping between the plugin version and the corresponding version of ceph-ansible.

Plugin version	version
>=0.2,<0.3	v2.1.9

Parameters and roles

The Add monitor host plugin parameters are mostly the same as the ones for the Deploy Ceph cluster plugin. However, the plugin has the following role:

mons
: Defines the nodes to deploy monitors.

> Note
>
> For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the

cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

ceph_version_verify

: A boolean setting that checks that strict mode is enabled. If set to false, no verification described above is performed.

ceph_version_verify_packagename

: The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the Add monitor host plugin configuration:

```
{
 "global_vars": {
  "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/\
  decapod_common/facts/ceph_facts_module.py.j2",
  "ceph_stable": true,
  "ceph_stable_distro_source": "jewel-xenial",
  "ceph_stable_release": "jewel",
  "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
  "cluster": "ceph",
  "cluster_network": "10.10.0.0/24",
  "copy_admin_key": true,
  "fsid": "d5069dc9-05d9-4ef2-bc21-04a938917260",
  "max_open_files": 131072,
  "nfs_file_gw": false,
  "nfs_obj_gw": false,
  "os_tuning_params": [
   {
    "name": "fs.file-max",
    "value": 26234859
   },
   {
    "name": "kernel.pid_max",
    "value": 4194303
   }
  ],
  "public_network": "10.10.0.0/24"
 },
 "inventory": {
  "_meta": {
   "hostvars": {
    "10.10.0.10": {
     "ansible_user": "ansible",
          "monitor_interface": "ens3"
        },
        "10.10.0.12": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        },
        "10.10.0.8": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        },
        "10.10.0.9": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        }
      }
    },
    "mons": [
      "10.10.0.10",
      "10.10.0.12",
      "10.10.0.8",
      "10.10.0.9"
    ]
  }
}
```

Remove monitor host

The Remove monitor host playbook plugin allows you to remove a host with monitor from a cluster.

Overview

The following table shows general information about the Remove monitor host plugin:

Property	Value
ID	remove_mon
Name	Remove monitor host
Required server list	Yes

Note

You must have enough monitor hosts to make PAXOS quorum.

Configuration example

The following is an example of the Remove monitor host plugin configuration:

```
{
 "global_vars": {
   "cluster": "ceph"
 },
 "inventory": {
   "_meta": {
     "hostvars": {
       "10.10.0.12": {
         "ansible_user": "ansible"
       },
       "10.10.0.9": {
         "ansible_user": "ansible"
       }
     }
   },
   "mons": [
     "10.10.0.9",
     "10.10.0.12"
   ]
 }
}
```

This playbook has the simplest possible configuration. You only need to define the monitors you want to remove.

Purge cluster

The Purge cluster playbook plugin allows you to remove a host with OSDs from a cluster.

Overview

The following table shows the general information about the Purge cluster plugin:

Property	Value
ID	purge_cluster
Name	Purge Cluster
Required Server List	No

Parameters and roles

The Purge cluster plugin has the following parameter:

cluster
: Defines the name of the cluster.

Important

Some tools require the ceph cluster name only. The default name allows executing the ceph utility without an explicit cluster name and with the --cluster option.

Configuration example

The following is an example of the Purge cluster plugin configuration:

```
{
 "global_vars": {
   "cluster": "ceph"
 },
 "inventory": {
   "_meta": {
     "hostvars": {
       "10.10.0.10": {
         "ansible_user": "ansible"
       },
       "10.10.0.11": {
         "ansible_user": "ansible"
       },
       "10.10.0.12": {
         "ansible_user": "ansible"
       },
       "10.10.0.8": {
         "ansible_user": "ansible"
       },
       "10.10.0.9": {
         "ansible_user": "ansible"
       }
     }
   },
   "mons": [
     "10.10.0.9"
   ],
   "osds": [
     "10.10.0.10",
     "10.10.0.12",
     "10.10.0.11",
     "10.10.0.8"
   ],
   "restapis": [
     "10.10.0.9"
   ]
```

```
  }
}
```

This playbook has the simplest possible configuration. You only need to define the nodes and their roles.

Telegraf integration

The Telegraf integration playbook plugin activates the Ceph metrics in Telegraf. These metrics can be sent to Prometheus, InfluxDB, or any other endpoint.

Overview

The following table shows general information about the Telegraf integration plugin:

Property	Value
ID	telegraf_integration
Name	Telegraf Integration
Required Server List	Yes

The plugin uses a standalone Ansible role from Ansible Galaxy. The following table shows the versions mapping:

Plugin version	Ansible Galaxy version
>=0.2,<0.3	dj-wasabi.telegraf 0.7.0

Configuration example

The following is an example of the Telegraf integration plugin configuration:

```
{
 "global_vars": {
   "ceph_binary": "/usr/bin/ceph",
   "ceph_config": "/etc/ceph/ceph.conf",
   "ceph_user": "client.admin",
   "configpath": "/etc/telegraf/telegraf.conf",
   "gather_admin_socket_stats": true,
   "gather_cluster_stats": true,
   "install": true,
   "interval": "1m",
   "mon_prefix": "ceph-mon",
   "osd_prefix": "ceph-osd",
   "socket_dir": "/var/run/ceph",
   "socket_suffix": "asock",
   "telegraf_agent_collection_jitter": 0,
   "telegraf_agent_deb_url": "https://dl.influxdata.com/telegraf/releases/telegraf_1.1.2_amd64.deb",
   "telegraf_agent_debug": false,
    "telegraf_agent_flush_interval": 10,
    "telegraf_agent_flush_jitter": 0,
    "telegraf_agent_interval": 10,
    "telegraf_agent_logfile": "",
    "telegraf_agent_metric_batch_size": 1000,
    "telegraf_agent_metric_buffer_limit": 10000,
    "telegraf_agent_omit_hostname": false,
    "telegraf_agent_output": [
     {
      "config": [
        "urls = [\"http://localhost:8086\"]",
        "database = \"telegraf\"",
        "precision = \"s\""
      ],
      "type": "influxdb"
     }
    ],
    "telegraf_agent_quiet": false,
    "telegraf_agent_round_interval": true,
    "telegraf_agent_tags": {},
    "telegraf_agent_version": "1.1.2",
    "telegraf_plugins_default": [
     {
      "config": [
        "percpu = true"
      ],
      "plugin": "cpu"
     },
     {
      "plugin": "disk"
     },
     {
      "plugin": "io"
     },
     {
      "plugin": "mem"
     },
     {
      "plugin": "net"
     },
     {
      "plugin": "system"
     },
     {
      "plugin": "swap"
     },
     {
      "plugin": "netstat"
     }
    ],
    "telegraf_plugins_extra": {}
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible"
        }
      }
    },
    "telegraf": [
      "10.0.0.20"
    ]
  }
}
```

Seealso

- Ceph storage input plugin
- Telegraf source for Ceph storage

Telegraf removal

While the Telegraf integration plugin installs and configures Telegraf, the Telegraf removal plugin uninstalls Telegraf or its managed section from the configuration.

Overview

The following table shows general information about the Telegraf removal plugin:

Property	Value
ID	purge_telegraf
Name	Telegraf removal
Required Server List	Yes

The following hints are available for the plugin:

remove_config_section_only

If set to true, the plugin will remove the corresponding section created by Telegraf integration plugin from /etc/telegraf/telegraf.conf.

Configuration example

The following is an example of the Telegraf removal plugin configuration:

```
{
 "global_vars": {
   "configpath": "/etc/telegraf/telegraf.conf",
   "remove_config_section_only": false
 },
 "inventory": {
   "_meta": {
     "hostvars": {
       "10.0.0.20": {
         "ansible_user": "ansible"
       }
     }
   },
   "telegraf": [
     "10.0.0.20"
   ]
 }
}
```

Cinder integration

The Cinder integration plugin allows you to perform an integration between a deployed Ceph cluster and the OpenStack Block storage service (Cinder).

Overview

The following table shows general information about the Cinder integration plugin:

Property	Value
ID	cinder_integration
Name	Cinder Integration
Required Server List	No

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
cinder	Use Cinder with Ceph back end	True	Defines if Cinder will be used with Ceph back end. This is required to create a volumes pool by default.

Cinder requires keyrings and the contents of the Ceph configuration file, for example, ceph.conf. This plugin creates required keyrings in Ceph, creates required pools, and allows Decapod to return required files.

To integrate Cinder:

1. Run the plugin through the Decapod Web UI.

2. Obtain the required files:

   ```
   $ decapod cluster cinder-integration a2b813b2-df23-462b-8dab-6a80f9bc7fce
   ```

   Where a2b813b2-df23-462b-8dab-6a80f9bc7fce is the cluster ID. This command will return the contents of required files.

   To obtain the files and store in the file system, use the --store option:

   ```
   $ decapod cluster cinder-integration --store 8b205db5-3d29-4f1b-82a5-e5cefb522d4f
   ```

   This command will output the contents of the files and store them in the file system.

Parameters and roles

The Cinder integration plugin has the following parameters:

cluster
: Name of the cluster to use.

clients
: A mapping of client to create in Ceph for permissions.

pools
: A mapping of pool name to PG count that should be used.

