Sahara

Release 2014.1.2.dev1.g2160131

OpenStack Foundation

June 17, 2014

Contents

Sahara project aims to provide users with simple means to provision a Hadoop cluster at OpenStack by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more.

CHAPTER 1

Overview

1.1 Rationale

1.1.1 Introduction

Apache Hadoop is an industry standard and widely adopted MapReduce implementation. The aim of this project is to enable users to easily provision and manage Hadoop clusters on OpenStack. It is worth mentioning that Amazon provides Hadoop for several years as Amazon Elastic MapReduce (EMR) service.

Sahara aims to provide users with simple means to provision Hadoop clusters by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more. After user fills in all the parameters, Sahara deploys the cluster in a few minutes. Also Sahara provides means to scale already provisioned cluster by adding/removing worker nodes on demand.

The solution will address following use cases:

- fast provisioning of Hadoop clusters on OpenStack for Dev and QA;
- utilization of unused compute power from general purpose OpenStack IaaS cloud;
- “Analytics as a Service” for ad-hoc or bursty analytic workloads (similar to AWS EMR).

Key features are:

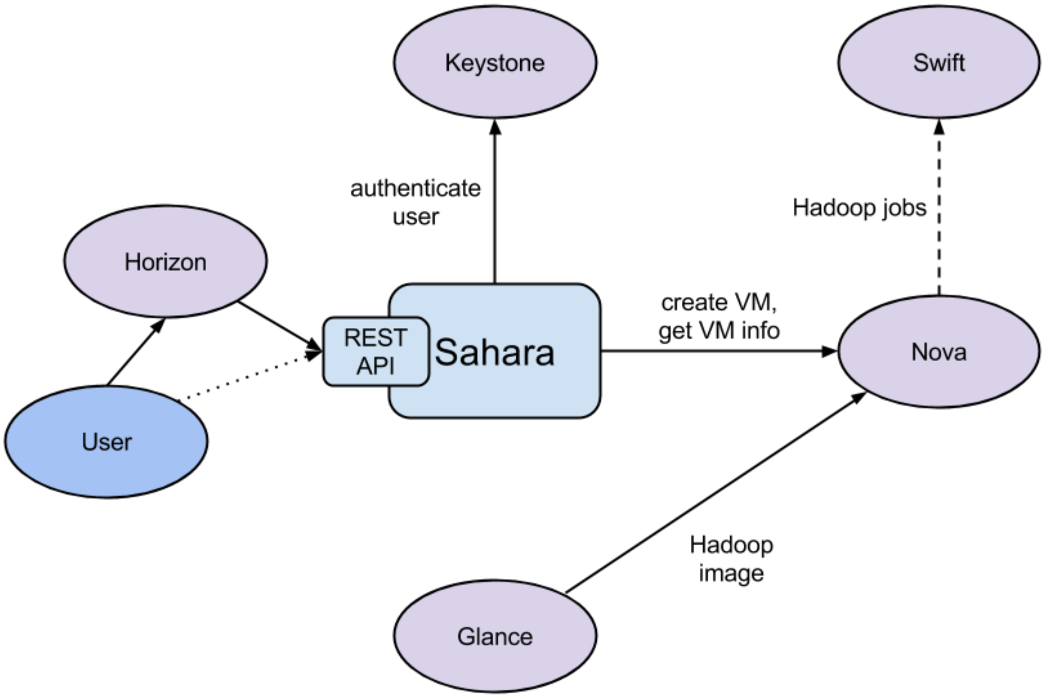
- designed as an OpenStack component;
- managed through REST API with UI available as part of OpenStack Dashboard;
- **support for different Hadoop distributions:**
    - pluggable system of Hadoop installation engines;
    - integration with vendor specific management tools, such as Apache Ambari or Cloudera Management Console;
- predefined templates of Hadoop configurations with ability to modify parameters.

1.1.2 Details

The Sahara product communicates with the following OpenStack components:

- Horizon - provides GUI with ability to use all of Sahara’s features;

- Keystone - authenticates users and provides security token that is used to work with the OpenStack, hence limiting user abilities in Sahara to his OpenStack privileges;
- Nova - is used to provision VMs for Hadoop Cluster;
- Glance - Hadoop VM images are stored there, each image containing an installed OS and Hadoop; the pre-installed Hadoop should give us good handicap on node start-up;
- Swift - can be used as a storage for data that will be processed by Hadoop jobs.



1.1.3 General Workflow

Sahara will provide two level of abstraction for API and UI based on the addressed use cases: cluster provisioning and analytics as a service.

For the fast cluster provisioning generic workflow will be as following:

- select Hadoop version;
- select base image with or without pre-installed Hadoop:
  - for base images without Hadoop pre-installed Sahara will support pluggable deployment engines integrated with vendor tooling;
- define cluster configuration, including size and topology of the cluster and setting the different type of Hadoop parameters (e.g. heap size):
  - to ease the configuration of such parameters mechanism of configurable templates will be provided;

- provision the cluster: Sahara will provision VMs, install and configure Hadoop;
- operation on the cluster: add/remove nodes;
- terminate the cluster when it’s not needed anymore.

For analytic as a service generic workflow will be as following:

- select one of predefined Hadoop versions;
- configure the job:
  - choose type of the job: pig, hive, jar-file, etc.;
  - provide the job script source or jar location;
  - select input and output data location (initially only Swift will be supported);
  - select location for logs;
- set limit for the cluster size;
- execute the job:
  - all cluster provisioning and job execution will happen transparently to the user;
  - cluster will be removed automatically after job completion;
- get the results of computations (for example, from Swift).

1.1.4 User’s Perspective

While provisioning cluster through Sahara, user operates on three types of entities: Node Group Templates, Cluster Templates and Clusters.

A Node Group Template describes a group of nodes within cluster. It contains a list of hadoop processes that will be launched on each instance in a group. Also a Node Group Template may provide node scoped configurations for those processes. This kind of templates encapsulates hardware parameters (flavor) for the node VM and configuration for Hadoop processes running on the node.

A Cluster Template is designed to bring Node Group Templates together to form a Cluster. A Cluster Template defines what Node Groups will be included and how many instances will be created in each. Some of Hadoop Configurations can not be applied to a single node, but to a whole Cluster, so user can specify this kind of configurations in a Cluster Template. Sahara enables user to specify which processes should be added to an anti-affinity group within a Cluster Template. If a process is included into an anti-affinity group, it means that VMs where this process is going to be launched should be scheduled to different hardware hosts.

The Cluster entity represents a Hadoop Cluster. It is mainly characterized by VM image with pre-installed Hadoop which will be used for cluster deployment. User may choose one of pre-configured Cluster Templates to start a Cluster. To get access to VMs after a Cluster has started, user should specify a keypair.

Sahara provides several constraints on Hadoop cluster topology. JobTracker and NameNode processes could be run either on a single VM or two separate ones. Also cluster could contain worker nodes of different types. Worker nodes could run both TaskTracker and DataNode, or either of these processes alone. Sahara allows user to create cluster with any combination of these options, but it will not allow to create a non working topology, for example: a set of workers with DataNodes, but without a NameNode.

Each Cluster belongs to some tenant determined by user. Users have access only to objects located in tenants they have access to. Users could edit/delete only objects they created. Naturally admin users have full access to every object. That way Sahara complies with general OpenStack access policy.

1.1.5 Integration with Swift

The Swift service is a standard object storage in OpenStack environment, analog of Amazon S3. As a rule it is deployed on bare metal machines. It is natural to expect Hadoop on OpenStack to process data stored there. There are a couple of enhancements on the way which can help there.

First, a FileSystem implementation for Swift: HADOOP-8545. With that thing in place, Hadoop jobs can work with Swift as naturally as with HDFS.

On the Swift side, we have the change request: Change I6b1ba25b (merged). It implements the ability to list endpoints for an object, account or container, to make it possible to integrate swift with software that relies on data locality information to avoid network overhead.

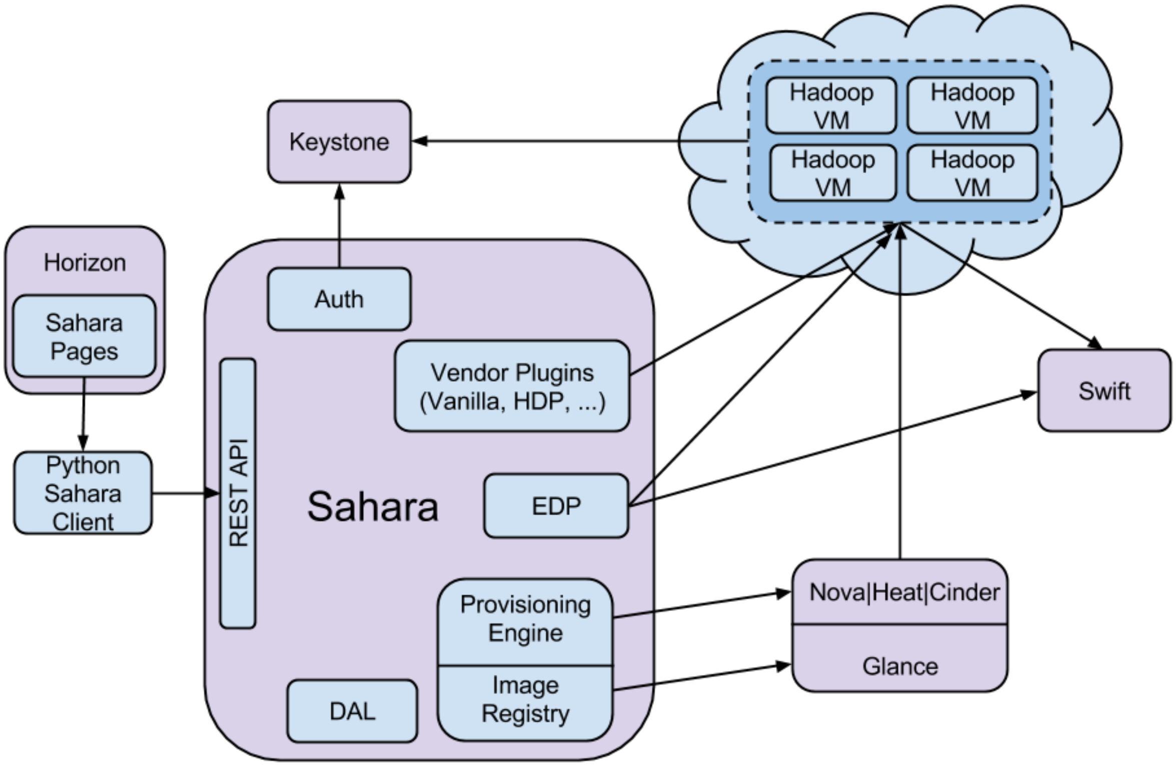
To get more information on how to enable Swift support see *Swift Integration*.

1.1.6 Pluggable Deployment and Monitoring

In addition to the monitoring capabilities provided by vendor-specific Hadoop management tooling, Sahara will provide pluggable integration with external monitoring systems such as Nagios or Zabbix.

Both deployment and monitoring tools will be installed on stand-alone VMs, thus allowing a single instance to manage/monitor several clusters at once.

1.2 Architecture



The Sahara architecture consists of several components:

- Auth component - responsible for client authentication & authorization, communicates with Keystone

- DAL - Data Access Layer, persists internal models in DB
- Provisioning Engine - component responsible for communication with Nova, Heat, Cinder and Glance
- Vendor Plugins - pluggable mechanism responsible for configuring and launching Hadoop on provisioned VMs; existing management solutions like Apache Ambari and Cloudera Management Console could be utilized for that matter
- EDP - *Elastic Data Processing (EDP)* responsible for scheduling and managing Hadoop jobs on clusters provisioned by Sahara
- REST API - exposes Sahara functionality via REST
- Python Sahara Client - similar to other OpenStack components Sahara has its own python client
- Sahara pages - GUI for the Sahara is located on Horizon

CHAPTER 2

User guide

Installation

2.1 Sahara Installation Guide

We recommend to install Sahara in a way that will keep your system in a consistent state. We suggest the following options:

- Install via Fuel
- Install via RDO Havana+
- Install into virtual environment

2.1.1 To install with Fuel

1. Start by following the Quickstart to install and setup OpenStack.
2. Enable Sahara service during installation.

2.1.2 To install with RDO

1. Start by following the Quickstart to install and setup OpenStack.
2. Install the sahara-api service:

```
$ yum install openstack-sahara
```

3. Configure the sahara-api service to your liking. The configuration file is located in `/etc/sahara/sahara.conf`. For details see *Sahara Configuration Guide*
4. Create database schema:

```
$ sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade head
```

5. Start the sahara-api service:

```
$ service openstack-sahara-api start
```

2.1.3 To install into a virtual environment

1. First you need to install a number of packages with your OS package manager. The list of packages depends on the OS you use. For Ubuntu run:

```
$ sudo apt-get install python-setuptools python-virtualenv python-dev
```

For Fedora:

```
$ sudo yum install gcc python-setuptools python-virtualenv python-devel
```

For CentOS:

```
$ sudo yum install gcc python-setuptools python-devel
$ sudo easy_install pip
$ sudo pip install virtualenv
```

2. Setup virtual environment for Sahara:

```
$ virtualenv sahara-venv
```

This will install python virtual environment into `sahara-venv` directory in your current working directory. This command does not require super user privileges and could be executed in any directory current user has write permission.

3. You can install the latest Sahara release from pypi:

```
$ sahara-venv/bin/pip install sahara
```

Or you can get Sahara archive from http://tarballs.openstack.org/sahara/ and install it using pip:

```
$ sahara-venv/bin/pip install 'http://tarballs.openstack.org/sahara/sahara-master.tar.gz'
```

Note that sahara-master.tar.gz contains the latest changes and might not be stable at the moment. We recommend browsing http://tarballs.openstack.org/sahara/ and selecting the latest stable release.

4. After installation you should create configuration file from a sample config located in `sahara-venv/share/sahara/sahara.conf.sample-basic`:

```
$ mkdir sahara-venv/etc
$ cp sahara-venv/share/sahara/sahara.conf.sample-basic sahara-venv/etc/sahara.conf
```

Make the necessary changes in `sahara-venv/etc/sahara.conf`. For details see *Sahara Configuration Guide*

5. If you use Sahara with MySQL database, then for storing big Job Binaries in Sahara Internal Database you must configure size of max allowed packet. Edit `my.cnf` and change parameter:

```
...
[mysqld]
...
max_allowed_packet          = 256M
```

and restart mysql server.

6. Create database schema:

```
$ sahara-venv/bin/sahara-db-manage --config-file sahara-venv/etc/sahara.conf upgrade head
```

7. To start Sahara call:

```
$ sahara-venv/bin/sahara-api --config-file sahara-venv/etc/sahara.conf
```

2.1.4 Notes:

One of the *Sahara Features*, Anti-Affinity, requires a Nova adjustment. See *Enabling Anti-Affinity* for details. But that is purely optional.

Make sure that your operating system is not blocking Sahara port (default: 8386). You may need to configure iptables in CentOS and some other operating systems.

To get the list of all possible options run:

```
$ sahara-venv/bin/python sahara-venv/bin/sahara-api --help
```

Further consider reading *Getting Started* for general Sahara concepts and *Provisioning Plugins* for specific plugin features/requirements.

2.2 Sahara Configuration Guide

This guide covers steps for basic configuration of Sahara. It will help you to configure the service in the most simple manner.

Let's start by configuring Sahara server. The server is packaged with two sample config files: `sahara.conf.sample-basic` and `sahara.conf.sample`. The former contains all essential parameters, while the later contains the full list. We recommend to create your config based on the basic sample, as most probably changing parameters listed here will be enough.

First, edit `connection` parameter in the `[database]` section. The URL provided here should point to an empty database. In case of SQLite, if the database file does not exist, it will be automatically created by `sahara-db-manage`. For instance, the following URL should work in most environments:

```
connection=sqlite:////tmp/sahara.db
```

Note that we recommend using MySQL or PostgreSQL backends for setups other than experimental. This is especially important if you plan to migrate later to a newer version of Sahara. With SQLite you will either have to start from scratch or migrate your DB to MySQL or PostgreSQL, might be non-trivial.

Return to the `[DEFAULT]` section. Start by editing the following 5 parameters:

```
os_auth_host
os_auth_port
os_admin_username
os_admin_password
os_admin_tenant_name
```

The first two parameters should point to Keystone public endpoint. The next three parameters must specify a user Sahara will use to verify credentials provided by users. The provided user must have `admin` role in the specified tenant.

Proceed to the networking parameters. If you are using Neutron for networking, then set

```
use_neutron=true
```

Otherwise if you are using Nova-Network set the given parameter to false.