Configuration example

The following is an example of the Cinder integration plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "clients": {
            "compute": {
              "mon": "allow r",
              "osd": "allow class-read object_prefix rbd_children, allow \
              rwx pool=compute, allow rwx pool=volumes, allow rx pool=images"
            },
            "images": {
              "mon": "allow r",
              "osd": "allow class-read object_prefix rbd_children, allow \
              rwx pool=images"
            },
```

```
          "volumes": {
            "mon": "allow r",
            "osd": "allow class-read object_prefix rbd_children, allow \
            rwx pool=volumes, allow rx pool=images"
          }
        },
        "pools": {
          "compute": 64,
          "images": 64,
          "volumes": 64
        }
      }
    }
  },
  "mons": [
    "10.0.0.20"
  ]
 }
}
```

Use the Decapod CLI

Install the Decapod CLI

To install the Decapod CLI on a local machine, install two packages:

- decapodlib, the RPC client library to access the Decapod API
- decapod-cli, the CLI wrapper for the library

To install the Decapod CLI:

1. At the top level of the source code repository, run the following command to build the packages and place them to the output/eggs directory:

   ```
   $ make build_eggs
   ```

2. Install the packages:

   ```
   $ pip install output/eggs/decapodlib*.whl output/eggs/decapod_cli*.whl
   ```

3. Run decapod to verify the installation.

Seealso

- Access the Decapod CLI

Access the Decapod CLI

To access Decapod, you need to know its URL (http://10.10.0.2:9999 or https://10.10.0.2:10000), your username and password (root/root for a development installation).

To access Decapod using CLI:

1. Set your credentials directly to the Decapod CLI or use the environment variables:

   ```
   export DECAPOD_URL=http://10.10.0.2:9999
   export DECAPOD_LOGIN=root
   export DECAPOD_PASSWORD=root
   ```

   Save this to a file and source when required.

2. Verify that it works:

   ```
   $ decapod -u http://10.10.0.2:9999 -l root -p root user get-all
   ```

   If you used environment variables, run:

   ```
   $ decapod user get-all
   ```

Cluster deployment workflow

This section describes the cluster deployment workflow and includes the following topics:

Create a cluster

To create a cluster:

1. Verify that you can log in to the Decapod using CLI.
2. To create a cluster, run:

   ```
   $ decapod cluster create <CUSTER_NAME>
   ```

   Example:

   ```
   $ decapod cluster create ceph
   {
       "data": {
           "configuration": {},
           "name": "ceph"
       },
       "id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
       "initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
       "model": "cluster",
       "time_deleted": 0,
       "time_updated": 1479902503,
   ```

```
    "version": 1
}
```

As a result, a new cluster with the name ceph and ID f2621e71-76a3-4e1a-8b11-fa4ffa4a6958 has been created. This ID is required for creating the playbook configuration.

3. Proceed to Discover a server.

Discover a server

To discover a server:

1. Generate the user-data configuration for cloud-init. For details, see Generate user data.

   The cloud-init execution generates the content of /etc/rc.local. The first and next reboots will call the Decapod API for server registering. Such registration is an idempotent operation. The execution of the Decapod API (POST /v1/server) creates a task for the controller server on facts discovery. The controller executes this task and collects facts from the remote host. A new server model is created or the information on the existing one is updated.

2. With this configuration, deploy an operating system on a Ceph node. For an example of such deployment, see: Deploy an OS on a Ceph node, official cloud-init documentation, or use kernel parameters.

As a result, the server should be listed in Decapod. The server discovery takes time because of cloud-init. Therefore, the server may appear in five minutes after deployment. Once the server appears in Decapod, the tool can use it.

Seealso

Ceph cluster deployed by Decapod in MCP Reference Architecture

Create a playbook configuration

To create a playbook configuration:

1. List the existing playbooks:

```
$ decapod playbook get-all
{
    "items": [
        {
            "description": "Adding new OSD to the cluster.\n\nThis \
            plugin adds OSD to the existing cluster.",
            "id": "add_osd",
            "name": "Add OSD to Ceph cluster",
            "required_server_list": true
        },
        {
            "description": "Ceph cluster deployment playbook.\n\nThis \
            plugin deploys Ceph cluster into a set of servers. After \
            sucessfull deployment, cluster model will be updated.",
            "id": "cluster_deploy",
            "name": "Deploy Ceph cluster",
            "required_server_list": true
        },
        {
            "description": "Example plugin for playbook.\n\nThis plugin \
            deploys simple hello world service on remote machine \
            If\nremote machine host is 'hostname', \
            then http://hostname:8085 will\nrespond with \
            '{\"result\": \"ok\"}' JSON.",
            "id": "hello_world",
            "name": "Hello World",
            "required_server_list": false
        },
        {
            "description": "Purge whole Ceph cluster.\n\nThis plugin \
            purges whole Ceph cluster. It removes packages, all data,\
            \nreformat Ceph devices.",
            "id": "purge_cluster",
            "name": "Purge cluster",
            "required_server_list": false
        },
        {
            "description": "Remove OSD host from cluster.",
            "id": "remove_osd",
            "name": "Remove OSD host from Ceph cluster",
            "required_server_list": true
        }
    ]
}
```

This will list the available playbooks in details. The name and description are the human-readable items to display in the Decapod UI.

2. Note the ID of the Ceph cluster deployment playbook. It is cluster_deploy in the example above.
3. The cluster deployment playbook requires a list of servers to operate with (field required_server_list is true). To list the available servers:

```
$ decapod server get-all
```

Note

The output of this command can be quite long. Therefore, we recommend that you use a tool for listing. One of the best tools available to work with JSON in CLI is jq.

4. Obtain the required server IDs:

   - Extract the IDs manually
   - Use compact listing:

   ```
   $ decapod server get-all --compact
   "machine_id","version","fqdn","username","default_ip",\
   "interface=mac=ipv4=ipv6","..."
   "015fd324-4437-4f28-9f4b-7e3a90bdc30f","1","chief-gull.maas","ansible",\
   "10.10.0.9","ens3=52:54:00:29:14:22=10.10.0.9=fe80::5054:ff:fe29:1422"
   "7e791f07-845e-4d70-bff1-c6fad6bfd7b3","1","exotic-swift.maas","ansible",\
   "10.10.0.11","ens3=52:54:00:05:b0:54=10.10.0.11=fe80::5054:ff:fe05:b054"
   "70753205-3e0e-499d-b019-bd6294cfbe0f","1","helped-pig.maas","ansible",\
   "10.10.0.12","ens3=52:54:00:01:7c:1e=10.10.0.12=fe80::5054:ff:fe01:7c1e"
   "40b96868-205e-48a2-b8f6-3e3fcfbc41ef","1","joint-feline.maas","ansible",\
   "10.10.0.10","ens3=52:54:00:4a:c3:6d=10.10.0.10=fe80::5054:ff:fe4a:c36d"
   "8dd33842-fee6-4ec7-a1e5-54bf6ae24710","1","polite-rat.maas","ansible",\
   "10.10.0.8","ens3=52:54:00:d4:da:29=10.10.0.8=fe80::5054:ff:fed4:da29"
   ```

   Where machine_id is the server ID.

   - Use the jq tool mentioned above:

   ```
   $ decapod server get-all | jq -rc '.[]|.id'
   015fd324-4437-4f28-9f4b-7e3a90bdc30f
   7e791f07-845e-4d70-bff1-c6fad6bfd7b3
   70753205-3e0e-499d-b019-bd6294cfbe0f
   40b96868-205e-48a2-b8f6-3e3fcfbc41ef
   8dd33842-fee6-4ec7-a1e5-54bf6ae24710
   ```

Note

We recommend using the jq tool as the compact representation shows only a limited amount of information. Using jq allows you to extract any certain data.

5. At this step you should have all the required data to create a playbook configuration:

   - The cluster name (can be any)
   - The playbook name

- The cluster ID
- The server IDs

6. Create a playbook configuration using the following command:

```
$ decapod playbook-configuration create <NAME> <PLAYBOOK> <CLUSTER_ID> [SERVER_IDS]...
```

Example:

```
$ decapod playbook-configuration create deploy cluster_deploy \
f2621e71-76a3-4e1a-8b11-fa4ffa4a6958 015fd324-4437-4f28-9f4b-7e3a90bdc30f \
7e791f07-845e-4d70-bff1-c6fad6bfd7b3 70753205-3e0e-499d-b019-bd6294cfbe0f \
40b96868-205e-48a2-b8f6-3e3fcfbc41ef 8dd33842-fee6-4ec7-a1e5-54bf6ae24710
{
   "data": {
      "cluster_id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
      "configuration": {
         "global_vars": {
            "ceph_facts_template": "/usr/local/lib/python3.5/\
            dist-packages/decapod_common/facts/ceph_facts_module.py.j2",
            "ceph_stable": true,
            "ceph_stable_distro_source": "jewel-xenial",
            "ceph_stable_release": "jewel",
            "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
            "cluster": "ceph",
            "cluster_network": "10.10.0.0/24",
            "copy_admin_key": true,
            "fsid": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
            "journal_collocation": true,
            "journal_size": 100,
            "max_open_files": 131072,
            "nfs_file_gw": false,
            "nfs_obj_gw": false,
            "os_tuning_params": [
               {
                  "name": "fs.file-max",
                  "value": 26234859
               },
               {
                  "name": "kernel.pid_max",
                  "value": 4194303
               }
            ],
            "public_network": "10.10.0.0/24"
         },
         "inventory": {
            "_meta": {
               "hostvars": {
                  "10.10.0.10": {
```

```
            "ansible_user": "ansible",
            "devices": [
                "/dev/vdc",
                "/dev/vde",
                "/dev/vdd",
                "/dev/vdb"
            ],
            "monitor_interface": "ens3"
        },
        "10.10.0.11": {
            "ansible_user": "ansible",
            "devices": [
                "/dev/vdc",
                "/dev/vde",
                "/dev/vdd",
                "/dev/vdb"
            ],
            "monitor_interface": "ens3"
        },
        "10.10.0.12": {
            "ansible_user": "ansible",
            "devices": [
                "/dev/vdc",
                "/dev/vde",
                "/dev/vdd",
                "/dev/vdb"
            ],
            "monitor_interface": "ens3"
        },
        "10.10.0.8": {
            "ansible_user": "ansible",
            "devices": [
                "/dev/vdc",
                "/dev/vde",
                "/dev/vdd",
                "/dev/vdb"
            ],
            "monitor_interface": "ens3"
        },
        "10.10.0.9": {
            "ansible_user": "ansible",
            "devices": [
                "/dev/vdc",
                "/dev/vde",
                "/dev/vdd",
                "/dev/vdb"
            ],
            "monitor_interface": "ens3"
```

```
                    }
                }
            },
            "clients": [],
            "iscsi_gw": [],
            "mdss": [],
            "mons": [
                "10.10.0.9"
            ],
            "nfss": [],
            "osds": [
                "10.10.0.10",
                "10.10.0.12",
                "10.10.0.11",
                "10.10.0.8"
            ],
            "rbdmirrors": [],
            "restapis": [
                "10.10.0.9"
            ],
            "rgws": []
        }
    },
    "name": "deploy",
    "playbook_id": "cluster_deploy"
},
"id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
"initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
"model": "playbook_configuration",
"time_deleted": 0,
"time_updated": 1479906402,
"version": 1
}
```

Where the playbook configuration ID is fd499a1e-866e-4808-9b89-5f582c6bd29e.

Update a playbook configuration

You may need to update a playbook configuration, for example, to use another host for the monitor.

To do so, update the playbook model using one of the following ways:

- Edit the playbook and send to stdin of the decapod playbook-configuration update fd499a1e-866e-4808-9b89-5f582c6bd29e command where fd499a1e-866e-4808-9b89-5f582c6bd29e is the playbook configuration ID.
- Run an external editor with the --model-editor option. Using this option, the Decapod CLI downloads the model and sends its data field to the editor. After you save and close the

editor, the updated model is sent to the Decapod API. To use this model, verify that your editor is set using the env | grep EDITOR command.

- Dump JSON with modifications and inject into the --model option.

> **Important**
>
> Avoid updating fields outside of the data field (that is why the --model-editor option shows only the data field). Sending the whole model back to the Decapod API allows keeping consistent behavior of the Decapod API.

To update a playbook configuration:

1. Run the decapod playbook-configuration update command with the --model-editor flag.

   Example:

```
$ decapod playbook-configuration update fd499a1e-866e-4808-9b89-5f582c6bd29e --model-editor
{
   "data": {
      "cluster_id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
      "configuration": {
         "global_vars": {
            "ceph_facts_template": "/usr/local/lib/python3.5/\
            dist-packages/decapod_common/facts/ceph_facts_module.py.j2",
            "ceph_stable": true,
            "ceph_stable_distro_source": "jewel-xenial",
            "ceph_stable_release": "jewel",
            "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
            "cluster": "ceph",
            "cluster_network": "10.10.0.0/24",
            "copy_admin_key": true,
            "fsid": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
            "journal_collocation": true,
            "journal_size": 100,
            "max_open_files": 131072,
            "nfs_file_gw": false,
            "nfs_obj_gw": false,
            "os_tuning_params": [
              {
                  "name": "fs.file-max",
                  "value": 26234859
              },
              {
                  "name": "kernel.pid_max",
                  "value": 4194303
              }
            ],
            "public_network": "10.10.0.0/24"
        },
        "inventory": {
            "_meta": {
                "hostvars": {
                    "10.10.0.10": {
                        "ansible_user": "ansible",
                        "devices": [
                            "/dev/vdc",
                            "/dev/vde",
                            "/dev/vdd",
                            "/dev/vdb"
                        ],
                        "monitor_interface": "ens3"
                    },
                    "10.10.0.11": {
                        "ansible_user": "ansible",
                        "devices": [
                            "/dev/vdc",
                            "/dev/vde",
                            "/dev/vdd",
                            "/dev/vdb"
                        ],
                        "monitor_interface": "ens3"
                    },
                    "10.10.0.12": {
                        "ansible_user": "ansible",
                        "devices": [
                            "/dev/vdc",
                            "/dev/vde",
                            "/dev/vdd",
                            "/dev/vdb"
                       ],
                        "monitor_interface": "ens3"
                    },
                    "10.10.0.8": {
                        "ansible_user": "ansible",
                        "devices": [
                            "/dev/vdc",
                            "/dev/vde",
                            "/dev/vdd",
                            "/dev/vdb"
                        ],
                        "monitor_interface": "ens3"
                    },
                    "10.10.0.9": {
                        "ansible_user": "ansible",
                        "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        }
                    }
                },
                "clients": [],
                "iscsi_gw": [],
                "mdss": [],
                "mons": [
                    "10.10.0.8"
                ],
                "nfss": [],
                "osds": [
                    "10.10.0.10",
                    "10.10.0.12",
                    "10.10.0.11",
                    "10.10.0.9"
                ],
                "rbdmirrors": [],
                "restapis": [
                    "10.10.0.8"
                ],
                "rgws": []
            }
        },
        "name": "deploy",
        "playbook_id": "cluster_deploy"
    },
    "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
    "initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1479907354,
    "version": 2
}
```

The example above shows replacing 10.10.0.9 in mons/restapis and adding it to the OSD list, and also placing the 10.10.0.8 from OSDs to mons/restapis. As a result, the playbook configuration ID is fd499a1e-866e-4808-9b89-5f582c6bd29e and the version is 2.

2. Save your changes and exit the editor. Proceed to Execute a playbook configuration.

Execute a playbook configuration

To execute a playbook configuration:

1. Run decapod execution create with the playbook configuration ID and version.

   Example:

   ```
   $ decapod execution create fd499a1e-866e-4808-9b89-5f582c6bd29e 2
   {
      "data": {
         "playbook_configuration": {
            "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
         },
         "state": "created"
      },
      "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
      "initiator_id": null,
      "model": "execution",
      "time_deleted": 0,
      "time_updated": 1479908503,
      "version": 1
   }
   ```

   Once done, the playbook configuration is in the created state. It takes some time for the execution to start.

2. To verify that the execution has started, use the decapod execution get command with the execution ID.

   Example:

   ```
    $ decapod execution get f2fbb668-6c89-42d2-9251-21e0b79ae973
    {
       "data": {
          "playbook_configuration": {
             "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
         },
         "state": "started"
      },
      "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
      "initiator_id": null,
      "model": "execution",
      "time_deleted": 0,
      "time_updated": 1479908503,
      "version": 2
   }
   ```

   Once completed, the execution state will turn to completed.

Additionally, you can perform the following actions:

- Track the execution steps using the decapod execution steps command with the execution ID.

  Example:

```
$ decapod execution steps f2fbb668-6c89-42d2-9251-21e0b79ae973
[
    {
        "data": {
            "error": {},
            "execution_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
            "name": "add custom repo",
            "result": "skipped",
            "role": "ceph.ceph-common",
            "server_id": "8dd33842-fee6-4ec7-a1e5-54bf6ae24710",
            "time_finished": 1479908609,
            "time_started": 1479908609
        },
        "id": "58359d01b3670f0089d9330b",
        "initiator_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "model": "execution_step",
        "time_deleted": 0,
        "time_updated": 1479908609,
        "version": 1
    },
    {
        "data": {
            "error": {},
            "execution_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
            "name": "add gluster nfs ganesha repo",
            "result": "skipped",
            "role": "ceph.ceph-common",
            "server_id": "8dd33842-fee6-4ec7-a1e5-54bf6ae24710",
            "time_finished": 1479908609,
            "time_started": 1479908609
        },
        "id": "58359d01b3670f0089d9330c",
        "initiator_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "model": "execution_step",
        "time_deleted": 0,
        "time_updated": 1479908609,
        "version": 1
    }
]
```

- View the execution history using the decapod execution get-version-all command with the execution ID.

  Example:

```
$ decapod execution get-version-all f2fbb668-6c89-42d2-9251-21e0b79ae973
[
   {
      "data": {
         "playbook_configuration": {
            "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
         },
         "state": "completed"
      },
      "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
      "initiator_id": null,
      "model": "execution",
      "time_deleted": 0,
      "time_updated": 1479909342,
      "version": 3
   },
   {
      "data": {
         "playbook_configuration": {
            "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
         },
         "state": "started"
      },
      "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
      "initiator_id": null,
      "model": "execution",
      "time_deleted": 0,
      "time_updated": 1479908503,
      "version": 2
   },
   {
      "data": {
         "playbook_configuration": {
            "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
         },
         "state": "created"
      },
      "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
      "initiator_id": null,
      "model": "execution",
      "time_deleted": 0,
      "time_updated": 1479908503,
      "version": 1
   }
]
```

- Once the execution is done, view the entire execution log using the decapod execution log command with the execution ID.

  Example:

```
$ decapod execution log f2fbb668-6c89-42d2-9251-21e0b79ae973
Using /etc/ansible/ansible.cfg as config file
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./checks/check_mandatory_vars.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./release.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/facts.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/deploy_monitors.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/start_monitor.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/ceph_keys.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/openstack_config.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/create_mds_filesystems.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/secure_cluster.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/./docker/main.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/checks.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/pre_requisite.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/dirs_permissions.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/create_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/fetch_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/selinux.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/start_docker_monitor.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/docker/copy_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-mon/tasks/calamari.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-agent/tasks/pre_requisite.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph-agent/tasks/start_agent.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./checks/check_mandatory_vars.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./release.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/facts.yml
statically included: /usr/local/lib/python2.7/dist-packages/\
decapod_ansible/ceph-ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml

...