That should be enough for the first run. If you want to increase logging level for troubleshooting, there are two parameters in the config: `verbose` and `debug`. If the former is set to true, Sahara will start to write logs of INFO level and above. If `debug` is set to true, Sahara will write all the logs, including the DEBUG ones.

2.3 Sahara UI Installation Guide

Sahara UI is a plugin for OpenStack Dashboard. There are two ways to install it. One is to plug it into existing Dashboard installation and another is to setup another Dashboard and plug Sahara UI there. The first approach advantage is that you will have Sahara UI in the very same Dashboard with which you work with OpenStack. The disadvantage is that you have to tweak your Dashboard configuration in order to enable the plugin. The second approach does not have this disadvantage.

Further steps describe installation for the first approach. For the second approach see *Sahara UI Dev Environment Setup*

2.3.1 1. Prerequisites

1. OpenStack IceHouse installed.
2. Sahara installed, configured and running, see *Sahara Installation Guide*.

2.3.2 2. Sahara Dashboard Installation

1. Go to the machine where Dashboard resides and install Sahara UI there:

   For RDO:

```
$ sudo yum install python-django-sahara
```

   Otherwise:

```
$ sudo pip install sahara-dashboard
```

   This will install the latest stable release of Sahara UI. If you want to install the development version of Sahara UI do the following instead:

```
$ sudo pip install http://tarballs.openstack.org/sahara-dashboard/sahara-dashboard-master.tar.gz
```

   Note that dev version might be broken at any time and also it might lose backward compatibility with Icehouse release at some point.

2. Configure OpenStack Dashboard. In `settings.py` add sahara to

```
HORIZON_CONFIG = {
    'dashboards': ('nova', 'syspanel', 'settings', ..., 'sahara'),
```

   and also add saharadashboard to

```
INSTALLED_APPS = (
    'saharadashboard',
    ....
```

   Note: `settings.py` file is located in `/usr/share/openstack-dashboard/openstack_dashboard/` by default.

3. Now let's switch to another file - `local_settings.py`. If you are using Neutron instead of Nova-Network add the following parameter there:

```
SAHARA_USE_NEUTRON = True
```

   If you are using Nova-Network with `auto_assign_floating_ip=False` add the following parameter:

```
AUTO_ASSIGNMENT_ENABLED = False
```

   Note: For RDO, the `local_settings.py` file is named `local_settings` and its absolute path is `/etc/openstack-dashboard/local_settings`, otherwise the file's absolute path is `/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py`.

4. You also need to tell Sahara UI where it can find Sahara service. There are two ways to do that. First is to define Sahara endpoint in Keystone. The endpoint type must be `data_processing`:

```
keystone service-create --name sahara --type data_processing \
    --description "Sahara Data Processing"

keystone endpoint-create --service sahara --region RegionOne \
    --publicurl "http://10.0.0.2:8386/v1.1/\$(tenant_id)s" \
    --adminurl "http://10.0.0.2:8386/v1.1/\$(tenant_id)s" \
    --internalurl "http://10.0.0.2:8386/v1.1/\$(tenant_id)s"
```

   While executing the commands above, don't forget to change IP addresses and ports to the ones actual for your setup.

   This approach might not work for you if your Keystone already has Sahara endpoint registered. This could be in DevStack and Fuel environments as both are capable to install Sahara and Sahara UI on their own. In that case use the second approach described below.

   The second way to tell Sahara UI where Sahara service is deployed is to specify `SAHARA_URL` parameter in `local_settings.py`. For example:

```
SAHARA_URL = 'http://localhost:8386/v1.1'
```

5. The installation is complete. You need to restart the apache web server for the changes to take effect.

   For Ubuntu:

```
$ sudo service apache2 restart
```

   For Centos:

```
$ sudo service httpd reload
```

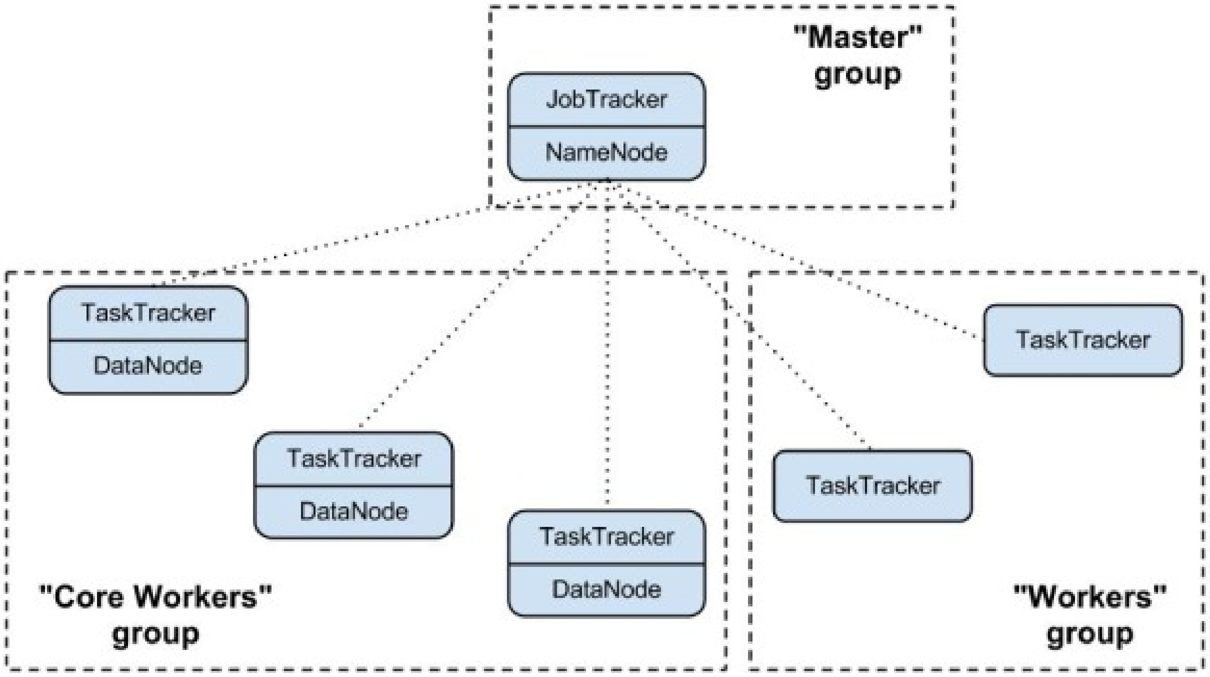
   Now if you log into Horizon you should see the Sahara menu available there.

How To

2.4 Getting Started

2.4.1 Clusters

A cluster deployed by Sahara consists of node groups. Node groups vary by their role, parameters and number of machines. The picture below illustrates example of Hadoop cluster consisting of 3 node groups each having different role (set of processes).



Node group parameters include Hadoop parameters like *io.sort.mb* or *mapred.child.java.opts*, and several infrastructure parameters like flavor for VMs or storage location (ephemeral drive or Cinder volume).

A cluster is characterized by its node groups and its parameters. Like a node group, cluster has Hadoop and infrastructure parameters. An example of cluster-wide Hadoop parameter is *dfs.replication*. For infrastructure an example could be image which will be used to launch cluster VMs.

2.4.2 Templates

In order to simplify cluster provisioning Sahara employs concept of templates. There are two kind of templates: node group template and cluster template. The former is used to create node groups, the later - clusters. Essentially templates have the very same parameters as corresponding entities. Their aim is to remove burden of specifying all the required parameters each time user wants to launch a cluster.

In the REST interface templates have extended functionality. First you can specify node-scoped parameters here, they will work as a defaults for node groups. Also with REST interface during cluster creation user can override template parameters for both cluster and node groups.

2.4.3 Provisioning Plugins

A provisioning plugin is a component responsible for provisioning Hadoop cluster. Generally each plugin is capable of provisioning a specific Hadoop distribution. Also plugin can install management and/or monitoring tools for a cluster.

Since Hadoop parameters vary depending on distribution and Hadoop version, templates are always plugin and Hadoop version specific. A template could not be used with plugin/Hadoop version different than ones it was created for.

You may find the list of available plugins on that page: *Provisioning Plugins*

2.4.4 Image Registry

OpenStack starts VMs based on pre-built image with installed OS. The image requirements for Sahara depend on plugin and Hadoop version. Some plugins require just basic cloud image and install Hadoop on VMs from scratch. Some plugins might require images with pre-installed Hadoop.

The Sahara Image Registry is a feature which helps filter out images during cluster creation. See *Registering an Image* for details on how to work with Image Registry.

2.4.5 Features

Sahara has several interesting features. The full list could be found there: *Features Overview*

2.5 Sahara UI User Guide

This guide assumes that you already have sahara-api and the Sahara Dashboard configured and running. If you require assistance with that, please see the installation guides.

2.5.1 Launching a cluster via the Sahara Dashboard

2.5.2 Registering an Image

1. Navigate to the "Sahara" tab in the dashboard, then click on the "Image Registry" panel.
2. From that page, click on the "Register Image" button at the top right.
3. Choose the image that you'd like to register as a Hadoop Image
4. Enter the username of the cloud-init user on the image.
5. Click on the tags that you want to add to the image. (A version ie: 1.2.1 and a type ie: vanilla are required for cluster functionality)
6. Click the "Done" button to finish the registration.

2.5.3 Create Node Group Templates

1. Navigate to the "Sahara" tab in the dashboard, then click on the "Node Group Templates" panel.
2. From that page, click on the "Create Template" button at the top right.
3. Choose your desired Plugin name and Version from the dropdowns and click "Create".
4. Give your Node Group Template a name (description is optional)
5. Choose a flavor for this template (based on your CPU/memory/disk needs)
6. Choose the storage location for your instance, this can be either "Ephemeral Drive" or "Cinder Volume". If you choose "Cinder Volume", you will need to add additional configuration.
7. Choose which processes should be run for any instances that are spawned from this Node Group Template.
8. Click on the "Create" button to finish creating your Node Group Template.

2.5.4 Create a Cluster Template

1. Navigate to the "Sahara" tab in the dashboard, then click on the "Cluster Templates" panel.
2. From that page, click on the "Create Template" button at the top right.
3. Choose your desired Plugin name and Version from the dropdowns and click "Create".
4. Under the "Details" tab, you must give your template a name.
5. Under the "Node Groups" tab, you should add one or more nodes that can be based on one or more templates.

- To do this, start by choosing a Node Group Template from the dropdown and click the "+" button.
- You can adjust the number of nodes to be spawned for this node group via the text box or the "-" and "+" buttons.
- Repeat these steps if you need nodes from additional node group templates.

6. Optionally, you can adjust your configuration further by using the "General Parameters", "HDFS Parameters" and "MapReduce Parameters" tabs.
7. Click on the "Create" button to finish creating your Cluster Template.

2.5.5 Launching a Cluster

1. Navigate to the "Sahara" tab in the dashboard, then click on the "Clusters" panel.
2. Click on the "Launch Cluster" button at the top right.
3. Choose your desired Plugin name and Version from the dropdowns and click "Create".
4. Give your cluster a name. (required)
5. Choose which cluster template should be used for your cluster.
6. Choose the image that should be used for your cluster (if you do not see any options here, see Registering an Image above).
7. Optionally choose a keypair that can be used to authenticate to your cluster instances.
8. Click on the "Create" button to start your cluster.

- Your cluster's status will display on the Clusters table.
- It will likely take several minutes to reach the "Active" state.

2.5.6 Scaling a Cluster

1. From the Sahara/Clusters page, click on the "Scale Cluster" button of the row that contains the cluster that you want to scale.
2. You can adjust the numbers of instances for existing Node Group Templates.
3. You can also add a new Node Group Template and choose a number of instances to launch.

- This can be done by selecting your desired Node Group Template from the dropdown and clicking the "+" button.
- Your new Node Group will appear below and you can adjust the number of instances via the text box or the +/- buttons.

4. To confirm the scaling settings and trigger the spawning/deletion of instances, click on "Scale".

2.5.7 Elastic Data Processing (EDP)

2.5.8 Data Sources

Data Sources are where the input and output from your jobs are housed.

1. From the Sahara/Data Sources page, click on the "Create Data Source" button at the top right.
2. Give your Data Source a name.
3. Enter the URL to the Data Source.

- For a Swift object, the url will look like <container>.sahara/<path> (ie: mycontainer.sahara/inputfile). The "swift://" is automatically added for you.
- For an HDFS object, the url will look like <host>/<path> (ie: myhost/user/hadoop/inputfile). The "hdfs://" is automatically added for you.

4. Enter the username and password for the Data Source.
5. Enter an optional description.
6. Click on "Create".
7. Repeat for additional Data Sources.

2.5.9 Job Binaries

Job Binaries are where you define/upload the source code (mains and libraries) for your job.

1. From the Sahara/Job Binaries page, click on the "Create Job Binary" button at the top right.
2. Give your Job Binary a name (this can be different than the actual filename).
3. Choose the type of storage for your Job Binary.

- For "Swift", you will need to enter the URL of your binary (<container>.sahara/<path>) as well as the username and password.
- For "Internal database", you can choose from "Create a script" or "Upload a new file".

4. Enter an optional description.
5. Click on "Create".
6. Repeat for additional Job Binaries

2.5.10 Jobs

Jobs are where you define the type of job you'd like to run as well as which "Job Binaries" are required.

1. From the Sahara/Jobs page, click on the "Create Job" button at the top right.
2. Give your Job a name.
3. Choose the type of job you'd like to run (Pig, Hive, MapReduce, Streaming MapReduce, Java Action)
4. Choose the main binary from the dropdown (not applicable for MapReduce or Java Action).
5. Enter an optional description for your Job.
6. Optionally, click on the "Libs" tab and add one or more libraries that are required for your job. Each library must be defined as a Job Binary.

7. Click on “Create”.

2.5.11 Job Executions

Job Executions are what you get by “Launching” a job. You can monitor the status of your job to see when it has completed its run.

1. From the Sahara/Jobs page, find the row that contains the job you want to launch and click on the “Launch Job” button at the right side of that row.
2. Choose the cluster (already running–see Launching a Cluster above) on which you would like the job to run.
3. Choose the Input and Output Data Sources (Data Sources defined above).
4. If additional configuration is required, click on the “Configure” tab.

- Additional configuration properties can be defined by clicking on the “Add” button.
- An example configuration entry might be mapred.mapper.class for the Name and org.apache.oozie.example.SampleMapper for the Value.

5. Click on “Launch”. To monitor the status of your job, you can navigate to the Sahara/Job Executions panel.
6. You can relaunch a Job Execution from the Job Executions page by using the “Relaunch on New Cluster” or “Relaunch on Existing Cluster” links.

- Relaunch on New Cluster will take you through the forms to start a new cluster before letting you specify input/output Data Sources and job configuration.
- Relaunch on Existing Cluster will prompt you for input/output Data Sources as well as allow you to change job configuration before launching the job.

2.5.12 Additional Notes

1) Throughout the Sahara UI, you will find that if you try to delete an object that you will not be able to delete it if another object depends on it. An example of this would be trying to delete a Job that has an existing Job Execution. In order to be able to delete that job, you would first need to delete any Job Executions that relate to that job.

2.6 Features Overview

2.6.1 Cluster Scaling

The mechanism of cluster scaling is designed to enable user to change the number of running instances without creating a new cluster. User may change number of instances in existing Node Groups or add new Node Groups.

If cluster fails to scale properly, all changes will be rolled back.

2.6.2 Swift Integration

In order to leverage Swift within Hadoop, including using Swift data sources from within EDP, Hadoop requires the application of a patch. For additional information about this patch and configuration, please refer to *Swift Integration*. Sahara automatically sets information about the Swift filesystem implementation, location awareness, URL and tenant name for authorization.

The only required information that is still needed to be set is username and password to access Swift. These parameters need to be explicitly set prior to launching the job.

E.g. :

```
$ hadoop distcp -D fs.swift.service.sahara.username=admin \
 -D fs.swift.service.sahara.password=swordfish \
 swift://integration.sahara/temp swift://integration.sahara/temp1
```

How to compose a swift URL? The template is: `swift://${container}.${provider}/${object}`. We don't need to point out the account because it will be automatically determined from tenant name from configs. Actually, account=tenant.

${provider} was designed to provide an opportunity to work with several Swift installations. E.g. it is possible to read data from one Swift installation and write it to another one. But as for now, Sahara automatically generates configs only for one Swift installation with name "sahara".

Currently user can only enable/disable Swift for a Hadoop cluster. But there is a blueprint about making Swift access more configurable: https://blueprints.launchpad.net/sahara/+spec/swift-configuration-through-rest-and-ui

2.6.3 Cinder support

Cinder is a block storage service that can be used as an alternative for an ephemeral drive. Using Cinder volumes increases reliability of data which is important for HDFS service.

User can set how many volumes will be attached to each node in a Node Group and the size of each volume.

All volumes are attached during Cluster creation/scaling operations.

2.6.4 Neutron and Nova Network support

OpenStack Cluster may use Nova Network or Neutron as a networking service. Sahara supports both, but when deployed, a special configuration for networking should be set explicitly. By default Sahara will behave as if Nova Network is used. If OpenStack Cluster uses Neutron, then `use_neutron` option should be set to `True` in Sahara configuration file. In addition, if the OpenStack Cluster supports network namespaces, set the `use_namespaces` option to `True`

```
use_neutron=True
use_namespaces=True
```

Sahara Dashboard should also be configured properly to support Neutron. `SAHARA_USE_NEUTRON` should be set to `True` in OpenStack Dashboard `local_settings.py` configuration file.

```
SAHARA_USE_NEUTRON=True
```

2.6.5 Floating IP Management

Sahara needs to access instances through ssh during a Cluster setup. To establish a connection Sahara may use both: fixed and floating IP of an Instance. By default `use_floating_ips` parameter is set to `True`, so Sahara will use Floating IP of an Instance to connect. In this case, user has two options for how to make all instances get a floating IP:

- Nova Network may be configured to assign floating IPs automatically by setting `auto_assign_floating_ip` to `True` in `nova.conf`
- User may specify a floating IP pool for each Node Group directly.

Note: When using floating IPs for management (use_floating_ip=True) **every** instance in the Cluster should have a floating IP, otherwise Sahara will not be able to work with it.

If use_floating_ips parameter is set to False Sahara will use Instances' fixed IPs for management. In this case the node where Sahara is running should have access to Instances' fixed IP network. When OpenStack uses Neutron for networking, user will be able to choose fixed IP network for all instances in a Cluster.

2.6.6 Anti-affinity

One of the problems in Hadoop running on OpenStack is that there is no ability to control where machine is actually running. We cannot be sure that two new virtual machines are started on different physical machines. As a result, any replication with cluster is not reliable because all replicas may turn up on one physical machine. Anti-affinity feature provides an ability to explicitly tell Sahara to run specified processes on different compute nodes. This is especially useful for Hadoop datanode process to make HDFS replicas reliable. The Anti-Affinity feature requires certain scheduler filters to be enabled on Nova. Edit your /etc/nova/nova.conf in the following way:

```
[DEFAULT]

...

scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_default_filters=DifferentHostFilter,SameHostFilter
```

This feature is supported by all plugins out of the box.

2.6.7 Data-locality

It is extremely important for data processing to do locally (on the same rack, openstack compute node or even VM) as much work as possible. Hadoop supports data-locality feature and can schedule jobs to tasktracker nodes that are local for input stream. In this case tasktracker could communicate directly with local data node.

Sahara supports topology configuration for HDFS and Swift data sources.

To enable data-locality set enable_data_locality parameter to True in Sahara configuration file

```
enable_data_locality=True
```

In this case two files with topology must be provided to Sahara. Options compute_topology_file and swift_topology_file parameters control location of files with compute and swift nodes topology descriptions correspondingly.

compute_topology_file should contain mapping between compute nodes and racks in the following format:

```
compute1 /rack1
compute1 /rack2
compute1 /rack2
```

Note that compute node name must be exactly the same as configured in openstack (host column in admin list for instances).

swift_topology_file should contain mapping between swift nodes and racks in the following format:

```
node1 /rack1
node2 /rack2
node3 /rack2
```

Note that swift node must be exactly the same as configures in object.builder swift ring. Also make sure that VMs with tasktracker service has direct access to swift nodes.

Hadoop versions after 1.2.0 support four-layer topology (https://issues.apache.org/jira/browse/HADOOP-8468). To enable this feature set `enable_hypervisor_awareness` option to `True` in Sahara configuration file. In this case Sahara will add compute node ID as a second level of topology for Virtual Machines.

2.6.8 Heat Integration

Sahara may use OpenStack Orchestration engine (aka Heat) to provision nodes for Hadoop cluster. To make Sahara work with Heat the following steps are required:

- Your OpenStack installation must have 'orchestration' service up and running
- Sahara must contain the following configuration parameter in *sahara.conf*:

```
# An engine which will be used to provision infrastructure for Hadoop cluster. (string value)
infrastructure_engine=heat
```

The following features are supported in the new Heat engine:

Feature	Heat engine	Known issues
Vanilla plugin provisioning	Implemented	
HDP plugin provisioning	Implemented	
IDH plugin provisioning	Implemented	
Cluster scaling	Implemented	
Cluster rollback	Implemented	
Volumes attachments	Implemented	https://launchpad.net/bugs/1281534
Hadoop and Swift integration	Not affected	
Anti-affinity	Implemented	https://launchpad.net/bugs/1268610
Floating IP Management	Implemented	
Neutron support	Implemented	
Nova Network support	TBD	https://launchpad.net/bugs/1259176
Elastic Data Processing	Not affected	

2.6.9 Plugin Capabilities

The below tables provides a plugin capability matrix:

Feature	Plugin		
Vanilla	HDP	IDH	
Nova and Neutron network	x	x	x
Cluster Scaling	x	Scale Up	x
Swift Integration	x	x	x
Cinder Support	x	x	x
Data Locality	x	x	N/A
EDP	x	x	x

2.7 Registering an Image

Sahara deploys cluster of machines based on images stored in Glance. Each plugin has its own requirements on image contents, see specific plugin documentation for details. A general requirement for an image is to have cloud-init package installed.

Sahara requires image to be registered in Sahara Image Registry order to work with it. A registered image must have two properties set:

- username - a name of the default cloud-init user.
- tags - certain tags mark image to be suitable for certain plugins.

Username depends on image used. Tags depend on the plugin used You can find both in the respective plugin's documentation.

Plugins

2.8 Provisioning Plugins

This page lists all available provisioning plugins. In general a plugin enables Sahara to deploy a specific Hadoop version/distribution in various topologies and with management/monitoring tools.

- *Vanilla Plugin* - deploys Vanilla Apache Hadoop
- *Hortonworks Data Plaform Plugin* - deploys Hortonworks Data Platform

2.9 Vanilla Plugin

Vanilla plugin is a reference plugin implementation which allows to operate with cluster with Apache Hadoop.

For cluster provisioning prepared images should be used. They already have Apache Hadoop 1.2.1 and Apache Hadoop 2.3.0 installed. Here you can find prepared images:

- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-1.2.1-ubuntu-13.10.qcow2
- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-1.2.1-fedora-20.qcow2
- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-1.2.1-centos-6.5.qcow2
- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-2.3.0-ubuntu-13.10.qcow2
- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-2.3.0-fedora-20.qcow2
- http://sahara-files.mirantis.com/sahara-icehouse-vanilla-2.3.0-centos-6.5.qcow2

Besides, you may build images by yourself using *Building Images for Vanilla Plugin*. Keep in mind that if you want to use "Swift Integration" feature (*Features Overview*), Hadoop 1.2.1 must be patched with implementation of Swift File System. For more information about patching required by "Swift Integration" feature see *Swift Integration*.

Vanilla plugin requires an image to be tagged in Sahara Image Registry with two tags: 'vanilla' and '<hadoop version>' (e.g. '1.2.1').

Also you should specify username of default cloud-user used in the Image:

OS	username
Ubuntu 13.10	ubuntu
Fedora 20	fedora
CentOS 6.5	cloud-user

2.9.1 Cluster Validation

When user creates or scales a Hadoop cluster using a Vanilla plugin, the cluster topology requested by user is verified for consistency.

Currently there are the following limitations in cluster topology for Vanilla plugin:

For Vanilla Hadoop version 1.X.X:

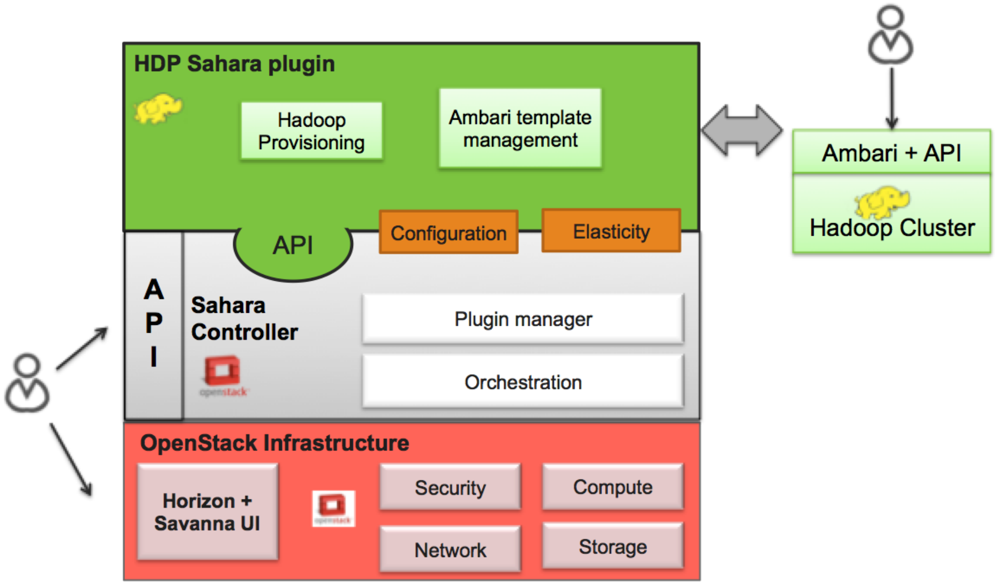
- Cluster must contain exactly one namenode
- Cluster can contain at most one jobtracker
- Cluster can contain at most one oozie and this process is also required for EDP
- Cluster can't contain oozie without jobtraker
- Cluster can't have tasktracker nodes if it doesn't have jobtracker

For Vanilla Hadoop version 2.X.X:

- Cluster must contain exactly one namenode
- Cluster can contain at most one resourcemanager
- Cluster can contain at most one historyserver
- Cluster can contain at most one oozie and this process is also required for EDP
- Cluster can't contain oozie without resourcemanager and without historyserver
- Cluster can't have nodemanager nodes if it doesn't have resourcemanager

2.10 Hortonworks Data Plaform Plugin

The Hortonworks Data Platform (HDP) Sahara plugin provides a way to provision HDP clusters on OpenStack using templates in a single click and in an easily repeatable fashion. As seen from the architecture diagram below, the Sahara controller serves as the glue between Hadoop and OpenStack. The HDP plugin mediates between the Sahara controller and Apache Ambari in order to deploy and configure Hadoop on OpenStack. Core to the HDP Plugin is Apache Ambari which is used as the orchestrator for deploying HDP on OpenStack.



The HDP plugin can make use of Ambari Blueprints for cluster provisioning.

2.10.1 Apache Ambari Blueprints

Apache Ambari Blueprints is a portable document definition, which provides a complete definition for an Apache Hadoop cluster, including cluster topology, components, services and their configurations. Ambari Blueprints can be consumed by the HDP plugin to instantiate a Hadoop cluster on OpenStack. The benefits of this approach is that it allows for Hadoop clusters to be configured and deployed using an Ambari native format that can be used with as well as outside of OpenStack allowing for clusters to be re-instantiated in a variety of environments.

For more information about Apache Ambari Blueprints, refer to: https://issues.apache.org/jira/browse/AMBARI-1783. Note that Apache Ambari Blueprints are not yet finalized.

2.10.2 Operation

The HDP Plugin performs the following four primary functions during cluster creation:

1. Software deployment - the plugin orchestrates the deployment of the required software to the target VMs
2. Services Installation - the Hadoop services configured for the node groups within the cluster are installed on the associated VMs
3. Services Configuration - the plugin merges the default configuration values and user provided configurations for each installed service to the cluster
4. Services Start - the plugin invokes the appropriate APIs to indicate to the Ambari Server that the cluster services should be started

2.10.3 Images

The Sahara HDP plugin can make use of either minimal (operating system only) images or pre-populated HDP images. The base requirement for both is that the image is cloud-init enabled and contains a supported operating system (see http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_hdp1-system-admin-guide/content/sysadminguides_ha_chap2_3.html).

The advantage of a pre-populated image is that provisioning time is reduced, as packages do not need to be downloaded and installed which make up the majority of the time spent in the provisioning cycle. In addition, provisioning large clusters will put a burden on the network as packages for all nodes need to be downloaded from the package repository.

For more information about HDP images, refer to https://github.com/openstack/sahara-image-elements.

There are three VM images provided for use with the HDP Plugin, that can also be built using the tools available in sahara-image-elemnts:

1. centos-6_64-hdp-1.3.qcow2: This image contains most of the requisite packages necessary for HDP deployment. The packages contained herein correspond to the HDP 1.3 release. The operating system is a minimal CentOS 6.5 cloud-init enabled install. This image can only be used to provision HDP 1.3 hadoop clusters.
2. centos-6_64-hdp-2.0.6.qcow2: This image contains most of the requisite packages necessary for HDP deployment. The packages contained herein correspond to the HDP 2.0.6 release. The operating system is a minimal CentOS 6.5 cloud-init enabled install. This image can only be used to provision HDP 2.0.6 hadoop clusters.
3. centos-6-64-hdp-vanilla.qcow2: This image provides only a minimal install of CentOS 6.5 and is cloud-init enabled. This image can be used to provision any versions of HDP supported by Sahara.

HDP plugin requires an image to be tagged in Sahara Image Registry with two tags: ‘hdp’ and ‘<hdp version>’ (e.g. ‘1.3.2’).

Also in the Image Registry you will need to specify username for an image. The username specified should be ‘root’.

2.10.4 Limitations

The HDP plugin currently has the following limitations:

- It is not possible to decrement the number of node-groups or hosts per node group in a Sahara generated cluster.

2.10.5 HDP Version Support

The HDP plugin currently supports HDP 1.3.2 and HDP 2.0.6. Support for future version of HDP will be provided shortly after software is generally available.

2.10.6 Cluster Validation

Prior to Hadoop cluster creation, the HDP plugin will perform the following validation checks to ensure a successful Hadoop deployment:

- Ensure the existence of a NAMENODE process in the cluster
- Ensure the existence of a JOBTRACKER should any TASKTRACKER be deployed to the cluster
- Ensure the deployment of one Ambari Server instance to the cluster
- Ensure that each defined node group had an associated Ambari Agent configured

2.10.7 The HDP Plugin and Sahara Support

For more information, please contact Hortonworks.

2.11 Intel Distribution for Apache Hadoop Plugin

The Intel Distribution for Apache Hadoop (IDH) Sahara plugin provides a way to provision IDH clusters on OpenStack using templates in a single click and in an easily repeatable fashion. The Sahara controller serves as the glue between Hadoop and OpenStack. The IDH plugin mediates between the Sahara controller and Intel Manager in order to deploy and configure Hadoop on OpenStack. Intel Manager is used as the orchestrator for deploying the IDH stack on OpenStack.

For cluster provisioning images supporting cloud init should be used. The only supported operation system for now is Cent OS 6.4. Here you can find the image:

- http://sahara-files.mirantis.com/CentOS-6.4-cloud-init.qcow2

IDH plugin requires an image to be tagged in Sahara Image Registry with two tags: ‘idh’ and ‘<IDH version>’ (e.g. ‘2.5.1’).

Also you should specify a default username of “cloud-user” to be used in the Image.

2.11.1 Limitations

The IDH plugin currently has the following limitations:

- IDH plugin uses requests python library 1.2.1 or later version. It is necessary for connection retries to IDH manager.

- IDH plugin downloads the Intel Manager package from a URL provided in the cluster configuration. A local HTTP mirror should be used in cases where the VMs do not have access to the Internet or have port limitations.
- IDH plugin adds the Intel rpm repository to the yum configuration. The repository URL can be chosen during Sahara cluster configuration. A local mirror should be used in cases where the VMs have no access to the Internet or have port limitations. Refer to the IDH documentation for instructions on how to create a local mirror.
- Hadoop cluster scaling is supported only for datanode and tasktracker (nodemanager for IDH 3.x) processes.

2.11.2 Cluster Validation

When a user creates or scales a Hadoop cluster using the IDH plugin, the cluster topology requested by the user is verified for consistency.

Currently there are the following limitations in cluster topology for IDH plugin:

- **Cluster should contain**
  - exactly one manager
  - exactly one namenode
  - at most one jobtracker for IDH 2.x or resourcemanager for IDH 3.x
  - at most one oozie
- Cluster cannot be created if it contains worker processes without containing corresponding master processes. E.g. it cannot contain tasktracker if there is no jobtracker.