TASK [ceph-restapi : run the ceph rest api docker image] ***********************
task path: /usr/local/lib/python2.7/dist-packages/decapod_ansible/\
ceph-ansible/roles/ceph-restapi/tasks/docker/start_docker_restapi.yml:2
skipping: [10.10.0.8] => {"changed": false, "skip_reason": "Conditional check failed", "skipped": true}

PLAY [rbdmirrors] **************************************************************
skipping: no hosts matched

PLAY [clients] *****************************************************************
skipping: no hosts matched

PLAY [iscsigws] ****************************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************
10.10.0.10                 : ok=61   changed=12   unreachable=0    failed=0
10.10.0.11                 : ok=60   changed=12   unreachable=0    failed=0
10.10.0.12                 : ok=60   changed=12   unreachable=0    failed=0
10.10.0.8                  : ok=90   changed=19   unreachable=0    failed=0
10.10.0.9                  : ok=60   changed=12   unreachable=0    failed=0
```

Admin service

Important

The Admin service must be used only by experienced users and administrators.

Along with ordinary Decapod Docker containers, docker-compose runs an optional but strongly recommended service called the admin service. This service provides special containers that act like a lightweight virtual machine with configured command-line interface and a decapod-admin tool that performs maintenance of low-level operations on Decapod or cluster.

This service has a number of additional utilities, such as vim, nano, less, jq, yaql, jmespath-terminal, and jp. Vim is configured as a default editor. Basically, it means that you can execute decapod from a container as is.

```
root@7252bfd5947d:/# decapod user get-all
[
    {
        "data": {
            "email": "noreply@example.com",
            "full_name": "Root User",
            "login": "root",
            "role_id": "e6ba587a-6256-401a-8734-8cead3d7a4c7"
        },
        "id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
        "initiator_id": null,
        "model": "user",
        "time_deleted": 0,
        "time_updated": 1487146111,
        "version": 1
    }
]
root@7252bfd5947d:/# decapod user get-all | jp '[0].id'
"7a52f762-7c2d-4164-b779-15f86f4aef2a"
root@7252bfd5947d:/# decapod user get-all | jq -r '.[0]|.id'
7a52f762-7c2d-4164-b779-15f86f4aef2a
```

The admin service runs Cron jobs that perform Keystone synchronization, monitoring, and data collection.

Additionally, the Decapod admin service enables various maintenance and admin tasks as described in the following topics.

Access the admin service

To access the Decapod admin service:

```
$ docker-compose -p PROJECT_NAME exec admin bash
```

Note

The -p option is the name of the project. If you have not specified it when running docker-compose, do not specify it now.

As a result, you will enter the container. The default environment allows you to run the decapod utility with a configured URL and login/password pair root/root.

```
root@7252bfd5947d:/# env | grep DECAPOD
DECAPOD_PASSWORD=root
DECAPOD_LOGIN=root
DECAPOD_URL=http://frontend:80
```

The admin service provides bundled documentation within the container. To access documentation:

1. Obtain the documentation port. The port is the value of the DECAPOD_DOCS_PORT environment variable and is 9998 by default.
2. Access the documentation using the obtained port and your credentials. For example, if you access Decapod using http://10.0.0.10:9999, the documentation will be served on http://10.0.0.10:9998.

Apply migrations

Migrations in Decapod are similar to migrations in databases but affect not only the schema but also the data. The main idea of such a migration is to adapt the existing data to a newer version of Decapod. For all available commands and options related to migrations, run decapod-admin migration --help.

To get a list of migrations, run decapod-admin migration list all.

Example:

```
root@7252bfd5947d:/# 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
```

You can apply migrations at any time. Decapod tracks migrations that have already been applied. To apply migrations, run decapod-admin migration apply or decapod-admin migration apply MIGRATION_NAME to apply a particular migration.

Example:

```
root@7252bfd5947d:/# decapod-admin migration apply
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:118 ): Lock \
applying_migrations was acquire by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:183 ): Prolong thread for \
locker applying_migrations of lock 071df271-d0ba-4fdc-83d0-49575d0acf3c \
has been started. Thread MongoLock prolonger \
071df271-d0ba-4fdc-83d0-49575d0acf3c for applying_migrations, ident 140625762334464
2017-02-15 10:19:25 [INFO    ] (   migration.py:119 ): No migration are \
required to be applied.
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:202 ): Prolong thread for \
locker applying_migrations of lock 071df271-d0ba-4fdc-83d0-49575d0acf3c \
has been stopped. Thread MongoLock prolonger \
071df271-d0ba-4fdc-83d0-49575d0acf3c for applying_migrations, ident 140625762334464
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:124 ): Try to release lock \
applying_migrations by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c.
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:140 ): Lock \
applying_migrations was released by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c.
```

To show details on a migration, run decapod-admin migration show MIGRATION_NAME.

Example:

```
root@7252bfd5947d:/# decapod-admin migration show 0006_create_cluster_data.py
Name:           0006_create_cluster_data.py
Result:         ok
Executed at:    Wed Feb 15 08:08:36 2017
SHA1 of script: 73eb7adeb1b4d82dd8f9bdb5aadddccbcef4a8b3

-- Stdout:
Migrate 0 clusters.

-- Stderr:
```

Generate cloud-init user data configuration

You can generate user data configuration for cloud-init as described in Generate user data. Alternatively, use the decapod-admin.

To generate user data configuration using the decapod-admin tool, run decapod-admin cloud-config [OPTIONS] PUBLIC_URL specifying the URL accessible by Ceph nodes. For all available options, run decapod-admin cloud-config --help.

Example:

```
root@7252bfd5947d:/# decapod-admin cloud-config http://10.0.0.10:9999
#cloud-config
packages: [python]
runcmd:
- [echo, === START DECAPOD SERVER DISCOVERY ===]
- [sh, -xc, 'grep -q ''/usr/share/server_discovery.sh'' /etc/rc.local || \
sed -i ''s?^exit 0?/usr/share/server_discovery.sh >> /var/log/\
server_discovery.log 2>\&1\nexit 0?'' /etc/rc.local']
- [sh, -xc, systemctl enable rc-local.service || true]
- [sh, -xc, /usr/share/server_discovery.sh 2>&1 | tee -a /var/log/\
server_discovery.log]
- [echo, === FINISH DECAPOD SERVER DISCOVERY ===]
users:
```

```
- groups: [sudo]
  name: ansible
  shell: /bin/bash
  ssh-authorized-keys: [ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC7K9bHPrSu5V\
  HnUOis2Uwc822fMyPTtwjfOkzNi/\
  oVOxmd1QE3DilrO5fJ33pRwEj7r1DfTlJmZWs8XwWaaUXkQ+iyfRPtgt/Ox+X5A/XaLdi/\
  yz7UjnHc8ERDUT/z73RzDwf21KNQOopGRuyhe+gvGZ5mhYDz3bnnYY9IRBNYaGw4bjS0q1\
  AbkPa1PvCo7P5b5UuRjhi4H74zCFkQD4evQsrQOcgev5GimnODqMntU0jnI/eEJwnnd1TcY\
  G7dS6FqMWpFX1gqcKjFIuqNTZLYzJu9U8mxxKmGOQSI6KWfP0etBw1YRHRIfdZmdaqSKHh0\
  ZhUUHjbf8Hb5Vqv1Fkzf0cGPbfrazEDI5FaVjkZMGFfdgs1be6xO7NHqzu1JJ3ZEur28o0A\
  QyOVvrEJIxQayDM0qyKi7B4+j6QDL0CDaWN3dUZO45il/KOm/eXCm4yQg0ImXHUmsDoW+6W\
  6akI/fSCAn8r9GK2QBBJPeTPA95WlOSXtICnrsqgb74yKPEsslzfrTUIiyoXBuuR9o5OoPX\
  ghKrazqcTeK/Vdl7w4nZ00O4jlIHMTrS1xyubN0QeBd+3D8Hy2bN5h7WjiJsZ2XhlKR0Z1i5\
  AbgCR9hfQl84aFIXRARz+6uuDDHe2ONXujcS9jhuN7SOLGckiaXNfAeAsbEkYZytnUgdoxbHYSfzw==]
  sudo: ['ALL=(ALL) NOPASSWD:ALL']
write_files:
- content: |
    #-*- coding: utf-8 -*-

    from __future__ import print_function

    import json
    import ssl
    import sys

    try:
        import urllib.request as urllib2
    except ImportError:
        import urllib2