Elastic Data Processing

2.12 Elastic Data Processing (EDP)

2.12.1 Overview

Sahara's Elastic Data Processing facility or *EDP* allows the execution of Hadoop jobs on clusters created from Sahara. EDP supports:

- Hive, Pig, MapReduce, and Java job types
- storage of job binaries in Swift or Sahara's own database
- access to input and output data sources in Swift or HDFS
- configuration of jobs at submission time
- execution of jobs on existing clusters or transient clusters

2.12.2 Interfaces

The EDP features can be used from the Sahara web UI which is described in the *Sahara UI User Guide*.

The EDP features also can be used directly by a client through the *Sahara REST API v1.1 (EDP)*.

2.12.3 EDP Concepts

Sahara EDP uses a collection of simple objects to define and execute Hadoop jobs. These objects are stored in the Sahara database when they are created, allowing them to be reused. This modular approach with database persistence allows code and data to be reused across multiple jobs.

The essential components of a job are:

- executable code to run
- input data to process
- an output data location
- any additional configuration values needed for the job run

These components are supplied through the objects described below.

Job Binaries

A *Job Binary* object stores a URL to a single Pig script, Hive script, or Jar file and any credentials needed to retrieve the file. The file itself may be stored in the Sahara internal database or in Swift.

Files in the Sahara database are stored as raw bytes in a *Job Binary Internal* object. This object's sole purpose is to store a file for later retrieval. No extra credentials need to be supplied for files stored internally.

Sahara requires credentials (username and password) to access files stored in Swift. The Swift service must be running in the same OpenStack installation referenced by Sahara.

There is a configurable limit on the size of a single job binary that may be retrieved by Sahara. This limit is 5MB and may be set with the *job_binary_max_KB* setting in the `sahara.conf` configuration file.

Jobs

A *Job* object specifies the type of the job and lists all of the individual Job Binary objects that are required for execution. An individual Job Binary may be referenced by multiple Jobs. A Job object specifies a main binary and/or supporting libraries depending on its type.

Job type	Main binary	Libraries
`Hive`	required	optional
`Pig`	required	optional
`MapReduce`	not used	required
`Java`	not used	required

Data Sources

A *Data Source* object stores a URL which designates the location of input or output data and any credentials needed to access the location.

Sahara supports data sources in Swift. The Swift service must be running in the same OpenStack installation referenced by Sahara.

Sahara also supports data sources in HDFS. Any HDFS instance running on a Sahara cluster in the same OpenStack installation is accessible without manual configuration. Other instances of HDFS may be used as well provided that the URL is resolvable from the node executing the job.

Job Execution

Job objects must be *launched* or *executed* in order for them to run on the cluster. During job launch, a user specifies execution details including data sources, configuration values, and program arguments. The relevant details will vary by job type. The launch will create a *Job Execution* object in Sahara which is used to monitor and manage the job.

To execute the job, Sahara generates a workflow and submits it to the Oozie server running on the cluster. Familiarity with Oozie is not necessary for using Sahara but it may be beneficial to the user. A link to the Oozie web console can be found in the Sahara web UI in the cluster details.

2.12.4 General Workflow

The general workflow for defining and executing a job in Sahara is essentially the same whether using the web UI or the REST API.

1. Launch a cluster from Sahara if there is not one already available
2. Create all of the Job Binaries needed to run the job, stored in the Sahara database or in Swift
    - When using the REST API and internal storage of job binaries, there is an extra step here to first create the Job Binary Internal objects
    - Once the Job Binary Internal objects are created, Job Binary objects may be created which refer to them by URL
3. Create a Job object which references the Job Binaries created in step 2
4. Create an input Data Source which points to the data you wish to process
5. Create an output Data Source which points to the location for output data

(Steps 4 and 5 do not apply to Java job types. See Additional Details for Java jobs)

6. Create a Job Execution object specifying the cluster and Job object plus relevant data sources, configuration values, and program arguments
    - When using the web UI this is done with the *Launch On Existing Cluster* or *Launch on New Cluster* buttons on the Jobs tab
    - When using the REST API this is done via the */jobs/<job_id>/execute* method

The workflow is simpler when using existing objects. For example, to construct a new job which uses existing binaries and input data a user may only need to perform steps 3, 5, and 6 above. Of course, to repeat the same job multiple times a user would need only step 6.

Specifying Configuration Values, Parameters, and Arguments

Jobs can be configured at launch. The job type determines the kinds of values that may be set:

Job type	Configration Values	Parameters	Arguments
`Hive`	Yes	Yes	No
`Pig`	Yes	Yes	Yes
`MapReduce`	Yes	No	No
`Java`	Yes	No	Yes

- *Configuration values* are key/value pairs. They set options for EDP, Oozie or Hadoop.
    - The EDP configuration values have names beginning with *edp.* and are consumed by Sahara
    - The Oozie and Hadoop configuration values may be read by running jobs

- *Parameters* are key/value pairs. They supply values for the Hive and Pig parameter substitution mechanisms.
- *Arguments* are strings passed to the pig shell or to a Java `main()` method.

These values can be set on the *Configure* tab during job launch through the web UI or through the *job_configs* parameter when using the */jobs/<job_id>/execute* REST method.

In some cases Sahara generates configuration values or parameters automatically. Values set explicitly by the user during launch will override those generated by Sahara.

Generation of Swift Properties for Data Sources

If a job is run with data sources in Swift, Sahara will automatically generate Swift username and password configuration values based on the credentials in the data sources. If the input and output data sources are both in Swift, it is expected that they specify the same credentials.

The Swift credentials can be set explicitly with the following configuration values:

Name
fs.swift.service.sahara.username
fs.swift.service.sahara.password

Additional Details for Hive jobs

Sahara will automatically generate values for the `INPUT` and `OUTPUT` parameters required by Hive based on the specified data sources.

Additional Details for Pig jobs

Sahara will automatically generate values for the `INPUT` and `OUTPUT` parameters required by Pig based on the specified data sources.

For Pig jobs, `arguments` should be thought of as command line arguments separated by spaces and passed to the `pig` shell.

`Parameters` are a shorthand and are actually translated to the arguments `-param name=value`

Additional Details for MapReduce jobs

Important!

If the job type is MapReduce, the mapper and reducer classes *must* be specified as configuration values:

Name	Example Value
mapred.mapper.class	org.apache.oozie.example.SampleMapper
mapred.reducer.class	org.apache.oozie.example.SampleReducer

Additional Details for Java jobs

Java jobs use two configuration values that do not apply to other job types:

- `edp.java.main_class` (required) Specifies the class containing `main(String[] args)`
- `edp.java.java_opts` (optional) Specifies configuration values for the JVM

A Java job will execute the `main(String[] args)` method of the specified main class. There are two methods of passing values to the `main` method:

- Passing values as arguments

  Arguments set during job launch will be passed in the `String[] args` array.

- Setting configuration values

  Any configuration values that are set can be read from a special file created by Oozie.

Data Source objects are not used with Java job types. Instead, any input or output paths must be passed to the `main` method using one of the above two methods. Furthermore, if Swift data sources are used the configuration values listed in Generation of Swift Properties for Data Sources must be passed with one of the above two methods and set in the configuration by `main`.

The `edp-wordcount` example bundled with Sahara shows how to use configuration values, arguments, and Swift data paths in a Java job type.

2.12.5 Special Sahara URLs

Sahara uses custom URLs to refer to objects stored in Swift or the Sahara internal database. These URLs are not meant to be used outside of Sahara.

Sahara Swift URLs have the form:

```
swift://container.sahara/object
```

Sahara internal database URLs have the form:

```
internal-db://sahara-generated-uuid
```

2.13 EDP Requirements

The OpenStack installation and the cluster launched from Sahara must meet the following minimum requirements in order for EDP to function:

2.13.1 OpenStack Services

When a job is executed, binaries are first uploaded to a job tracker and then moved from the job tracker's local filesystem to HDFS. Therefore, there must be an instance of HDFS available to the nodes in the Sahara cluster.

If the Swift service *is not* running in the OpenStack installation

- Job binaries may only be stored in the Sahara internal database
- Data sources require a long-running HDFS

If the Swift service *is* running in the OpenStack installation

- Job binaries may be stored in Swift or the Sahara internal database
- Data sources may be in Swift or a long-running HDFS

2.13.2 Cluster Processes

At a minimum the Sahara cluster must run a single instance of these processes to support EDP:

- For Hadoop version 1:
  - jobtracker
  - namenode
  - oozie
  - tasktracker
  - datanode
- For Hadoop version 2:
  - namenode
  - datanode
  - resourcemanager
  - nodemanager
  - historyserver
  - oozie

Note, a typical cluster may have more than a single instance of the tasktracker and datanode processes.

2.14 EDP Technical Considerations

There are a several things in EDP which require attention in order to work properly. They are listed on this page.

2.14.1 Transient Clusters

EDP allows running jobs on transient clusters. In this case the cluster is created specifically for the job and is shut down automatically once the job is finished.

Two config parameters control the behaviour of periodic clusters:

- periodic_enable - if set to ‘False’, Sahara will do nothing to a transient cluster once the job it was created for is completed. If it is set to ‘True’, then the behaviour depends on the value of the next parameter.
- use_identity_api_v3 - set it to ‘False’ if your OpenStack installation does not provide Keystone API v3. In that case Sahara will not terminate unneeded clusters. Instead it will set their state to ‘AwaitingTermination’ meaning that they could be manually deleted by a user. If the parameter is set to ‘True’, Sahara will itself terminate the cluster. The limitation is caused by lack of ‘trusts’ feature in Keystone API older than v3.

If both parameters are set to ‘True’, Sahara works with transient clusters in the following manner:

1. When a user requests for a job to be executed on a transient cluster, Sahara creates such a cluster.
2. Sahara drops the user’s credentials once the cluster is created but prior to that it creates a trust allowing it to operate with the cluster instances in the future without user credentials.
3. Once a cluster is not needed, Sahara terminates its instances using the stored trust. Sahara drops the trust after that.

APIs

2.15 Sahara REST API docs

2.15.1 Sahara REST API v1.0

Note: REST API v1.0 corresponds to Sahara v0.2.X

1 General API information

This section contains base info about the Sahara REST API design.

1.1 Authentication and Authorization

The Sahara API uses the Keystone Identity Service as the default authentication service. When Keystone is enabled, users who submit requests to the Sahara service must provide an authentication token in the X-Auth-Token request header. A user can obtain the token by authenticating to the Keystone endpoint. For more information about Keystone, see the OpenStack Identity Developer Guide.

Also with each request a user must specify the OpenStack tenant in the url path, for example: '/v1.0/{tenant_id}/clusters'. Sahara will perform the requested operation in the specified tenant using the provided credentials. Therefore, clusters may be created and managed only within tenants to which the user has access.

1.2 Request / Response Types

The Sahara API supports the JSON data serialization format. This means that for requests that contain a body, the Content-Type header must be set to the MIME type value "application/json". Also, clients should accept JSON serialized responses by specifying the Accept header with the MIME type value "application/json" or adding the ".json" extension to the resource name. The default response format is "application/json" if the client does not specify an Accept header or append the ".json" extension in the URL path.

Example:

```
GET /v1.0/{tenant_id}/clusters.json
```

or

```
GET /v1.0/{tenant_id}/clusters
Accept: application/json
```

1.3 Faults

The Sahara API returns an error response if a failure occurs while processing a request. Sahara uses only standard HTTP error codes. 4xx errors indicate problems in the particular request being sent from the client and 5xx errors indicate server-side problems.

The response body will contain richer information about the cause of the error. An error response follows the format illustrated by the following example:

```
HTTP/1.1 400 BAD REQUEST
Content-type: application/json
Content-length: 126
```

```
{
    "error_name": "CLUSTER_NAME_ALREADY_EXISTS",
    "error_message": "Cluster with name 'test-cluster' already exists",
    "error_code": 400
}
```

The 'error_code' attribute is an HTTP response code. The 'error_name' attribute indicates the generic error type without any concrete ids or names, etc. The last attribute, 'error_message', contains a human readable error description.

2 Plugins

Description

A Plugin object provides information about what Hadoop distribution/version it can install, and what configurations can be set for the cluster.

Plugins ops

Verb	URI	Description
GET	/v1.0/{tenant_id}/plugins	Lists all plugins registered in Sahara.
GET	/v1.0/{tenant_id}/plugins/{plugin_name}	Shows short information about specified plugin.
GET	/v1.0/{tenant_id}/plugins/{plugin_name}/{version}	Shows detailed information for plugin, like node_processes, required_image_tags and configs.
POST	/v1.0/{tenant_id}/plugins/{plugin_name}/{version}/convert-config	Converts file-based cluster config to Cluster Template Object

Examples

2.1 List all Plugins

`GET /v1.0/{tenant_id}/plugins`

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all plugins.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/plugins
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "plugins": [
        {
            "description": "This plugin provides an ability to launch vanilla Apache Hadoop clus
            "versions": [
                "1.2.1"
            ],
            "name": "vanilla",
            "title": "Vanilla Apache Hadoop"
        }
    ]
}
```

2.2 Short Plugin information

GET /v1.0/{tenant_id}/plugins/{plugin_name}

Normal Response Code: 200 (OK)

Errors: none

This operation returns short plugin description.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/plugins/vanilla
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "plugin": {
        "title": "Vanilla Apache Hadoop",
        "description": "This plugin provides an ability to launch vanilla Apache Hadoop cluster
        "name": "vanilla",
        "versions": [
            "1.2.1"
        ]
    }
}
```

2.3 Detailed Plugin information

GET /v1.0/{tenant_id}/plugins/{plugin_name}/{version}

Normal Response Code: 200 (OK)

Errors: none

This operation returns detailed plugin description.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/plugins/vanilla/1.2.1
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "plugin": {
        "node_processes": {
            "HDFS": [
                "namenode",
                "datanode",
                "secondarynamenode"
            ],
            "MapReduce": [
                "tasktracker",
                "jobtracker"
            ]
        },
        "description": "This plugin provides an ability to launch vanilla Apache Hadoop cluster
        "versions": [
            "1.2.1"
        ],
        "required_image_tags": [
            "vanilla",
            "1.2.1"
        ],
        "configs": [
            {
                "default_value": "/tmp/hadoop-${user.name}",
                "name": "hadoop.tmp.dir",
                "priority": 2,
                "config_type": "string",
                "applicable_target": "HDFS",
                "is_optional": true,
                "scope": "node",
                "description": "A base for other temporary directories."
            },
            {
                "default_value": true,
                "name": "hadoop.native.lib",
                "priority": 2,
                "config_type": "bool",
                "applicable_target": "HDFS",
                "is_optional": true,
                "scope": "node",
                "description": "Should native hadoop libraries, if present, be used."
            },
        ],
        "title": "Vanilla Apache Hadoop",
        "name": "vanilla"
    }
}
```

2.4 Convert configuration file

`POST /v1.0/{tenant_id}/plugins/{plugin_name}/{version}/convert-config`

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns short plugin description.

The request body should contain configuration file.

Example: request

```
POST http://sahara/v1.0/775181/plugins/some-plugin/1.1/convert-config
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "cluster_template": {
        "name": "cluster-template",
        "cluster_configs": {
            "HDFS": {},
            "MapReduce": {},
            "general": {}
        },
        "plugin_name": "some-plugin",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
            },
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
            }
        ],
        "hadoop_version": "1.1",
        "id": "c365b7dd-9b11-492d-a119-7ae023c19b51",
        "description": "Converted Cluster Template"
    }
}
```

3 Image Registry

Description

The Image Registry is a tool for managing images. Each plugin provides a list of required tags an image should have. Sahara also requires a username to login into an instance's OS for remote operations execution.

The Image Registry provides an ability to add/remove tags to images and define the OS username.

Image Registry ops

Verb	URI	Description
GET	/v1.0/{tenant_id}/images	Lists all images registered in Image Registry
GET	/v1.0/{tenant_id}/images?tags=tag1&tags=tag2	Lists all images with both tag1 and tag2
GET	/v1.0/{tenant_id}/images/{image_id}	Shows information about specified Image.
POST	/v1.0/{tenant_id}/images/{image_id}	Registers specified Image in Image Registry
DELETE	/v1.0/{tenant_id}/images/{image_id}	Removes specified Image from Image Registry
POST	/v1.0/{tenant_id}/images/{image_id}/tag	Adds tags to specified Image
POST	/v1.0/{tenant_id}/images/{image_id}/untag	Removes tags for specified Image

Examples

3.1 List all Images

`GET /v1.0/{tenant_id}/images`

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all registered images.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/images
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "images": [
        {
            "status": "ACTIVE",
            "username": "ec2-user",
            "name": "fedoraSwift_hadoop_sahara_v02",
            "tags": [
                "vanilla",
                "1.2.1"
            ],
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_tag_vanilla": "True",
                "_sahara_tag_1.2.1": "True",
                "_sahara_username": "ec2-user"
            },
            "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
        }
    ]
}
```

3.2 List Images with specified tags

GET /v1.0/{tenant_id}/images?tags=tag1&tags=tag2

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of images with specified tags.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/images?tags=vanilla
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "images": [
        {
            "status": "ACTIVE",
            "username": "ec2-user",
            "name": "fedoraSwift_hadoop_sahara_v02",
            "tags": [
                "vanilla",
                "1.2.1"
            ],
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_tag_vanilla": "True",
                "_sahara_tag_1.2.1": "True",
                "_sahara_username": "ec2-user"
            },
            "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
        }
    ]
}
```

3.3 Show Image

GET /v1.0/{tenant_id}/images/{image_id}

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about the requested Image.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/images/daa50c37-b11b-4f3d-a586-e5dcd0a4110f
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "image": {
        "status": "ACTIVE",
        "username": "ec2-user",
        "name": "fedoraSwift_hadoop_sahara_v02",
        "tags": [
            "vanilla",
            "1.2.1"
        ],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_tag_vanilla": "True",
            "_sahara_tag_1.2.1": "True",
            "_sahara_username": "ec2-user"
        },
        "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
    }
}
```

3.4 Register Image

`POST /v1.0/{tenant_id}/images/{image_id}`

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns the registered image.

Example: request

```
POST http://sahara/v1.0/775181/images/daa50c37-b11b-4f3d-a586-e5dcd0a4110f

{
    "username": "ec2-user",
    "description": "Fedora image"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "image": {
        "status": "ACTIVE",
        "username": "ec2-user",
        "name": "fedoraSwift_hadoop_sahara_v02",
        "tags": [],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ec2-user",
            "_sahara_description": "Fedora image"
        },
        "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
    }
}
```

3.5 Delete Image

DELETE /v1.0/{tenant_id}/images/{image_id}

Normal Response Code: 204 (NO CONTENT)

Errors: none

Remove an Image from the Image Registry

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara/v1.0/775181/images/daa50c37-b11b-4f3d-a586-e5dcd0a4110f
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

3.6 Add Tags to Image

POST /v1.0/{tenant_id}/images/{image_id}/tag

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns the updated image.

Add Tags to Image.

Example: request

```
POST http://sahara/v1.0/775181/images/daa50c37-b11b-4f3d-a586-e5dcd0a4110f/tag

{
    "tags": ["tag1", "some_other_tag"]
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
```

```
{
    "image": {
        "status": "ACTIVE",
        "username": "ec2-user",
        "name": "fedoraSwift_hadoop_sahara_v02",
        "tags": ["tag1", "some_other_tag"],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ec2-user",
            "_sahara_description": "Fedora image",
            "_sahara_tag_tag1": "True",
            "_sahara_tag_some_other_tag": "True"
        },
        "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
    }
}
```

3.7 Remove Tags from Image

`POST /v1.0/{tenant_id}/images/{image_id}/untag`

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns the updated image.

Removes Tags from Image.

Example: request

```
POST http://sahara/v1.0/775181/images/daa50c37-b11b-4f3d-a586-e5dcd0a4110f/untag

{
    "tags": ["unnecessary_tag"],
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "image": {
        "status": "ACTIVE",
        "username": "ec2-user",
        "name": "fedoraSwift_hadoop_sahara_v02",
        "tags": ["tag1"],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ec2-user",
            "_sahara_description": "Fedora image",
            "_sahara_tag_tag1": "True"
        },
        "id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
        }
    }
```

4 Node Group Templates

Description

A Node Group Template is a template for configuring a group of nodes. A Node Group Template contains a list of processes that will be launched on each node. Also node scoped configurations can be defined in a Node Group Template.

Node Group Templates ops

Verb	URI	Description
GET	/v1.0/{tenant_id}/node-group-templates	Lists all Node Group Templates.
GET	/v1.0/{tenant_id}/node-group-templates/<node_group_template_id>	Shows Information about specified Node Group Template by id
POST	/v1.0/{tenant_id}/node-group-templates	Creates a new Node Group Template.
DELETE	/v1.0/{tenant_id}/node-group-templates/<node_group_template_id>	Deletes an existing Node Group Template by id.

Examples

4.1 List all Node Group Templates

GET /v1.0/{tenant_id}/node-group-templates

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all Node Group Templates.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/node-group-templates
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "node_group_templates": [
        {
            "name": "master",
            "volume_mount_prefix": "/volumes/disk",
            "plugin_name": "vanilla",
            "volumes_size": 10,
            "node_processes": [
                "namenode",
                "jobtracker"
            ],
            "flavor_id": "42",
            "volumes_per_node": 0,
            "node_configs": {
                "HDFS": {},
                "MapReduce": {}
            },
            "hadoop_version": "1.2.1",
            "id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9",
            "description": ""
        },
        {
            "name": "worker",
            "volume_mount_prefix": "/volumes/disk",
            "plugin_name": "vanilla",
            "volumes_size": 10,
            "node_processes": [
                "datanode",
                "tasktracker"
            ],
            "flavor_id": "42",
            "volumes_per_node": 0,
            "node_configs": {
                "HDFS": {},
                "MapReduce": {}
            },
            "hadoop_version": "1.2.1",
            "id": "6bbaba84-d936-4e76-9381-987d3568cf4c",
            "description": ""
        }
    ]
}
```

4.2 Show Node Group Template

GET /v1.0/{tenant_id}/node-group-templates/{node_group_template_id}

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Node Group Template.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/node-group-templates/ea34d320-09d7-4dc1-acbf-75b57cec81c9
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "node_group_template": {
        "name": "master",
        "volume_mount_prefix": "/volumes/disk",
        "plugin_name": "vanilla",
        "volumes_size": 10,
        "node_processes": [
            "namenode",
            "jobtracker"
        ],
        "flavor_id": "42",
        "volumes_per_node": 0,
        "floating_ip_pool": "public",
        "node_configs": {
            "HDFS": {},
            "MapReduce": {}
        },
        "hadoop_version": "1.2.1",
        "id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9",
        "description": ""
    }
}
```

4.3 Create Node Group Template

POST /v1.0/{tenant_id}/node-group-templates

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns created Node Group Template.

Example without configurations: request

```
POST http://sahara/v1.0/775181/node-group-templates

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_processes": [
        "namenode",
        "jobtracker"
    ],
    "name": "master",
    "floating_ip_pool", "public",
    "flavor_id": "42"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "node_group_template": {
        "name": "master",
        "volume_mount_prefix": "/volumes/disk",
        "plugin_name": "vanilla",
        "volumes_size": 10,
        "node_processes": [
            "namenode",
            "jobtracker"
        ],
        "flavor_id": "42",
        "volumes_per_node": 0,
        "floating_ip_pool", "public",
        "node_configs": {},
        "hadoop_version": "1.2.1",
        "id": "ddefda09-9ab9-4555-bf48-e996243af6f2"
    }
}
```

Example with configurations: request

```
POST http://sahara/v1.0/775181/node-group-templates

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_processes": [
        "datanode",
        "tasktracker"
    ],
    "name": "worker",
    "flavor_id": "42",
    "node_configs": {
        "HDFS": {
            "data_node_heap_size": 1024
        },
        "MapReduce": {
            "task_tracker_heap_size": 1024
        }
    }
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "node_group_template": {
        "name": "worker",
        "volume_mount_prefix": "/volumes/disk",
        "plugin_name": "vanilla",
        "volumes_size": 10,
        "node_processes": [
            "datanode",
            "tasktracker"
        ],
        "flavor_id": "42",
        "volumes_per_node": 0,
        "node_configs": {
            "HDFS": {
                "data_node_heap_size": 1024
            },
            "MapReduce": {
                "task_tracker_heap_size": 1024
            }
        },
        "hadoop_version": "1.2.1",
        "id": "060afabe-f4b3-487e-8d48-65c5bb5eb79e"
    }
}
```

4.4 Delete Node Group Template

DELETE /v1.0/{tenant_id}/node-group-templates/{node_group_template_id}

Normal Response Code: 204 (NO CONTENT)

Errors: none

Remove Node Group Template

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara/v1.0/775181/node-group-templates/060afabe-f4b3-487e-8d48-65c5bb5eb79e
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

5 Cluster Templates

Description

A Cluster Template is a template for configuring a Hadoop cluster. A Cluster Template contains a list of node groups with number of instances in each. Also cluster scoped configurations can be defined in a Cluster Template.

Cluster Templates ops

Verb	URI	Description
GET	/v1.0/{tenant_id}/cluster-templates	Lists all Cluster Templates.
GET	/v1.0/{tenant_id}/cluster-templates/<cluster_template_id>	Shows Information about specified Cluster Template by id
POST	/v1.0/{tenant_id}/cluster-templates	Creates a new Cluster Template.
DELETE	/v1.0/{tenant_id}/cluster-templates/<cluster_template_id>	Deletes an existing Cluster Template by id.

Examples

5.1 List all Cluster Templates

GET /v1.0/{tenant_id}/cluster-templates

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all Cluster Templates.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/cluster-templates
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cluster_templates": [
        {
            "name": "cluster-template",
            "cluster_configs": {
                "HDFS": {},
                "MapReduce": {},
                "general": {}
            },
            "plugin_name": "vanilla",
            "anti_affinity": [],
            "node_groups": [
                {
                    "count": 1,
                    "name": "master",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_size": 10,
                    "node_configs": {
                        "HDFS": {},
                        "MapReduce": {}
                    },
                    "flavor_id": "42",
                    "volumes_per_node": 0,
                    "node_processes": [
                        "namenode",
                        "jobtracker"
                    ],
                    "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
                },
                {
                    "count": 3,
                    "name": "worker",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_size": 10,
                    "node_configs": {
                        "HDFS": {},
                        "MapReduce": {}
                    },
                    "flavor_id": "42",
                    "volumes_per_node": 0,
                    "node_processes": [
                        "datanode",
                        "tasktracker"
                    ],
                    "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
                }
            ],
            "hadoop_version": "1.2.1",
            "id": "c365b7dd-9b11-492d-a119-7ae023c19b51",
            "description": ""
        }
    ]
}
```

5.2 Show Cluster Template

GET /v1.0/{tenant_id}/cluster-templates/{cluster_template_id}

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Cluster Template.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/cluster-templates/c365b7dd-9b11-492d-a119-7ae023c19b51
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cluster_template": {
        "name": "cluster-template",
        "cluster_configs": {
            "HDFS": {},
            "MapReduce": {},
            "general": {}
        },
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
            },
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
            }
        ],
        "hadoop_version": "1.2.1",
        "id": "c365b7dd-9b11-492d-a119-7ae023c19b51",
        "description": ""
    }
}
```

5.3 Create Cluster Template

POST /v1.0/{tenant_id}/cluster-templates

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns created Cluster Template.

Example without configurations. Node groups taken from templates: request

```
POST http://sahara/v1.0/775181/cluster-templates

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_groups": [
        {
            "name": "worker",
            "count": 3,
            "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
        },
        {
            "name": "master",
            "count": 1,
            "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
        }
    ],
    "name": "cl-template",
    "neutron_management_network": "e017fdde-a2f7-41ed-b342-2d63083e7772",
    "cluster_configs": {}
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "cluster_template": {
        "name": "cl-template",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
            },
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
            }
        ],
        "neutron_management_network": "e017fdde-a2f7-41ed-b342-2d63083e7772",
        "cluster_configs": {},
        "hadoop_version": "1.2.1",
        "id": "e2ad1d5d-5fff-45e8-8c3c-34697c7cd5ac"
    }
}
```

Example with configurations and no Node Group Templates: request

```
POST http://sahara/v1.0/775181/node-group-templates

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_groups": [
        {
            "name": "master",
            "count": 1,
            "flavor_id": "42",
            "node_processes": [
                "namenode",
                "jobtracker"
            ]
        },
        {
            "name": "worker",
            "count": 3,
            "flavor_id": "42",
            "node_processes": [
                "datanode",
                "tasktracker"
            ]
        }
    ],
    "name": "cl-template2",
    "cluster_configs": {
        "HDFS": {
            "dfs.replication": 2
        }
    },
    "anti_affinity": []
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "cluster_template": {
        "name": "cl-template2",
        "cluster_configs": {
            "HDFS": {
                "dfs.replication": 2
            }
        },
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {},
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ]
            },
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_configs": {},
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ]
            }
        ],
        "hadoop_version": "1.2.1",
        "id": "9d72bc1a-8d38-493e-99f3-ebca4ec99ad8"
    }
}
```

5.4 Delete Cluster Template

DELETE /v1.0/{tenant_id}/cluster-templates/{cluster_template_id}

Normal Response Code: 204 (NO CONTENT)

Errors: none

Remove Cluster Template

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara/v1.0/775181/cluster-templates/9d72bc1a-8d38-493e-99f3-ebca4ec99ad8
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

6 Clusters

Description

A Cluster object represents a Hadoop cluster. A Cluster like a Cluster Template contains a list of node groups with the number of instances in each. Also cluster scoped configurations can be defined in a Cluster Object. Users should provide an OpenStack keypair to get access to cluster nodes via SSH.

Cluster ops

Verb	URI	Description
GET	/v1.0/{tenant_id}/clusters	Lists all Clusters.
GET	/v1.0/{tenant_id}/clusters/<cluster_id>	Shows Information about specified Cluster by id.
POST	/v1.0/{tenant_id}/clusters	Starts a new Cluster.
PUT	/v1.0/{tenant_id}/clusters/<cluster_id>	Scale existing Cluster by adding nodes or Node Groups.
DELETE	/v1.0/{tenant_id}/clusters/<cluster_id>	Terminates an existing Cluster by id.

Examples

6.1 List all Clusters

GET /v1.0/{tenant_id}/clusters

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all Clusters.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/clusters
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "clusters": [
        {
            "status": "Waiting",
            "info": {},
            "name": "doc-cluster",
            "cluster_configs": {
                "HDFS": {},
                "MapReduce": {},
                "general": {}
            },
            "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
            "user_keypair_id": "doc-keypair",
            "plugin_name": "vanilla",
            "anti_affinity": [],
            "node_groups": [
                {
                    "count": 1,
                    "updated": "2013-07-09T09:24:44",
                    "name": "master",
                    "created": "2013-07-09T09:24:44",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_size": 10,
                    "node_processes": [
                        "namenode",
                        "jobtracker"
                    ],
                    "flavor_id": "42",
                    "volumes_per_node": 0,
                    "node_configs": {
                        "HDFS": {},
                        "MapReduce": {}
                    },
                    "instances": [
                        {
                            "instance_name": "doc-cluster-master-001",
                            "instance_id": "b366f88c-bf7d-4371-a046-96179ded4c83",
                            "volumes": []
                        }
                    ],
                    "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
                },
                {
                    "count": 3,
                    "updated": "2013-07-09T09:24:44",
                    "name": "worker",
                    "created": "2013-07-09T09:24:44",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_size": 10,
                    "node_processes": [
                        "datanode",
                        "tasktracker"
                    ],
                    "flavor_id": "42",
                    "volumes_per_node": 0,
                    "node_configs": {
                        "HDFS": {},
                        "MapReduce": {}
                    },
                    "instances": [
                        {
                            "instance_name": "doc-cluster-worker-001",
                            "instance_id": "f9fcd132-0534-4023-b4f6-9e10e2156299",
                            "volumes": []
                        },
                        {
                            "instance_name": "doc-cluster-worker-002",
                            "instance_id": "ce486914-364c-456e-8b0e-322ad178ca9e",
                            "volumes": []
                        },
                        {
                            "instance_name": "doc-cluster-worker-003",
                            "instance_id": "21312b4f-82fd-4840-8ba6-1606c7a2a75a",
                            "volumes": []
                        }
                    ],
                    "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
                }
            ],
            "hadoop_version": "1.2.1",
            "id": "1bb1cced-765e-4a2b-a5b6-ac6bbb0bb798"
        }
    ]
}
```

6.2 Show Cluster

GET /v1.0/{tenant_id}/clusters/{cluster_id}

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Cluster.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.0/775181/clusters/c365b7dd-9b11-492d-a119-7ae023c19b51
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cluster": {
        "status": "Waiting",
        "info": {},
        "name": "doc-cluster",
        "cluster_configs": {
            "HDFS": {},
            "MapReduce": {},
            "general": {}
        },
        "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
        "user_keypair_id": "doc-keypair",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "updated": "2013-07-09T09:24:44",
                "name": "master",
                "created": "2013-07-09T09:24:44",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-master-001",
                        "instance_id": "b366f88c-bf7d-4371-a046-96179ded4c83",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
            },
            {
                "count": 3,
                "updated": "2013-07-09T09:24:44",
                "name": "worker",
                "created": "2013-07-09T09:24:44",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-worker-001",
                        "instance_id": "f9fcd132-0534-4023-b4f6-9e10e2156299",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-002",
                        "instance_id": "ce486914-364c-456e-8b0e-322ad178ca9e",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-003",
                        "instance_id": "21312b4f-82fd-4840-8ba6-1606c7a2a75a",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
            }
        ],
        "hadoop_version": "1.2.1",
        "id": "1bb1cced-765e-4a2b-a5b6-ac6bbb0bb798"
    }
}
```

6.3 Start Cluster

POST /v1.0/{tenant_id}/clusters

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns created Cluster.