    data = {
        "username": 'ansible',
        "host": sys.argv[1].lower().strip(),
        "id": sys.argv[2].lower().strip()
    }
    headers = {
        "Content-Type": "application/json",
        "Authorization": '26758c32-3421-4f3d-9603-e4b5337e7ecc',
        "User-Agent": "cloud-init server discovery"
    }

    def get_response(url, data=None):
        if data is not None:
            data = json.dumps(data).encode("utf-8")
        request = urllib2.Request(url, data=data, headers=headers)
        request_kwargs = {"timeout": 20}
        if sys.version_info >= (2, 7, 9):
            ctx = ssl.create_default_context()
            ctx.check_hostname = False
            ctx.verify_mode = ssl.CERT_NONE
            request_kwargs["context"] = ctx
        try:
            return urllib2.urlopen(request, **request_kwargs).read()
        except Exception as exc:
            print("Cannot request {0}: {1}".format(url, exc))

    metadata_ip = get_response('http://169.254.169.254/latest/meta-data/public-ipv4')
    if metadata_ip is not None:
        data["host"] = metadata_ip
        print("Use IP {0} discovered from metadata API".format(metadata_ip))

    response = get_response('http://10.0.0.10:9999', data)
    if response is None:
        sys.exit("Server discovery failed.")
    print("Server discovery completed.")
  path: /usr/share/server_discovery.py
  permissions: '0440'
- content: |
    #!/bin/bash
    set -xe -o pipefail

    echo "Date $(date) | $(date -u) | $(date '+%s')"

    main() {
        local ip="$(get_local_ip)"
        local hostid="$(get_local_hostid)"

        python /usr/share/server_discovery.py "$ip" "$hostid"
    }

    get_local_ip() {
        local remote_ipaddr="$(getent ahostsv4 "10.0.0.10" | head -n 1 | cut -f 1 -d ' ')"

        ip route get "$remote_ipaddr" | head -n 1 | rev | cut -d ' ' -f 2 | rev
    }

    get_local_hostid() {
        dmidecode | grep UUID | rev | cut -d ' ' -f 1 | rev
    }

    main
  path: /usr/share/server_discovery.sh
  permissions: '0550'
```

Back up and restore database

Using the decapod-admin tool, you can back up and restore MongoDB, the main storage system used by Decapod. The archive format created by this tool is a native MongoDB archive that is compressed by default.

The output of decapod-admin db backup and decapod-admin db restore is similar to the output of mongodump --archive --gzip and mongorestore --archive --gzip. The decapod-admin tool uses /etc/decapod/config.yaml to read Decapod MongoDB settings and correctly constructs the command line taking the SSL settings into account. To get a list of available commands and options, run decapod-admin db --help.

To back up the database:

```
$ decapod-admin db backup > backupfile
```

To restore the database:

```
$ decapod-admin db restore < backupfile
```

If you do not require compression, use the -r flag. In such case, mongodump and mongorestore will not use the --gzip flag.

Seealso

Archiving and compression in MongoDB tools

SSH to Ceph hosts

Using the decapod-admin tool, you can SSH to remote hosts with the same user as used by Ansible.

To SSH to a remote host, use the decapod-admin ssh server-ip SERVER_IP or decapod-admin ssh server-id SERVER_ID command.

Example:

```
root@7252bfd5947d:/# decapod-admin ssh server-id 8cf8af12-89a0-477d-85e7-ce6cbe5f8a07
2017-02-15 09:42:40 [DEBUG   ] (        ssh.py:111 ): Execute \
['/usr/bin/ssh', '-4', '-tt', '-x', '-o', 'UserKnownHostsFile=/dev/null', \
'-o', 'StrictHostKeyChecking=no', '-l', 'ansible', '-i', \
'/root/.ssh/id_rsa', '10.0.0.23']
Warning: Permanently added '10.0.0.23' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

171 packages can be updated.
73 updates are security updates.
```

```
Last login: Wed Feb 15 09:30:45 2017 from 10.0.0.10
ansible@ceph-node04:~$ whoami
ansible
```

For all available options, run decapod-admin ssh --help.

Execute SSH in parallel

You may need to execute commands on remote hosts in parallel. For such purposes, decapod-admin uses its own implementation of pdsh integrated with Decapod.

Using decapod-admin pdsh, you can execute commands on multiple hosts in parallel, upload files and download them from remote hosts. For details on decapod-admin pdsh usage and all available commands and options, run decapod-admin pdsh --help.

Examples

```
root@7252bfd5947d:/# decapod-admin pdsh exec -- ls -la
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: total 32
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 09:22 .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      : total 32
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:40 .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: total 36
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:00 .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw------- 1 ansible ansible    7 Feb 15 09:43 .bash_history
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      : total 32
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:30 .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
```

```
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
```

```
root@7252bfd5947d:/# decapod-admin pdsh upload /etc/decapod/config.yaml .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Start to upload /etc/decapod/config.yaml to .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Finished uploading of /etc/decapod/config.yaml to .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: Start to upload /etc/decapod/config.yaml to .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Start to upload /etc/decapod/config.yaml to .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Finished uploading of /etc/decapod/config.yaml to .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: Finished uploading of /etc/decapod/config.yaml to .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Start to upload /etc/decapod/config.yaml to .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Finished uploading of /etc/decapod/config.yaml to .

root@7252bfd5947d:/# decapod-admin pdsh exec -- ls -lah config.yaml
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
```

```
root@7252bfd5947d:/# decapod-admin pdsh download config.yaml results/
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Start to download config.yaml to results/9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Start to download config.yaml to results/26261da0-2dde-41e9-8ab6-8836c806623e
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Start to download config.yaml to results/8cf8af12-89a0-477d-85e7-ce6cbe5f8a07
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: Start to download config.yaml to results/62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93
```

Restore entities

You can restore entities that were explicitly or accidentally deleted, for example, a cluster, user, server, role, and others. To restore a deleted entity, use the decapod-admin restore ITEM_TYPE ITEM_ID command specifying the type of the entity and its ID.

Example:

```
root@7252bfd5947d:/# decapod-admin restore user 6805075b-e40d-4800-8520-8569dd7327bd
{
   "data": {
      "email": "test@example.com",
      "full_name": "Full",
      "login": "test",
      "role_id": null
   },
   "id": "6805075b-e40d-4800-8520-8569dd7327bd",
   "initiator_id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
   "model": "user",
   "time_deleted": 1487154755,
   "time_updated": 1487154755,
   "version": 2
}
Undelete item? [y/N]: y
{
   "data": {
      "email": "test@example.com",
      "full_name": "Full",
      "login": "test",
      "role_id": null
   },
   "id": "6805075b-e40d-4800-8520-8569dd7327bd",
   "initiator_id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
   "model": "user",
   "time_deleted": 0,
   "time_updated": 1487154769,
   "version": 3
}
```

For command options and entity types, run decapod-admin restore -h.

Unlock servers

All playbook executions lock the servers they use to eliminate issues caused by a concurrent execution. However, you may be required to manually unlock servers. To do so, use the decapod-admin locked-servers command.

To list all locked servers:

```
decapod-admin locked-servers get-all
```

To unlock a server:

```
decapod-admin locked-servers unlock SERVER_ID
```

For all available options, run decapod-admin locked-servers --help.

Reset password

You can reset a user password through the Decapod web UI. However, in some cases you may want to change the password bypassing the usual procedure. For example, if a user has an obsolete, non-working email or if you want to change the default root/root username and password pair.

To explicitly reset a user password:

```
decapod-admin password-reset [OPTIONS] USER_ID
```

If you do not pass the new password, decapod-admin will prompt you to enter it.

Example:

```
$ decapod-admin password-reset c83d0ede-aad1-4f1f-b6f0-730879974763
New password []:
Repeat for confirmation:
```

If you do not pass any password, the tool will generate one and output it to stdout.

Example:

```
$ decapod-admin password-reset c83d0ede-aad1-4f1f-b6f0-730879974763
New password []:
54\gE'1Ck_
```

For all available options, run decapod-admin password-reset -h.

Seealso

- jq
- YAQL
- JMESPath terminal
- jp

Monitor Ceph

Decapod supports integration with various monitoring systems. By default, it uses an in-house tool called ceph-monitoring. You can also integrate Decapod with other tools, such as Prometheus through Telegraf. For a list of supported plugins, see Playbook plugins.

The ceph-monitoring tool collects statistics on the cluster state and monitors its performance by running particular scripts under the admin service.

To access the collected data:

1. Obtain the Decapod monitoring port. The port is the value of the DECAPOD_MONITORING_PORT environment variable and is 10001 by default.
2. Access the data using the obtained port and your credentials. For example, if you access Decapod using http://10.0.0.10:9999, the data will be served on http://10.0.0.10:10001.

If there is no information available about a recently deployed cluster, try again in 15 minutes. If the data is still not accessible, obtain the logs of the admin service:

```
$ docker-compose -p myprojectname logs admin
```

Generate a diagnostic snapshot

To simplify the interaction between development and operation, Decapod supports diagnostic or debug snapshots, similar to Fuel snapshots. A snapshot is an archive that contains all information required to debug and troubleshoot issues.

Snapshots store the following information:

- Backup of the database
- Logs from services
- File docker-compose.yml
- Configuration files from Decapod services (config.yaml)
- Datetimes from services
- Data from ceph-monitoring
- Version of installed packages
- Git commit SHAs of Decapod itself
- Information about docker and containers

Snapshots do not store Ansible private keys or user passwords. Passwords are hashed by Argon2.

To generate a diagnostic snapshot:

Run the script:

```
$ ./scripts/debug_snapshot.py snapshot
```

Alternatively, if you have containers only, follow the steps below.

1. Run the following command:

   ```
   $ docker-compose exec -T admin cat /debug-snapshot | python - snapshot
   ```

2. Configure the snapshot settings as required:

```
$ docker-compose -p myproject exec -T admin cat /debug-snapshot | python - --help
usage: - [-h] [-f COMPOSE_FILE] [-p PROJECT_NAME] snapshot_path

Create a debug snapshot for Decapod.

positional arguments:
  snapshot_path         Path where to store snapshot (do not append extension,
                        we will do it for you).

optional arguments:
  -h, --help            show this help message and exit
  -f COMPOSE_FILE, --compose-file COMPOSE_FILE
                        path to docker-compose.yml file. (default:
                        /vagrant/docker-compose.yml)
  -p PROJECT_NAME, --project-name PROJECT_NAME
                        the name of the project. (default: vagrant)

Please find all logs in syslog by ident 'decapod-debug-snapshot'.