Example Cluster creation from template: request

```
POST http://sahara/v1.0/775181/clusters

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "cluster_template_id": "1bb1cced-765e-4a2b-a5b6-ac6bbb0bb798",
    "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
    "user_keypair_id": "doc-keypair",
    "name": "doc-cluster",
    "cluster_configs": {}
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
```

```
{
    "cluster": {
        "status": "Waiting",
        "info": {},
        "name": "doc-cluster",
        "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
        "user_keypair_id": "doc-keypair",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-master-001",
                        "instance_id": "b366f88c-bf7d-4371-a046-96179ded4c83",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
            },
            {
                "count": 3,
                "updated": "2013-07-09T09:24:44",
                "name": "worker",
                "created": "2013-07-09T09:24:44",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-worker-001",
                        "instance_id": "f9fcd132-0534-4023-b4f6-9e10e2156299",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-002",
                        "instance_id": "ce486914-364c-456e-8b0e-322ad178ca9e",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-003",
                        "instance_id": "21312b4f-82fd-4840-8ba6-1606c7a2a75a",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
            }
        ],
        "cluster_configs": {
            "HDFS": {},
            "MapReduce": {},
            "general": {}
        },
        "hadoop_version": "1.2.1",
        "id": "1bb1cced-765e-4a2b-a5b6-ac6bbb0bb798"
    }
}
```

Example Cluster creation from Node Groups: request

```
POST http://sahara/v1.0/775181/clusters

{
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
    "user_keypair_id": "doc-keypair",
    "node_groups": [
        {
            "name": "master",
            "count": 1,
            "flavor_id": "42",
            "node_processes": [
                "namenode",
                "jobtracker"
            ]
        },
        {
            "name": "worker",
            "count": 3,
            "flavor_id": "42",
            "node_processes": [
                "datanode",
                "tasktracker"
            ]
        }
    ],
    "name": "doc-cluster2",
    "cluster_configs": {
        "HDFS": {
            "dfs.replication": 2
        }
    },
    "anti_affinity": []
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "cluster": {
        "status": "Waiting",
        "info": {},
        "name": "doc-cluster2",
        "cluster_configs": {
            "HDFS": {
                "dfs.replication": 2
            },
            "MapReduce": {},
            "general": {}
        },
        "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
        "user_keypair_id": "doc-keypair",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-master-001",
                        "instance_id": "b366f88c-bf7d-4371-a046-96179ded4c83",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "ea34d320-09d7-4dc1-acbf-75b57cec81c9"
            },
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-worker-001",
                        "instance_id": "f9fcd132-0534-4023-b4f6-9e10e2156299",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-002",
                        "instance_id": "ce486914-364c-456e-8b0e-322ad178ca9e",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-003",
                        "instance_id": "21312b4f-82fd-4840-8ba6-1606c7a2a75a",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "6bbaba84-d936-4e76-9381-987d3568cf4c"
            }
        ],
        "hadoop_version": "1.2.1",
        "id": "1bb1cced-765e-4a2b-a5b6-ac6bbb0bb798"
    }
}
```

6.4 Scale Cluster

PUT /v1.0/{tenant_id}/clusters/{cluster_id}

Normal Response Code: 202 (ACCEPTED)

Errors: none

Scale Cluster changing number of nodes in existing Node Groups or adding new Node Groups.

This operation returns updated Cluster.

Example: request

```
PUT http://sahara/v1.0/775181/clusters/9d7g51a-8123-424e-sdsr3-eb222ec989b1

{
    "resize_node_groups": [
        {
            "count": 3,
            "name": "worker"
        }
    ],

    "add_node_groups": [
        {
            "count": 2,
            "name": "big-worker",
            "node_group_template_id": "daa50c37-b11b-4f3d-a586-e5dcd0a4110f"
        }
    ]
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "cluster": {
        "status": "Validating",
        "info": {
            "HDFS": {
                "Web UI": "http://172.18.79.166:50070"
            },
            "MapReduce": {
                "Web UI": "http://172.18.79.166:50030"
            }
        },
        "description": "",
        "cluster_configs": {
            "HDFS": {},
            "MapReduce": {},
            "general": {}
        },
        "default_image_id": "db12c199-d0b5-47d3-8a97-e95eeaeae615",
        "user_keypair_id": "doc-keypair",
        "cluster_template_id": "9426fcb7-4c61-457f-8138-ff3bcf8a55ae",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "namenode",
                    "jobtracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-master-001",
                        "internal_ip": "10.155.0.85",
                        "instance_id": "c6ddd972-e9a3-4c3d-a572-ee5f689dbd54",
                        "management_ip": "172.18.79.166",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "e66689e0-4486-4634-ac92-66ac74a86ba6"
            },
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-worker-001",
                        "internal_ip": "10.155.0.86",
                        "instance_id": "4652aec1-0086-41fc-9d52-e0a22497fa36",
                        "management_ip": "172.18.79.165",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-worker-002",
                        "internal_ip": "10.155.0.84",
                        "instance_id": "42609367-20b9-4211-9fbb-bc20348d43e5",
                        "management_ip": "172.18.79.164",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "24ed6654-7160-4705-85f3-9e28310842af"
            },
            {
                "count": 2,
                "name": "big-worker",
                "volume_mount_prefix": "/volumes/disk",
                "volumes_size": 10,
                "node_processes": [
                    "datanode",
                    "tasktracker"
                ],
                "flavor_id": "42",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {},
                    "MapReduce": {}
                },
                "instances": [
                    {
                        "instance_name": "doc-cluster-big-worker-001",
                        "internal_ip": "10.155.0.88",
                        "instance_id": "747ba11f-ccc8-4119-ac46-77161f0bf12c",
                        "management_ip": "172.18.79.169",
                        "volumes": []
                    },
                    {
                        "instance_name": "doc-cluster-big-worker-002",
                        "internal_ip": "10.155.0.89",
                        "instance_id": "2b0431aa-0707-4e9f-96bb-8f4493e6e340",
                        "management_ip": "172.18.79.160",
                        "volumes": []
                    }
                ],
                "node_group_template_id": "24ed6654-7160-4705-85f3-9e28310842af"
            }
        ],
        "hadoop_version": "1.2.1",
        "id": "e8918684-0941-4637-8238-6fc03a9ba043",
        "name": "doc-cluster"
    }
}
```

6.5 Terminate Cluster

`DELETE /v1.0/{tenant_id}/clusters/{cluster_id}`

Normal Response Code: 204 (NO CONTENT)

Errors: none

Terminate existing cluster.

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara/v1.0/775181/clusters/9d7g51a-8123-424e-sdsr3-eb222ec989b1
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

2.15.2 Sahara REST API v1.1 (EDP)

Note: REST API v1.1 corresponds to Sahara v0.3.X and Sahara Icehouse release

1. General information

REST API v1.1 enhances the *Sahara REST API v1.0* and includes all requests from v1.0. REST API V1.1 is *Elastic Data Processing (EDP)* REST API. It covers the majority of new functions related to creating job binaries and job objects on running Hadoop clusters.

2. Data Sources

Description

A Data Source object provides the location of input or output for MapReduce jobs and may reference different types of storage. Sahara doesn't perform any validation checks for data source locations.

Data Source ops

Verb	URI	Description
GET	/v1.1/{tenant_id}/data-sources	Lists all Data Sources
GET	/v1.1/{tenant_id}/data-sources/<data_source_id>	Shows information about specified Data Source by id
POST	/v1.1/{tenant_id}/data-sources	Create a new Data Source
DELETE	/v1.1/{tenant_id}/data-sources/<data_source_id>	Removes specified Data Source

Examples

2.1 List all Data Sources

`GET /v1.1/{tenant_id}/data-sources`

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all created data sources.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/data-sources
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "data_sources": [
        {
            "description": "This is input",
            "url": "swift://container.sahara/text",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-09 12:37:19.295701",
            "updated_at": null,
            "type": "swift",
            "id": "151d0c0c-464f-4724-96a6-4732d0ca62e1",
            "name": "input"
        },
        {
            "description": "This is output",
            "url": "swift://container.sahara/result",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-09 12:37:58.155911",
            "updated_at": null,
            "type": "swift",
            "id": "577e8bd8-b105-46f0-ace7-baee61e0adda",
            "name": "output"
        },
        {
            "description": "This is hdfs input",
            "url": "hdfs://test-master-node:8020/user/hadoop/input",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2014-01-23 12:37:24.720387",
            "updated_at": null,
            "type": "hdfs",
            "id": "63e3d1e6-52d0-4d27-ab8a-f8e236ded200",
            "name": "hdfs_input"
        }
    ]
}
```

2.2 Show Data Source

GET /v1.1/{tenant_id}/data-sources/<data_source_id>

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Data Source.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/data-sources/151d0c0c-464f-4724-96a
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "data_source": {
        "description": "",
        "url": "swift://container.sahara/text",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-09 12:37:19.295701",
        "updated_at": null,
        "type": "swift",
        "id": "151d0c0c-464f-4724-96a6-4732d0ca62e1",
        "name": "input"
    }
}
```

2.3 Create Data Source

POST /v1.1/{tenant_id}/data-sources

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns the created Data Source.

Example: request

```
POST http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/data-sources
```

```
{
    "description": "This is input",
    "url": "swift://container.sahara/text",
    "credentials": {
        "password": "swordfish",
        "user": "admin"
    },
    "type": "swift",
    "name": "text"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "data_source": {
        "description": "This is input",
        "url": "swift://container.sahara/text",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-15 11:15:25.971886",
        "type": "swift",
        "id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
        "name": "text"
    }
}
```

Example:

This example creates an hdfs data source.

request

```
POST http://sahara:8386/v1.1/e262c255a7de4a0ab0434bafd75660cd/data-sources

{
    "description": "This is hdfs input",
    "url": "hdfs://test-master-node:8020/user/hadoop/input",
    "type": "hdfs",
    "name": "hdfs_input"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "data_source": {
        "description": "This is hdfs input",
        "url": "hdfs://test-master-node:8020/user/hadoop/input",
        "tenant_id": "e262c255a7de4a0ab0434bafd75660cd",
        "created_at": "2014-01-23 12:37:24.720387",
        "type": "hdfs",
        "id": "63e3d1e6-52d0-4d27-ab8a-f8e236ded200",
        "name": "hdfs_input"
    }
}
```

2.4 Delete Data Source

DELETE /v1.1/{tenant_id}/data-sources/<data-source-id>

Normal Response Code: 204 (NO CONTENT)

Errors: none

Removes Data Source

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/data-sources/af7dc864-6331-4c30-
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

3 Job Binary Internals

Description

Job Binary Internals are objects for storing job binaries in the Sahara internal database. A Job Binary Internal contains raw data of executable Jar files, Pig or Hive scripts.

Job Binary Internal ops

Verb	URI	Description
GET	/v1.1/{tenant_id}/job-binary-internals	Lists all Job Binary Internals
GET	/v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>	Shows info about specified Job Binary Internal by id
PUT	/v1.1/{tenant_id}/job-binary-internals/<name>	Create a new Job Binary Internal with specified name
DELETE	/v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>	Removes specified Job Binary Internal
GET	/v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>/data	Retrieves data of specified Job Binary Internal

Examples

3.1 List all Job Binary Internals

GET /v1.1/{tenant_id}/job-binary-internals

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all stored Job Binary Internals.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binary-internals
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "binaries": [
        {
            "name": "example.pig",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-15 12:36:59.329034",
            "updated_at": null,
            "datasize": 161,
            "id": "d2498cbf-4589-484a-a814-81436c18beb3"
        },
        {
            "name": "udf.jar",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-15 12:43:52.008620",
            "updated_at": null,
            "datasize": 3745,
            "id": "22f1d87a-23c8-483e-a0dd-cb4a16dde5f9"
        }
    ]
}
```

3.2 Show Job Binary Internal

GET /v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Job Binary Internal.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binary-internals/d2498cbf-4589-
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "job_binary_internal": {
        "name": "example.pig",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-15 12:36:59.329034",
        "updated_at": null,
        "datasize": 161,
        "id": "d2498cbf-4589-484a-a814-81436c18beb3"
    }
}
```

3.3 Create Job Binary Internal

PUT /v1.1/{tenant_id}/job-binary-internals/<name>

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation shows information about the uploaded Job Binary Internal.

The request body should contain raw data (file) or script text.

Example: request

```
PUT http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binary-internals/script.pig
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "job_binary_internal": {
        "name": "script.pig",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-15 13:17:35.994466",
        "updated_at": null,
        "datasize": 160,
        "id": "4833dc4b-8682-4d5b-8a9f-2036b47a0996"
    }
}
```

3.4 Delete Job Binary Internal

DELETE /v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>

Normal Response Code: 204 (NO CONTENT)

Errors: none

Removes Job Binary Internal object from Sahara's db

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binary-internals/4833dc4b-86
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

3.5 Get Job Binary Internal data

GET /v1.1/{tenant_id}/job-binary-internals/<job_binary_internal_id>/data

Normal Response Code: 200 (OK)

Errors: none

Retrieves data of specified Job Binary Internal object.

This operation returns raw data.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binary-internals/4248975-3c82-4
```

response

```
HTTP/1.1 200 OK
Content-Length: 161
Content-Type: text/html; charset=utf-8
```

4. Job Binaries

Description

Job Binaries objects are designed to create links to certain binaries stored either in the Sahara internal database or in Swift.

Job Binaries ops

Verb	URI	Description
GET	/v1.1/{tenant_id}/job-binaries	Lists all Job Binaries
GET	/v1.1/{tenant_id}/job-binaries/<job_binary_id>	Shows info about specified Job Binary by id
POST	/v1.1/{tenant_id}/job-binaries	Create a new Job Binary object
DELETE	/v1.1/{tenant_id}/job-binaries/<job_binary_id>	Removes specified Job Binary
GET	/v1.1/{tenant_id}/job-binaries/<job_binary_id>/data	Retrieves data of specified Job Binary

Examples

4.1 List all Job Binaries

`GET /v1.1/{tenant_id}/job-binaries`

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all created Job Binaries.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binaries
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "binaries": [
        {
            "description": "",
            "url": "internal-db://d2498cbf-4589-484a-a814-81436c18beb3",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-15 12:36:59.375060",
            "updated_at": null,
            "id": "84248975-3c82-4206-a58d-6e7fb3a563fd",
            "name": "example.pig"
        },
        {
            "description": "",
            "url": "internal-db://22f1d87a-23c8-483e-a0dd-cb4a16dde5f9",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-15 12:43:52.265899",
            "updated_at": null,
            "id": "508fc62d-1d58-4412-b603-bdab307bb926",
            "name": "udf.jar"
        },
        {
            "description": "",
            "url": "swift://container/jar-example.jar",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-15 14:25:04.970513",
            "updated_at": null,
            "id": "a716a9cd-9add-4b12-b1b6-cdb71aaef350",
            "name": "jar-example.jar"
        }
    ]
}
```

4.2 Show Job Binary

GET /v1.1/{tenant_id}/job-binaries/<job_binary_id>

Normal Response Code: 200 (OK)

Errors: none

This operation shows information about a specified Job Binary.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binaries/a716a9cd-9add-4b12-b1b
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "job_binary": {
        "description": "",
        "url": "swift://container/jar-example.jar",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-15 14:25:04.970513",
        "updated_at": null,
        "id": "a716a9cd-9add-4b12-b1b6-cdb71aaef350",
        "name": "jar-example.jar"
    }
}
```

4.3 Create Job Binary

POST /v1.1/{tenant_id}/job-binaries

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation shows information about the created Job Binary.

Example: request

```
POST http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binaries

{
    "url": "swift://container/jar-example.jar",
    "name": "jar-example.jar",
    "description": "This is job binary",
    "extra": {
      "password": "swordfish",
      "user": "admin"
    }
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "job_binary": {
        "description": "This is job binary",
        "url": "swift://container/jar-example.jar",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-15 14:49:20.106452",
        "id": "07f86352-ee8a-4b08-b737-d705ded5ff9c",
        "name": "jar-example.jar"
    }
}
```

4.4 Delete Job Binary

DELETE /v1.1/{tenant_id}/job-binaries/<job_binary_id>

Normal Response Code: 204 (NO CONTENT)

Errors: none

Removes Job Binary object

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binaries/07f86352-ee8a-4b08-
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

4.5 Get Job Binary data

GET /v1.1/{tenant_id}/job-binaries/<job_binary_id>/data

Normal Response Code: 200 (OK)

Errors: none

Retrieves data of specified Job Binary object.

This operation returns raw data.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/job-binaries/84248975-3c82-4206-a58
```

response

```
HTTP/1.1 200 OK
Content-Length: 161
Content-Type: text/html; charset=utf-8
```

5. Jobs

Description

Job objects represent Hadoop jobs. A Job object contains lists of all binaries needed for job execution. User should provide data sources and Job parameters to start job execution. A Job may be run on an existing cluster or a new transient cluster may be created for the Job run.

Job ops

Verb	URI	Description
GET	/v1.1/{tenant_id}/jobs	Lists all created Jobs
GET	/v1.1/{tenant_id}/jobs/<job_id>	Shows info about specified Job by id
POST	/v1.1/{tenant_id}/jobs	Create a new Job object
DELETE	/v1.1/{tenant_id}/jobs/<job_id>	Removes specified Job
GET	/v1.1/{tenant_id}/jobs/config-hints/<job_type>	Shows default configuration by specified Job type
POST	/v1.1/{tenant_id}/jobs/<job_id>/execute	Starts Job executing

Examples

5.1 List all Jobs

GET /v1.1/{tenant_id}/jobs

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all created Jobs.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "jobs": [
        {
            "description": "",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-16 11:26:54.109123",
            "mains": [
                {
                    "description": "",
                    "url": "internal-db://d2498cbf-4589-484a-a814-81436c18beb3",
                    "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                    "created_at": "2013-10-15 12:36:59.375060",
                    "updated_at": null,
                    "id": "84248975-3c82-4206-a58d-6e7fb3a563fd",
                    "name": "example.pig"
                }
            ],
            "updated_at": null,
            "libs": [
                {
                    "description": "",
                    "url": "internal-db://22f1d87a-23c8-483e-a0dd-cb4a16dde5f9",
                    "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                    "created_at": "2013-10-15 12:43:52.265899",
                    "updated_at": null,
                    "id": "508fc62d-1d58-4412-b603-bdab307bb926",
                    "name": "udf.jar"
                }
            ],
            "type": "Pig",
            "id": "65afed9c-dad7-4658-9554-b7b4e1ca908f",
            "name": "pig-job"
        },
        {
            "description": "",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "created_at": "2013-10-16 11:29:55.008351",
            "mains": [],
            "updated_at": null,
            "libs": [
                {
                    "description": "This is job binary",
                    "url": "swift://container/jar-example.jar",
                    "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                    "created_at": "2013-10-15 16:03:37.979630",
                    "updated_at": null,
                    "id": "8955b12f-ed32-4152-be39-5b7398c3d04c",
                    "name": "hadoopexamples.jar"
                }
            ],
            "type": "Jar",
            "id": "7600373c-d262-45c6-845f-77f339f3e503",
            "name": "jar-job"
        }
    ]
}
```

5.2 Show Job

GET /v1.1/{tenant_id}/jobs/<job_id>

Normal Response Code: 200 (OK)

Errors: none

This operation returns the information about the specified Job.

This operation does not require a request body.

Example: request

```
GET http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs/7600373c-d262-45c6-845f-77f339
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "job": {
        "description": "",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-16 11:29:55.008351",
        "mains": [],
        "updated_at": null,
        "libs": [
            {
                "description": "This is job binary",
                "url": "swift://container/jar-example.jar",
                "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                "created_at": "2013-10-15 16:03:37.979630",
                "updated_at": null,
                "id": "8955b12f-ed32-4152-be39-5b7398c3d04c",
                "name": "hadoopexamples.jar"
            }
        ],
        "type": "Jar",
        "id": "7600373c-d262-45c6-845f-77f339f3e503",
        "name": "jar-job"
    }
}
```

5.3 Create Job

`POST /v1.1/{tenant_id}/jobs`

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation shows information about the created Job object.

Example: request

```
POST http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs

{
    "description": "This is pig job example",
    "mains": ["84248975-3c82-4206-a58d-6e7fb3a563fd"],
    "libs": ["508fc62d-1d58-4412-b603-bdab307bb926"],
    "type": "Pig",
    "name": "pig-job-example"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "job": {
        "description": "This is pig job example",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-17 09:52:20.957275",
        "mains": [
            {
                "description": "",
                "url": "internal-db://d2498cbf-4589-484a-a814-81436c18beb3",
                "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                "created_at": "2013-10-15 12:36:59.375060",
                "updated_at": null,
                "id": "84248975-3c82-4206-a58d-6e7fb3a563fd",
                "name": "example.pig"
            }
        ],
        "libs": [
            {
                "description": "",
                "url": "internal-db://22f1d87a-23c8-483e-a0dd-cb4a16dde5f9",
                "tenant_id": "11587919cc534bcbb1027a161c82cf58",
                "created_at": "2013-10-15 12:43:52.265899",
                "updated_at": null,
                "id": "508fc62d-1d58-4412-b603-bdab307bb926",
                "name": "udf.jar"
            }
        ],
        "type": "Pig",
        "id": "3cb27eaa-2f88-4c75-ab81-a36e2ab58d4e",
        "name": "pig-job-example"
    }
}
```

5.4 Delete Job

DELETE /v1.1/{tenant_id}/jobs/<job_id>

Normal Response Code: 204 (NO CONTENT)

Errors: none

Removes the Job object

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs/07f86352-ee8a-4b08-b737-d70
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

5.5 Show Job Configuration Hints

GET /v1.1/{tenant_id}/jobs/config-hints/<job-type>

Normal Response Code: 200 (OK)

Errors: none

This operation returns hints for configuration parameters which can be applied during job execution.

This operation does not require a request body.

Note This REST call is used just for hints and doesn't force the user to apply any of them.

Example: request

```
GET http://sahara/v1.1/11587919cc534bcbb1027a161c82cf58/jobs/config-hints/MapReduce
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "job_config": {
        "configs": [
             {
                "name": "mapred.reducer.new-api",
                "value": "true",
                "description": ""
            },
            {
                "name": "mapred.mapper.new-api",
                "value": "true",
                "description": ""
            },
            {
                "name": "mapred.input.dir",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.output.dir",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.mapoutput.key.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.mapoutput.value.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.output.key.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.output.value.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapreduce.map.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapreduce.reduce.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.mapper.class",
                "value": "",
                "description": ""
            },
            {
                "name": "mapred.reducer.class",
                "value": "",
                "description": ""
            }
        ],
        "args": []
    }
}
```

5.6 Execute Job

POST /v1.1/{tenant_id}/jobs/<job_id>/execute

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns the created Job Execution object. Note that different job types support different combinations of `configs`, `args`, and `params`. The *Elastic Data Processing (EDP)* document discusses these differences.

Example execution of a Pig job: request

```
POST http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs/65afed9c-dad7-4658-9554-b7b4e
```

```
{
    "cluster_id": "776e441b-5816-4d47-9e07-7ded58f9a5f6",
    "input_id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
    "output_id": "b63780f3-13d7-4286-b731-88270fb204de",
    "job_configs": {
        "configs": {
            "mapred.map.tasks": "1",
            "mapred.reduce.tasks": "1"
        },
        "args": ["arg1", "arg2"],
        "params": {
            "param2": "value2",
            "param1": "value1"
        }
    }
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "job_execution": {
        "output_id": "b63780f3-13d7-4286-b731-88270fb204de",
        "info": {
            "status": "Pending"
        },
        "job_id": "65afed9c-dad7-4658-9554-b7b4e1ca908f",
        "tenant_id": "11587919cc534bcbb1027a161c82cf58",
        "created_at": "2013-10-17 13:17:03.631362",
        "input_id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
        "cluster_id": "776e441b-5816-4d47-9e07-7ded58f9a5f6",
        "job_configs": {
            "configs": {
                "mapred.map.tasks": "1",
                "mapred.reduce.tasks": "1"
            },
            "args": ["arg1", "arg2"],
            "params": {
                "param2": "value2",
                "param1": "value1"
            }
        },
        "id": "fb2ba667-1162-4f6d-ba77-662c04dfac35"
    }
}
```

Example execution of a Java job:

The main class is specified with `edp.java.main_class`. The input/output paths are passed in `args` because Java jobs do not use data sources. Finally, the swift configs must be specified because the input/output paths are swift paths.

request

```
POST http://sahara:8386/v1.1/11587919cc534bcbb1027a161c82cf58/jobs/65afed9c-dad7-4658-9554-b7b4e

{
    "cluster_id": "776e441b-5816-4d47-9e07-7ded58f9a5f6",
    "job_configs": {
        "configs": {
            "fs.swift.service.sahara.username": "myname",
            "fs.swift.service.sahara.password": "mypassword",
            "edp.java.main_class": "org.apache.hadoop.examples.WordCount"
        },
        "args": ["swift://integration.sahara/demo/make_job.sh", "swift://integration.sahara/frid
    }
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json

{
    "job_execution": {
        "output_id": null,
        "info": {
            "status": "Pending"
        },
        "job_id": "8236b1b4-e1b8-46ef-9174-355cd4234b62",
        "tenant_id": "a4e4599e87e04bf1996862ae295f6f53",
        "created_at": "2014-02-05 23:31:57.752897",
        "input_id": null,
        "cluster_id": "466a2b6d-df00-4310-b985-c106f5231ec0",
        "job_configs": {
            "configs": {
                "edp.java.main_class": "org.apache.hadoop.examples.WordCount",
                "fs.swift.service.sahara.password": "myname",
                "fs.swift.service.sahara.username": "mypassword"
            },
            "args": [
                "swift://integration.sahara/demo/make_job.sh",
                "swift://integration.sahara/friday"
            ]
        },
        "id": "724709bf-2268-46ed-8daf-47898b4630b4"
    }
}
```

6. Job Executions

Description

A Job Execution object represents a Hadoop Job executing on specified cluster. A Job Execution polls the status of a running Job and reports it to the user. Also a user has the ability to cancel a running job.

Job Executions ops

Verb	URI	Description
GET	/v1.1/{tenant_id}/job-executions	Lists all Job Executions
GET	/v1.1/{tenant_id}/job-executions/<job_execution_id>	Shows info about specified Job Execution by id
GET	/v1.1/{tenant_id}/job-executions/<job_execution_id>/refresh-status	Refreshes status and shows info about specified Job by id
GET	/v1.1/{tenant_id}/job-executions/<job_execution_id>/cancel	Cancels specified Job by id
DELETE	/v1.1/{tenant_id}/job-executions/<job_execution_id>	Removes specified Job

Examples

6.1 List all Job Executions

`GET /v1.1/{tenant_id}/job-executions`

Normal Response Code: 200 (OK)

Errors: none

This operation returns the list of all Job Executions.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.1/11587919cc534bcbb1027a161c82cf58/job-executions
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "job_executions": [
        {
            "output_id": "b63780f3-13d7-4286-b731-88270fb204de",
            "info": {
                "status": "RUNNING",
                "externalId": null,
                "run": 0,
                "startTime": "Thu, 17 Oct 2013 13:53:14 GMT",
                "appName": "job-wf",
                "lastModTime": "Thu, 17 Oct 2013 13:53:17 GMT",
                "actions": [
                    {
                        "status": "OK",
                        "retries": 0,
                        "transition": "job-node",
                        "stats": null,
                        "startTime": "Thu, 17 Oct 2013 13:53:14 GMT",
                        "cred": "null",
                        "errorMessage": null,
                        "externalId": "-",
                        "errorCode": null,
                        "consoleUrl": "-",
                        "toString": "Action name[:start:] status[OK]",
                        "externalStatus": "OK",
                        "conf": "",
                        "type": ":START:",
                        "trackerUri": "-",
                        "externalChildIDs": null,
                        "endTime": "Thu, 17 Oct 2013 13:53:15 GMT",
                        "data": null,
                        "id": "0000000-131017135256789-oozie-hado-W@:start:",
                        "name": ":start:"
                    },
                    {
                        "status": "RUNNING",
                        "retries": 0,
                        "transition": null,
                        "stats": null,
                        "startTime": "Thu, 17 Oct 2013 13:53:15 GMT",
                        "cred": "null",
                        "errorMessage": null,
                        "externalId": "job_201310171352_0001",
                        "errorCode": null,
                        "consoleUrl": "http://edp-master-001:50030/jobdetails.jsp?jobid=job_2013
                        "toString": "Action name[job-node] status[RUNNING]",
                        "externalStatus": "RUNNING",
                        "conf": "<pig xmlns=\"uri:oozie:workflow:0.2\">\r\n  <job-tracker>edp-ma
                        "type": "pig",
                        "trackerUri": "edp-master-001:8021",
                        "externalChildIDs": null,
                        "endTime": null,
                        "data": null,
                        "id": "0000000-131017135256789-oozie-hado-W@job-node",
                        "name": "job-node"
                    }
                ],
                "acl": null,
                "consoleUrl": "http://edp-master-001.novalocal:11000/oozie?job=0000000-131017135
                "appPath": "hdfs://edp-master-001:8020/user/hadoop/pig-job/9ceb6469-4d06-474d-99
                "toString": "Workflow id[0000000-131017135256789-oozie-hado-W] status[RUNNING]",
                "user": "hadoop",
                "conf": "<configuration>\r\n  <property>\r\n    <name>user.name</name>\r\n    <v
                "parentId": null,
                "createdTime": "Thu, 17 Oct 2013 13:53:14 GMT",
                "group": null,
                "endTime": null,
                "id": "0000000-131017135256789-oozie-hado-W"
            },
            "job_id": "65afed9c-dad7-4658-9554-b7b4e1ca908f",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "start_time": "2013-10-17T17:53:14",
            "updated_at": "2013-10-17 13:53:32.227919",
            "return_code": null,
            "oozie_job_id": "0000000-131017135256789-oozie-hado-W",
            "input_id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
            "end_time": null,
            "cluster_id": "eb85e8a0-510c-489f-b78e-ad1d29e957c8",
            "id": "e63bdc21-0126-4fd2-90c6-5163d16f31df",
            "progress": null,
            "job_configs": {},
            "created_at": "2013-10-17 13:51:11.671977"
        },
        {
```

```
            "output_id": "b63780f3-13d7-4286-b731-88270fb204de",
            "info": {
                "status": "Pending"
            },
            "job_id": "65afed9c-dad7-4658-9554-b7b4e1ca908f",
            "tenant_id": "11587919cc534bcbb1027a161c82cf58",
            "start_time": null,
            "updated_at": null,
            "return_code": null,
            "oozie_job_id": null,
            "input_id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
            "end_time": null,
            "cluster_id": "eb85e8a0-510c-489f-b78e-ad1d29e957c8",
            "id": "e63bdc21-0126-4fd2-90c6-5163d16f31df",
            "progress": null,
            "job_configs": {},
            "created_at": "2013-10-17 14:37:04.107096"
        }
    ]
}
```

6.2 Show Job Execution

`GET /v1.1/{tenant_id}/job-executions/<job_execution_id>`

Normal Response Code: 200 (OK)

Errors: none

This operation shows the information about a specified Job Execution.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.1/11587919cc534bcbb1027a161c82cf58/job-executions/e63bdc21-0126-4fd2-90c6-5
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Response body contains *Job Execution object*

6.3 Refresh Job Execution status

`GET /v1.1/{tenant_id}/job-executions/<job-execution-id>/refresh-status`

Normal Response Code: 200 (OK)

Errors: none

This operation refreshes the status of the specified Job Execution and shows its information.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.1/11587919cc534bcbb1027a161c82cf58/job-executions/4a911624-1e25-4650-bd1d-3
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Response body contains *Job Execution object*

6.4 Cancel Job Execution

`GET /v1.1/{tenant_id}/job-executions/<job-execution-id>/cancel`

Normal Response Code: 200 (OK)

Errors: none

This operation cancels specified Job Execution.

This operation does not require a request body.