$ docker-compose -p myproject exec -T admin cat /debug-snapshot | python - -p myproject snapshot
```

As a result, you will get a snapshot like snapshot_path.*. The snapshot tool calculates the best compression algorithm available on your platform and uses its extension. Therefore, the snapshot may look like snapshot_path.tar.bz2 or snapshot_path.tar.xz depending on how your Python was built.

Upgrade Decapod

Before starting any upgrade procedure, perform a backup of Decapod. The upgrade procedure is consequent, which means before upgrading to the latest version, you must first upgrade to the next version. For example, to upgrade Decapod from version X to X+3, first upgrade to X+1, then to X+2, and only then to X+3.

Verify Decapod version

To verify the Decapod version installed:

1. Obtain the project name:

```
$ docker-compose ps | grep api | cut -f 1 -d '_' | sort -u
shrimp
```

2. Verify the Decapod version:

```
docker inspect -f '{{ .Config.Labels.version }}' $(docker-compose -p \
PROJ ps -q database | head -n 1)
```

Where PROJ is the project name.

Example:

```
docker inspect -f '{{ .Config.Labels.version }}' $(docker-compose -p \
PROJ ps -q database | head -n 1)
0.1.0
```

Upgrade Decapod from 0.1.x to 1.0

Prior to upgrading Decapod, perform the steps described in Prerequisites.

> Note
>
> You do not need to perform any changes in the existing Ceph deployments.

Prerequisites

Prior to upgrading Decapod from 0.1.x to 1.0.0, verify that you have completed the following tasks:

1. From the machine that runs Decapod, obtain the latest versions of Decapod 1.0 release series:

   ```
   $ git clone -b stable-1.0 --recurse-submodules https://github.com/Mirantis/ceph-lcm.git ~/decapod
   ```

2. Create a directory to store configuration files and private keys for Decapod:

   ```
   $ mkdir -p ~/decapod_runtime
   ```

3. Obtain the project name:

   ```
   $ docker-compose ps | grep api | cut -f 1 -d '_' | sort -u
   shrimp
   ```

   For simplicity, further examples use PROJ as the project name.

   > Note
   >
   > To avoid passing -p each time you run docker-compose, use the COMPOSE_PROJECT_NAME environment variable.

4. Copy the required configuration files to the ~/decapod_runtime directory:

```
$ cp ~/decapod/{.env,docker-compose.yml,docker-compose.override.yml} ~/decapod_runtime
```

5. Set the path to the SSH private key in the .env file:

   ```
   $ sed -i "s?^DECAPOD_SSH_PRIVATE_KEY=.*?DECAPOD_SSH_PRIVATE_KEY=$HOME/\
   decapod_runtime/id_rsa?" ~/decapod_runtime/.env
   ```

   Use the name of your private key if it differs from the id_rsa in the example above.

Upgrade Decapod

To upgrade Decapod from 0.1.x to 1.0.0:

1. Back up the database:

   * To use the existing configuration, run the following command from the directory where you run Decapod:

     ```
     $ docker exec -i proj_database_1 mongodump --gzip --archive --ssl \
     --sslAllowInvalidCertificates > ~/pre_upgrade
     ```

     Where proj is the lowercase container name.

     Note

     To restore the database:

     ```
     $ docker exec -i proj_database_1 mongorestore --drop --gzip \
     --archive --ssl --sslAllowInvalidCertificates < ~/pre_upgrade
     ```

   * To use the default configuration, rename the database in MongoDB from shrimp to decapod and back up the data.

     1. Rename the database:

        ```
        $ docker-compose -p PROJ exec database moshell
        MongoDB shell version: 3.2.10
        connecting to: false
        2017-02-14T06:38:15.400+0000 W NETWORK  [thread1] The server \
        certificate does not match the host name 127.0.0.1
        Welcome to the MongoDB shell.
        For interactive help, type "help".
        For more comprehensive documentation, see
                http://docs.mongodb.org/
        Questions? Try the support group
                http://groups.google.com/group/mongodb-user
        ```

```
Server has startup warnings:
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] ** WARNING:\
 /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] ** WARNING:\
 /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
> db.copyDatabase("shrimp", "decapod", "localhost")
{ "ok" : 1 }
> use shrimp
switched to db shrimp
> db.dropDatabase()
{ "dropped" : "shrimp", "ok" : 1 }
```

2. Back up the database:

```
$ docker exec -i proj_database_1 mongodump --gzip --archive --ssl \
--sslAllowInvalidCertificates > ~/pre_upgrade_renamed
```

2. Optional. If you have modified any configuration files such as config.yaml or id_rsa, copy the files to a custom directory, for example, ~/decapod_runtime. To do so, run the following commands from the same directory used to run Decapod 0.1:

```
$ mkdir ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q api):/etc/shrimp/config.yaml" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q controller):/root/.ssh/id_rsa" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/dhparam.pem" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/ssl.crt" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/ssl.key" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q database):/certs/mongodb.pem" ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q database):/certs/mongodb-ca.crt" ~/decapod_runtime
```

Note

If you did not generate any custom configuration files and used the default configuration, skip this step and proceed to step 4.

3. Obtain the Decapod 1.0.0 images. To do so, follow steps 1-2 in the Install Decapod section of MCP Deployment Guide.

> Note
>
> The required configuration files are stored in ~/decapod_runtime and the repository for Decapod 1.0.0 is cloned to ~/decapod as described in Prerequisites.

4. Stop and remove containers for version 0.1.x. Since Docker containers are stateless and you have created a backup of the state (the database backup), drop the existing containers and start new ones. Execute the following command from the directory where you run Decapod:

   ```
   $ docker-compose -p PROJ down -v
   ```

5. Run Decapod 1.0.0.

   1. Change the directory to ~/decapod_runtime.
   2. Run Decapod:

      ```
      $ docker-compose -p PROJ up --remove-orphans -d
      ```

6. Restore the database:

   ```
   $ docker exec -i $(docker-compose -p PROJ ps -q admin) decapod-admin db \
   restore < ~/pre_upgrade_renamed
   ```

   Alternatively, if you did not rename the database:

   ```
   $ docker exec -i (docker-compose -p PROJ ps admin) decapod-admin db restore < ~/pre_upgrade
   ```

7. Apply migrations:

   ```
   $ docker-compose -p PROJ exec admin decapod-admin migration apply
   ```

8. Optional. You can configure MongoDB to be not backward compatible with the previous release. To do so, run:

   ```
   $ docker-compose -p PROJ exec database moshell
   MongoDB server version: 3.4.2
   Welcome to the MongoDB shell.
   For interactive help, type "help".
   For more comprehensive documentation, see
           http://docs.mongodb.org/
   Questions? Try the support group
           http://groups.google.com/group/mongodb-user
   Server has startup warnings:
   2017-02-14T07:00:13.729+0000 I STORAGE  [initandlisten]
   ```

```
2017-02-14T07:00:13.730+0000 I STORAGE  [initandlisten] ** WARNING: \
Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2017-02-14T07:00:13.730+0000 I STORAGE  [initandlisten] **          \
See http://dochub.mongodb.org/core/prodnotes-filesystem
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
Access control is not enabled for the database.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **          \
Read and write access to data and configuration is unrestricted.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
> db.adminCommand({setFeatureCompatibilityVersion: "3.4"})
{ "ok" : 1  }
```

9. Optional. Change the root password as described in Reset password.

Seealso

Back up and restore Decapod

Back up and restore Decapod

The decapod-admin tool allows you to manually create a backup of Decapod and its configuration and restore it.

Decapod stores its state in MongoDB. Restoring the database backups restores all the Decapod data except the internal container state, such as the data from ceph-monitoring that is refreshed every 10 minutes by default. However, you can collect such data explicitly using the docker-compose exec controller decapod-collect-data command.

To perform a backup:

```
$ docker-compose exec -T admin decapod-admin db backup > db_backup
```

To restore Decapod:

```
$ docker exec -i $(docker-compose ps -q admin) admin decapod-admin restore < db_backup
```

Note

Using docker-compose exec to perform the restore is currently not possible due to a docker-compose bug.

Alternatively, use the backup_db.py and restore_db.py scripts in the ./scripts directory:

1. Run the scripts:

   ```
   $ ./scripts/backup_db.py /var/backup/decapod_db
   $ ./scripts/restore_db.py /var/backup/decapod_db
   ```

2. Add the backup to Cron:

   ```
   0 */6 * * * /home/user/decapod_scripts/backup_db.py -p decapod -f \
   /home/user/decapod_runtime/docker-compose.yml /var/backups/decapod/\
   decapod_$(date --iso-8601) > /var/log/cron.log 2>&1
   ```

Seealso

- Monitor Ceph
- Admin service

Secure installation

See MCP Security Best Practices

Back up and restore a MySQL database

The Mirantis Cloud Platform (MCP) uses MySQL databases to store the data generated by different components of MCP. Mirantis recommends backing up your MySQL databases daily to ensure the integrity of your data. You should create an instant backup before upgrading your MySQL database. You may also need to create a MySQL database backup for testing purposes.

You can choose from the following backup tools:

- Backupninja that can copy databases along with backing up and restoring them. For example, copy the nova database to the nova_upgrade database.
- Xtrabackup that only backs up and restores the databases but is faster than Backupninja.

Back up and restore a MySQL database using Backupninja

MCP uses the Backupninja utility to back up MySQL databases. Backupninja installs automatically with your cloud environment using a SaltStack formula and includes the following components:

- Backupninja server collects requests from the Backupninja client nodes and runs on any node, for example, the Salt Master node.
- Backupninja client sends database backups to the Backupninja server and runs on the database cluster nodes.

This section describes how to create and restore your MySQL databases using Backupninja.

Create a backup schedule for a MySQL database using Backupninja

This section describes how to create a backup schedule for a MySQL database using the Backupninja utility. By default, Backupninja runs daily at 1.00 AM.

To create a backup schedule for a MySQL database:

1. Log in to the Salt Master node.

2. Add the following lines to cluster/openstack/database_init.yml:

```
classes:
- system.backupninja.client.single
- system.openssh.client.root
parameters:
  _param:
    backupninja_backup_host: <IP>
```

> Note
>
> The backupninja_backup_host parameter is the backupninja server that runs on any server, for example, on the Salt Master node.

3. Include the following pillar to the node that runs the backupninja server and will store the database backups remotely:

```
classes:
- system.backupninja.server.single
```

4. Run the salt.minion state:

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

5. Refresh grains for the backupninja client node:

```
salt-call state.apply salt.minion.grains
```

6. Refresh grains for the backupninja server node:

```
salt-call state.apply salt.minion.grains
```

7. Run the backupninja state on the backupninja server node:

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

8. Run the backupninja state on the backupninja client node:

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

   Running this state creates two backup configuration scripts in /etc/backup.d/ that will run daily at 1.00 AM.

Seealso

- Create an instant backup of a MySQL database using Backupninja
- Restore a MySQL database using Backupninja

Create an instant backup of a MySQL database using Backupninja

After you create a backup schedule as described in Create a backup schedule for a MySQL database using Backupninja, you may also need to create an instant backup of a MySQL database.

To create an instant backup of a MySQL database:

1. Verify that you have completed the steps described in Create a backup schedule for a MySQL database using Backupninja.
2. Log in to the MySQL Galera Database Master node, for example, dbs01.
3. Compress all MySQL databases and store them in /var/backups/mysql/sqldump/:

```
backupninja -n --run /etc/backup.d/101.mysql
```

4. Move the local backup files to the backupninja server using rsync:

```
backupninja -n --run /etc/backup.d/200.backup.rsync
```

Seealso

Restore a MySQL database using Backupninja

Restore a MySQL database using Backupninja

You may need to restore a MySQL database after a hardware or software failure or if you want to create a clone of the existing database from a backup.

To restore a MySQL database using Backupninja:

1. Log in to the Salt Master node.
2. Include the following pillar to cluster/openstack/database_init.yml to restore the MySQL database from any database node:

```
classes:
- system.mysql.client.single_init
parameters:
  _param:
    backupninja_backup_host: <IP>
```

3. Run the mysql.client state on the database node:

   ```
   salt-call state.sls mysql.client
   ```

   Running this state restores the databases and creates a file for every restored database in /root/mysql/flags.

   > Caution!
   >
   > If you rerun the state, it will not restore the database again. To repeat the restore procedure for any database, first delete the database file from /root/mysql/flags and then rerun the mysql.client state again.

Back up and restore a MySQL database using Xtrabackup

MCP uses the Xtrabackup utility to back up MySQL databases. Xtrabackup installs automatically with your cloud environment using a SaltStack formula and includes the following components:

- Xtrabackup server collects requests from the Xtrabackup client nodes and runs on any node, for example, the Salt Master node.
- Xtrabackup client sends database backups to the Xtrabackup server and runs on the MySQL Galera Database Master node.

This section describes how to create and restore your MySQL databases using Xtrabackup.

Create a backup schedule for a MySQL database using Xtrabackup

To ensure the consistent and timely backing up of your data, create a backup schedule using Xtrabackup.

To create a backup schedule for a MySQL database:

1. Log in to the Salt Master node.
2. Configure the xtrabackup server role by adding the following lines to cluster/infra/config.yml:

   ```
   classes:
   - system.xtrabackup.server.single
   ```

   By default, adding this include statement results in Xtrabackup keeping five complete backups and their incrementals. To change the default setting, include the following pillar to cluster/infra/config.yml.

```
parameters:
  xtrabackup:
    server:
      enabled: true
      hours_before_full: 48
      full_backups_to_keep: 5
```

3. Configure the xtrabackup client role by adding the following lines to cluster/openstack/database.init:

   ```
   classes:
   - system.xtrabackup.client.single
   parameters:
     _param:
       # Change the hostname to the Salt Master node hostname if unsure.
       xtrabackup_remote_server: cfg01
   ```

   > Note
   >
   > The xtrabackup_remote_server parameter should contain the resolvable hostname of the host where the xtrabackup server is running.

   By default, adding this include statement results in Xtrabackup keeping three complete backups and their incrementals on the xtrabackup client node. To change the default setting, include the following pillar to cluster/openstack/database.init:

   ```
   parameters:
     xtrabackup:
       client:
         enabled: true
         # The number of complete backups and their increments to keep
         # on the MySQL Galera Database Master node.
         full_backups_to_keep: 3
         hours_before_full: 48
         # This parameter schedules a cron job to run incremental backups
         # every 12 hours.
         hours_before_incr: 12
         compression: true
         compression_threads: 2
         database:
           user: username
           password: password
         target:
           # The xtrabackup server host. In this example, it is the
           # Salt Master node where backups will be stored.
           host: cfg01
   ```

4. Verify that the hours_before_full parameter of the xtrabackup client in cluster/openstack/database.init matches the same parameter of the xtrabackup server in cluster/infra/config.yml.

5. Run the xtrabackup server state:

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

6. Run the xtrabackup client state:

```
salt -C 'I@xtrabackup:client' state.sls openssh.client,xtrabackup
```

> Seealso
>
> - Create an instant backup of a MySQL database using Xtrabackup
> - Restore a MySQL database using Xtrabackup

Create an instant backup of a MySQL database using Xtrabackup

After you create a backup schedule as described in Create a backup schedule for a MySQL database using Xtrabackup, you may also need to create an instant backup of a MySQL database.

To create an instant backup for a MySQL database using Xtrabackup:

1. Verify that you have completed the steps described in Create a backup schedule for a MySQL database using Xtrabackup.
2. Log in to the MySQL Galera Database Master node, for example, dbs01.
3. Run the following script:

```
/usr/local/bin/innobackupex-runner.sh
```

4. Verify that a complete backup was created on the MySQL Galera Database Master node:

```
ls /srv/backups/mysql/xtrabackup/full
```

> Note
>
> If you rerun /usr/local/bin/innobackupex-runner.sh, it creates an incremental backup for the previous complete backup in /srv/backups/mysql/xtrabackup/incr.

5. Verify that the complete backup was rsynced to the xtrabackup server node from the Salt Master node:

```
salt -C 'I@xtrabackup:server' cmd.run 'ls /srv/backups/mysql/xtrabackup/full'
```

> Note
>
> If you run /usr/local/bin/innobackupex-runner.sh more than once, at least one incremental backup is created in /srv/backups/mysql/xtrabackup/incr on the node.

> Seealso
>
> Restore a MySQL database using Xtrabackup

Restore a MySQL database using Xtrabackup

You may need to restore a MySQL database after a hardware or software failure or if you want to create a clone of the existing database from a backup.

To restore a MySQL database using Xtrabackup:

1. Log in to the Salt Master node.
2. Add the following lines to cluster/openstack/database_init.yml:

   ```
   xtrabackup:
     client:
       enabled: true
       restore_full_latest: 1
       restore_from: remote
   ```

   where:

   * restore_full_latest can have the following values: 1 or 2. 1 means restoring the database from the last complete backup and its increments. 2 means restoring the second latest complete backup and its increments.
   * restore_from can have the following values: local or remote. The remote value uses scp to get the files from the xtrabackup server.
3. Proceed with either automatic restore steps using the Jenkins web UI pipeline or with manual restore steps:

   * Automatic restore steps:

     1. Add the upgrade pipeline to DriveTrain:

        1. Add the following lines to cluster/cicd/control/leader.yml:

           ```
           classes:
           - system.jenkins.client.job.deploy.update.restore_mysql
           ```
        2. Run the salt state.sls jenkins.client state.

2. Log in to the Jenkins web UI.
3. Open the Xtrabackup - restore mysql db pipeline.
4. In the SALT_MASTER_CREDENTIALS field, add salt-qa-credentials as credentials for connection.
5. In the SALT_MASTER_URL field, add the IP address of you Salt Master node host and salt-api port, for example, http://172.18.170.27:6969
6. Click Deploy.

- Manual restore steps:

1. Stop the mysql service on the MySQL Galera Database dbs02 and dbs03 nodes:

   ```
   salt -C 'I@galera:slave' service.stop mysql
   ```

2. Remove the MySQL log files from the MySQL Galera Database dbs02 and dbs03 nodes:

   ```
   salt -C 'I@galera:slave' cmd.run 'rm /var/lib/mysql/ib_logfile*'
   ```

3. Stop the mysql service on the MySQL Galera Database Master node:

   ```
   salt -C 'I@galera:master' service.stop mysql
   ```

4. Log in to the MySQL Galera Database Master node.
5. Replace the wsrep_cluster_address row in /etc/mysql/my.cnf with the following:

   ```
   wsrep_cluster_address="gcomm://"
   ```

6. Log in to the Salt Master node.
7. Move the MySQL database files to a new location /root/mysql/mysql.bak/ on the MySQL Galera Database Master node:

   ```
   salt -C 'I@galera:master' cmd.run 'mkdir /root/mysql/mysql.bak/'
   salt -C 'I@galera:master' cmd.run 'mv /var/lib/mysql/* /root/mysql/mysql.bak'
   salt -C 'I@galera:master' cmd.run 'rm /var/lib/mysql/.galera_bootstrap'
   ```

8. Verify that the MySQL database files are removed from /var/lib/mysql/ on the MySQL Galera Database Master node:

   ```
   salt -C 'I@galera:master' cmd.run 'ls /var/lib/mysql/'
   salt -C 'I@galera:master' cmd.run 'ls -ld /var/lib/mysql/.?*'
   ```

9. Log in to the MySQL Galera Database Master node where the restore operation occurs.
10. Run the xtrabackup state:

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

This state restores the databases and creates a file in

Caution!

If you rerun the state, it will not restore the database again. To repeat the restore procedure, first delete the /srv/backups/mysql/xtrabackupd/dbrestored file and then rerun the xtrabackup state again.

11. Log in to the Salt Master node.
12. Verify that the MySQL database files are present again on the MySQL Galera Database Master node:

    ```
    salt -C 'I@galera:master' cmd.run 'ls /var/lib/mysql/'
    ```

13. Start the mysql service on the MySQL Galera Database Master node:

    ```
    salt -C 'I@galera:master' service.start mysql
    ```

    Note

    This process takes a certain amount of time and does not provide an immediate output.

14. Start the mysql service on the MySQL Galera Database dbs02 and dbs03 nodes from the Salt Master node:

    ```
    salt -C 'I@galera:slave' service.start mysql
    ```

    Note

    This process takes a certain amount of time and does not provide an immediate output.

15. Verify that all MySQL Galera Database nodes joined the Galera cluster:

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

Update and upgrade

This section describes update and upgrade procedures for MCP clusters.

Minor update

Minor updates enable:

- Delivering hot fixes to source code of OpenStack, Kubernetes, or MCP Control Plane.
- Updating packages for an OpenStack service.
- Applying security patches to operating system components and packages.

Change service configuration

MCP Control Plane enables you to apply service configuration changes to your cluster leveraging the CI/CD pipeline that will run your change through review and deployment processes.

To change service configuration:

1. Change a Metadata model of your cluster.
2. Commit your change to Gerrit.
3. Submit the patch to the Git repository that will trigger the update job automatically.

   > Note
   >
   > You can trigger the job manually in Jenkins web UI by selecting Rebuild in in the drop-down menu of a build.

4. Go to Update Service(s) Configuration job in Jenkins available from the customer's DevOps Portal.
5. [Optional] Modify the default build parameters and rebuild.

   1. Select Rebuild in the drop-down menu of the current build and modify the following parameters:

Parameter	Description	Comment
SALT_MASTER_CREDENTIALS	Credentials to the Salt API.	N/A
SALT_MASTER_URL	Full Salt API address. For example, https://10.10.10.1:8000.	N/A
TARGET_BATCH_LIVE	Batch size for the complete live configurations changes on all nodes, empty string means apply to all targeted nodes.	Use the batch mode to apply changes on more than 100 nodes.

TARGET_SERVERS	Salt compound target to match nodes to be updated. For example, G@osfamily:debian.	Use * to apply changes on all nodes.
TARGET_STATES	Configuration changes to be applied. For example, linux, linux,openssh, salt.minion.grains.	An empty string means running highstate.
TARGET_SUBSET_LIVE	The number of nodes to apply live the configurations changes.	Specify the subset number to apply live changes on more than 100 nodes.
TARGET_SUBSET_TEST	The number of nodes to test configuration changes.	An empty string means all targeted nodes.

2. Click Rebuild.

6. Approve the changes to be applied live on a sample node and all other nodes. Click the Proceed button in the web UI or Console.

> Note
>
> You can use both: the Console view or web UI to approve operations in the pipeline.

7. Verify the status of all tasks in the pipeline for the current build. The task items in the web UI should be green.

Update service packages

To update service packages:

1. Go to Update System Package(s) job in Jenkins available from the customer's DevOps Portal.
2. Specify the parameters for the build:

    1. Select Rebuild in the drop-down menu of the current build and specify the following parameters:

Parameter	Description	Comment
SALT_MASTER_CREDENTIALS	Credentials to the Salt API.	N/A
SALT_MASTER_URL	A full Salt API address. For example, https://10.10.10.1:8000.	N/A

TARGET_BATCH_LIVE	Batch size for the complete live configurations changes on all nodes, empty string means apply to all targeted nodes.	Use the batch mode to apply changes on more than 100 nodes.
TARGET_PACKAGES	List of packages to be updated. For example, <PACKAGE1>=<VERSION> <PACKAGE2>=<VERSION>.	An empty string means updating all packages to the latest version.
TARGET_SERVERS	Salt compound target to match nodes to be updated. For example, cfg01*.	Use * to apply changes on all nodes.
TARGET_SUBSET_LIVE	The number of nodes to apply live the configurations changes.	Specify the subset number to apply live changes on more than 100 nodes.
TARGET_SUBSET_TEST	The number of nodes to test configuration changes.	An empty string means all targeted nodes.

2. Click Rebuild.

3. If you did not specify the package names in the job parameters before, you will be asked to enter the package names during the job execution. For example, to update the nova-api packages, print:

```
nova-api
```

Note

Use * to update all available packages.

4. Approve the changes to be applied live on a sample node and all other nodes. Click the Proceed button in the web UI or Console.

Note

You can use both: the Console view or web UI to approve operations in the pipeline.

5. Verify the status of all tasks in the pipeline for the current build. The task items in the web UI should be green.

Add a service

You can install a new service to the Virtualized Control Plane through the MCP Control Plane using the DevOps Portal.

To add a service:

1. Modify YAML configuration files in a Reclass model to enable a new service.
2. Commit your change to Gerrit.
3. Submit the patch to the Git repository that will trigger the update job automatically.

   > Note
   >
   > You can trigger the job manually in Jenkins web UI by selecting Rebuild in in the drop-down menu of a build.

4. Go to Update Service(s) Configuration job in Jenkins available from the customer's DevOps Portal.
5. [Optional] Modify the default build parameters and rebuild.
   1. Select Rebuild in the drop-down menu of the current build and modify the following parameters:

Parameter	Description	Comment
SALT_MASTER_CREDENTIALS	Credentials to the Salt API.	N/A
SALT_MASTER_URL	Full Salt API address. For example, https://10.10.10.1:8000.	N/A
TARGET_BATCH_LIVE	Batch size for the complete live configurations changes on all nodes, empty string means apply to all targeted nodes.	Use the batch mode to apply changes on more than 100 nodes.
TARGET_SERVERS	Salt compound target to match nodes to be updated. For example, G@osfamily:debian.	Use * to apply changes on all nodes.
TARGET_STATES	Configuration changes to be applied. For example, linux, linux,openssh, salt.minion.grains.	An empty string means running highstate.

TARGET_SUBSET_LIVE	The number of nodes to apply live the configurations changes.	Specify the subset number to apply live changes on more than 100 nodes.
TARGET_SUBSET_TEST	The number of nodes to test configuration changes.	An empty string means all targeted nodes.

   2. Click Rebuild.

6. Approve the changes to be applied live on a sample node. Click the Proceed button in the web UI or Console.

   Note

   You can use both: the Console view or web UI to approve operations in the pipeline.

7. Go to a sample node where the service has been already installed and verify if the service is running properly.
8. Go back to the Jenkins job and approve the changes to be applied live on all other nodes.
9. Verify the status of all tasks in the pipeline for the current build. The task items in the web UI should be green.

Major upgrade

Major upgrades enable:

- Delivering new features of MCP Control Plane.
- Upgrading between OpenStack releases.
- Upgrading host and guest operating systems to new versions including kernels.
- Updating the LCM platform.

Troubleshooting

For MAAS troubleshooting see MAAS documentation