Example: request

```
GET http://sahara/v1.1/11587919cc534bcbb1027a161c82cf58/job-executions/4a911624-1e25-4650-bd1d-3
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Response body contains *Job Execution object* with Job Execution in KILLED state

6.5 Delete Job Execution

`DELETE /v1.1/{tenant_id}/job-executions/<job-execution-id>`

Normal Response Code: 204 (NO CONTENT)

Errors: none

Remove an existing Job Execution.

This operation returns nothing.

This operation does not require a request body.

Example: request

```
DELETE http://sahara/v1.1/job-executions/<job-execution-id>/d7g51a-8123-424e-sdsr3-eb222ec989b1
```

response

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

Job Execution object

The following json response represents a Job Execution object returned from Sahara

```
{
    "output_id": "b63780f3-13d7-4286-b731-88270fb204de",
    "info": {
        "status": "RUNNING",
        "externalId": null,
        "run": 0,
        "startTime": "Thu, 17 Oct 2013 13:53:14 GMT",
        "appName": "job-wf",
        "lastModTime": "Thu, 17 Oct 2013 13:53:17 GMT",
        "actions": [
            {
                "status": "OK",
                "retries": 0,
                "transition": "job-node",
                "stats": null,
                "startTime": "Thu, 17 Oct 2013 13:53:14 GMT",
                "cred": "null",
                "errorMessage": null,
                "externalId": "-",
                "errorCode": null,
                "consoleUrl": "-",
                "toString": "Action name[:start:] status[OK]",
                "externalStatus": "OK",
                "conf": "",
                "type": ":START:",
                "trackerUri": "-",
                "externalChildIDs": null,
                "endTime": "Thu, 17 Oct 2013 13:53:15 GMT",
                "data": null,
                "id": "0000000-131017135256789-oozie-hado-W@:start:",
                "name": ":start:"
            },
            {
                "status": "RUNNING",
                "retries": 0,
                "transition": null,
                "stats": null,
                "startTime": "Thu, 17 Oct 2013 13:53:15 GMT",
                "cred": "null",
                "errorMessage": null,
                "externalId": "job_201310171352_0001",
                "errorCode": null,
                "consoleUrl": "http://edp-master-001:50030/jobdetails.jsp?jobid=job_201310171352_0001
                "toString": "Action name[job-node] status[RUNNING]",
                "externalStatus": "RUNNING",
                "conf": "<pig xmlns=\"uri:oozie:workflow:0.2\">\r\n  <job-tracker>edp-master-001:8021
                "type": "pig",
                "trackerUri": "edp-master-001:8021",
                "externalChildIDs": null,
                "endTime": null,
                "data": null,
                "id": "0000000-131017135256789-oozie-hado-W@job-node",
                "name": "job-node"
            }
        ],
        "acl": null,
        "consoleUrl": "http://edp-master-001.novalocal:11000/oozie?job=0000000-131017135256789-oozie-
        "appPath": "hdfs://edp-master-001:8020/user/hadoop/pig-job/9ceb6469-4d06-474d-995d-76fbc3b8c
        "toString": "Workflow id[0000000-131017135256789-oozie-hado-W] status[RUNNING]",
        "user": "hadoop",
        "conf": "<configuration>\r\n  <property>\r\n    <name>user.name</name>\r\n    <value>hadoop<
        "parentId": null,
        "createdTime": "Thu, 17 Oct 2013 13:53:14 GMT",
        "group": null,
        "endTime": null,
        "id": "0000000-131017135256789-oozie-hado-W"
    },
    "job_id": "65afed9c-dad7-4658-9554-b7b4e1ca908f",
    "tenant_id": "11587919cc534bcbb1027a161c82cf58",
    "start_time": "2013-10-17T17:53:14",
    "updated_at": "2013-10-17 13:53:32.227919",
    "return_code": null,
    "oozie_job_id": "0000000-131017135256789-oozie-hado-W",
    "input_id": "af7dc864-6331-4c30-80f5-63d74b667eaf",
    "end_time": null,
    "cluster_id": "eb85e8a0-510c-489f-b78e-ad1d29e957c8",
    "id": "e63bdc21-0126-4fd2-90c6-5163d16f31df",
    "progress": null,
    "job_configs": {},
    "created_at": "2013-10-17 13:51:11.671977"
}
```

Miscellaneous

2.16 Requirements for Guests

Sahara manages guests of various platforms (for example Ubuntu, Fedora, RHEL, and CentOS) with various versions of the Hadoop ecosystem projects installed. There are common requirements for all guests, and additional requirements based on the plugin that is used for cluster deployment.

2.16.1 Common Requirements

- The operating system must be Linux
- cloud-init must be installed
- ssh-server must be installed
  - if a firewall is active it must allow connections on port 22 to enable ssh

2.16.2 Vanilla Plugin Requirements

If the Vanilla Plugin is used for cluster deployment the guest is required to have

- ssh-client installed
- Java (version >= 6)
- Apache Hadoop installed
- ‘hadoop’ user created

See *Swift Integration* for information on using Swift with your Sahara cluster (for EDP support Swift integration is currently required).

To support EDP, the following components must also be installed on the guest:

- Oozie version 4 or higher
- mysql
- hive

See *Building Images for Vanilla Plugin* for instructions on building images for this plugin.

2.16.3 HDP Plugin

This plugin does not have any additional requirements. Currently, only the CentOS Linux distribution is supported but other distributions will be supported in the future. To speed up provisioning, the HDP packages can be pre-installed on the image used. The packages' versions depend on the HDP version being used.

2.17 Swift Integration

Hadoop and Swift integration is the essential continuation of Hadoop&OpenStack marriage. There were two steps to achieve this:

- **Hadoop side: https://issues.apache.org/jira/browse/HADOOP-8545** This patch is not merged yet and is still being developed, so that's why there is an ability to get the latest-version jar file from CDN: http://sahara-files.mirantis.com/hadoop-swift/hadoop-swift-latest.jar
- **Swift side: https://review.openstack.org/#/c/21015** This patch is merged into Grizzly. If you want to make it work in Folsom see the instructions in the section below.

2.17.1 Swift patching

If you are still using Folsom you need to follow these steps:

- Go to proxy server and find proxy-server.conf file. Go to `[pipeline-main]` section and insert a new filter BEFORE 'authtoken' filter. The name of your new filter is not very important, you will use it only for configuration. E.g. let it be `${list_endpoints}`:

```
[pipeline:main]
pipeline = catch_errors healthcheck cache ratelimit swift3 s3token list_endpoints authtoken keystone
```

  The next thing you need to do here is to add the description of new filter:

```
[filter:list_endpoints]
use = egg:swift#${list_endpoints}
# list_endpoints_path = /endpoints/
```

  `list_endpoints_path` is not mandatory and is "endpoints" by default. This param is used for http-request construction. See details below.

- Go to `entry_points.txt` in egg-info. For swift-1.7.4 it may be found in `/usr/lib/python2.7/dist-packages/swift-1.7.4.egg-info/entry_points.txt`. Add the following description to `[paste.filter_factory]` section:

```
${list_endpoints} = swift.common.middleware.list_endpoints:filter_factory
```

- And the last step: put list_endpoints.py to `/python2.7/dist-packages/swift/common/middleware/`.

2.17.2 Is Swift was patched successfully?

You may check if patching is successful just sending the following http requests:

```
http://${proxy}:8080/endpoints/${account}/${container}/${object}
http://${proxy}:8080/endpoints/${account}/${container}
http://${proxy}:8080/endpoints/${account}
```

You don't need any additional headers here and authorization (see previous section: filter ${list_endpoints} is before 'authtoken' filter). The response will contain ip's of all swift nodes which contains the corresponding object.

2.17.3 Hadoop patching

You may build jar file by yourself choosing the latest patch from https://issues.apache.org/jira/browse/HADOOP-8545. Or you may get the latest one from CDN http://sahara-files.mirantis.com/hadoop-swift/hadoop-swift-latest.jar You need to put this file to hadoop libraries (e.g. /usr/lib/share/hadoop/lib) into each job-tracker and task-tracker node in cluster. The main step in this section is to configure core-site.xml file on each of this node.

2.17.4 Hadoop configurations

All of configs may be rewritten by Hadoop-job or set in `core-site.xml` using this template:

```
<property>
    <name>${name} + ${config}</name>
    <value>${value}</value>
    <description>${not mandatory description}</description>
</property>
```

There are two types of configs here:

1. General. The `${name}` in this case equals to `fs.swift`. Here is the list of `${config}`:

   - `.impl` - Swift FileSystem implementation. The ${value} is `org.apache.hadoop.fs.swift.snative.SwiftNativeFileSystem`
   - `.connect.timeout` - timeout for all connections by default: 15000
   - `.socket.timeout` - how long the connection waits for responses from servers. by default: 60000
   - `.connect.retry.count` - connection retry count for all connections. by default: 3
   - `.connect.throttle.delay` - delay in millis between bulk (delete, rename, copy operations). by default: 0
   - `.blocksize` - blocksize for filesystem. By default: 32Mb
   - `.partsize` - the partition size for uploads. By default: 4608*1024Kb
   - `.requestsize` - request size for reads in KB. By default: 64Kb

2. Provider-specific. Patch for Hadoop supports different cloud providers. The `${name}` in this case equals to `fs.swift.service.${provider}`.

   Here is the list of `${config}`:

   - `.auth.url` - authorization URL
   - `.tenant`
   - `.username`

- `.password`
- `.http.port`
- `.https.port`
- `.region` - Swift region is used when cloud has more than one Swift installation. If region param is not set first region from Keystone endpoint list will be chosen. If region param not found exception will be thrown.
- `.location-aware` - turn On location awareness. Is false by default
- `.apikey`
- `.public`

2.17.5 Example

By this point Swift and Hadoop is ready for use. All configs in hadoop is ok.

In example below provider's name is `sahara`. So let's copy one object to another in one swift container and account. E.g. /dev/integration/temp to /dev/integration/temp1. Will use distcp for this purpose: http://hadoop.apache.org/docs/r0.19.0/distcp.html

How to write swift path? In our case it will look as follows: `swift://integration.sahara/temp`. So the template is: `swift://${container}.${provider}/${object}`. We don't need to point out the account because it will be automatically determined from tenant name from configs. Actually, account=tenant.

Let's run the job:

```
$ hadoop distcp -D fs.swift.service.sahara.username=admin \
 -D fs.swift.service.sahara.password=swordfish \
 swift://integration.sahara/temp swift://integration.sahara/temp1
```

After that just check if temp1 is created.

2.17.6 Limitations

Note: Please note that container name should be a valid URI.

2.18 Building Images for Vanilla Plugin

In this document you will find instruction on how to build Ubuntu, Fedora, and CentOS images with Apache Hadoop versions 1.x.x and 2.x.x.

As of now the vanilla plugin works with images with pre-installed versions of Apache Hadoop. To simplify the task of building such images we use Disk Image Builder.

Disk Image Builder builds disk images using elements. An element is a particular set of code that alters how the image is built, or runs within the chroot to prepare the image.

Elements for building vanilla images are stored in Sahara extra repository

Note: Sahara requires images with cloud-init package installed:

- For Fedora
- For Ubuntu

To create vanilla images follow these steps:

1. Clone repository "https://github.com/openstack/sahara-image-elements" locally.
2. Run the diskimage-create.sh script.

   You can run the script diskimage-create.sh in any directory (for example, in your home directory). By default this script will attempt to create 6 cloud images, 2 each of Ubuntu, Fedora, and CentOS with versions 1 and 2 of Apache Hadoop. This script must be run with root privileges.

   ```
   sudo bash diskimage-create.sh
   ```

   This scripts will update your system and install required packages.

   - kpartx
   - qemu

   Then it will clone the repositories "https://github.com/openstack/diskimage-builder" and "https://github.com/openstack/sa

   - `DIB_HADOOP_VERSION` - version of Hadoop to install
   - `JAVA_DOWNLOAD_URL` - download link for JDK (tarball or bin)
   - `OOZIE_DOWNLOAD_URL` - download link for OOZIE (we have built

   Oozie libs here: http://sahara-files.mirantis.com/oozie-4.0.0.tar.gz

   - `HIVE_VERSION` - version of Hive to install (currently supports only 0.11.0)
   - `ubuntu_image_name`
   - `fedora_image_name`
   - `DIB_IMAGE_SIZE` - parameter that specifies a volume of hard disk of instance. You need to specify it only for Fedora because Fedora doesn't use all available volume
   - `DIB_COMMIT_ID` - latest commit id of diksimage-builder project
   - `SAHARA_ELEMENTS_COMMIT_ID` - latest commit id of sahara-image-elements project

   NOTE: If you don't want to use default values, you should edit this script and set your values of parameters.

   Then it will create a series of cloud images with `hadoop`, `hive`, `oozie`, `mysql`, and `swift_hadoop` elements that install all the necessary packages and configure them. You will find these images in current directory.

For finer control of diskimage-create.sh see the official documentation or run `$ diskimage-create.sh -h`.

CHAPTER 3

Developer Guide

Programming HowTos and Tutorials

3.1 Development Guidelines

3.1.1 Coding Guidelines

For all the code in Sahara we have a rule - it should pass PEP 8.

To check your code against PEP 8 run:

```
$ tox -e pep8
```

Note: For more details on coding guidelines see file HACKING.rst in the root of Sahara repo.

3.1.2 Modification of Upstream Files

We never modify upstream files in Sahara. Any changes in upstream files should be made in the upstream project and then merged back in to Sahara. This includes whitespace changes, comments, and typos. Any change requests containing upstream file modifications are almost certain to receive lots of negative reviews. Be warned.

Examples of upstream files are default xml configuration files used to configure Hadoop, or code imported from the OpenStack Oslo project. The xml files will usually be found in resource directories with an accompanying README file that identifies where the files came from. For example:

```
$ pwd
/home/me/sahara/sahara/plugins/vanilla/v2_3_0/resources

$ ls
core-default.xml     hdfs-default.xml     oozie-default.xml    README.rst
create_oozie_db.sql  mapred-default.xml   post_conf.template   yarn-default.xml
```

3.1.3 Testing Guidelines

Sahara has a suite of tests that are run on all submitted code, and it is recommended that developers execute the tests themselves to catch regressions early. Developers are also expected to keep the test suite up-to-date with any submitted code changes.

Unit tests are located at `sahara/tests`.

Sahara's suite of unit tests can be executed in an isolated environment with Tox. To execute the unit tests run the following from the root of Sahara repo:

```
$ tox -e py27
```

3.1.4 Documentation Guidelines

All Sahara docs are written using Sphinx / RST and located in the main repo in `doc` directory. You can add/edit pages here to update http://docs.openstack.org/developer/sahara site.

The documentation in docstrings should follow the PEP 257 conventions (as mentioned in the PEP 8 guidelines).

More specifically:

1. Triple quotes should be used for all docstrings.
2. If the docstring is simple and fits on one line, then just use one line.
3. For docstrings that take multiple lines, there should be a newline after the opening quotes, and before the closing quotes.
4. Sphinx is used to build documentation, so use the restructured text markup to designate parameters, return values, etc. Documentation on the sphinx specific markup can be found here:

Run the following command to build docs locally.

```
$ tox -e docs
```

After it you can access generated docs in `doc/build/` directory, for example, main page - `doc/build/html/index.html`.

To make docs generation process faster you can use:

```
$ SPHINX_DEBUG=1 tox -e docs
```

or to avoid sahara reinstallation to virtual env each time you want to rebuild docs you can use the following command (it could be executed only after running `tox -e docs` first time):

```
$ SPHINX_DEBUG=1 .tox/docs/bin/python setup.py build_sphinx
```

Note: For more details on documentation guidelines see file HACKING.rst in the root of Sahara repo.

3.2 Setting Up a Development Environment

This page describes how to setup a Sahara development environment by either installing it as a part of DevStack or pointing a local running instance at an external OpenStack. You should be able to debug and test your changes without having to deploy Sahara.

3.2.1 Setup a Local Environment with Sahara inside DevStack

See the main article.

3.2.2 Setup a Local Environment with an external OpenStack

1. Install prerequisites

On OS X Systems:

```
# we actually need pip, which is part of python package
$ brew install python mysql postgresql
$ pip install virtualenv tox
```

On Ubuntu:

```
$ sudo apt-get update
$ sudo apt-get install git-core python-dev python-virtualenv gcc libpq-dev libmysqlclient-dev python-
$ sudo pip install tox
```

On Fedora-based distributions (e.g., Fedora/RHEL/CentOS/Scientific Linux):

```
$ sudo yum install git-core python-devel python-virtualenv gcc python-pip mariadb-devel postgresql-de
$ sudo pip install tox
```

2. Grab the code from GitHub:

```
$ git clone git://github.com/openstack/sahara.git
$ cd sahara
```

3. Prepare virtual environment:

```
$ tools/install_venv
```

4. Create config file from default template:

```
$ cp ./etc/sahara/sahara.conf.sample-basic ./etc/sahara/sahara.conf
```

5. Look through the sahara.conf and change parameters which default values do not suite you. Set `os_auth_host` to the address of OpenStack keystone.

If you are using Neutron instead of Nova Network add `use_neutron = True` to config. If the linux kernel you're utilizing support network namespaces then also specify `use_namespaces = True`.

Note: Config file can be specified for `sahara-api` command using `--config-file` flag.

6. Create database schema:

```
$ tox -evenv -- sahara-db-manage --config-file etc/sahara/sahara.conf upgrade head
```

7. To start Sahara call:

```
$ tox -evenv -- sahara-api --config-file etc/sahara/sahara.conf -d
```

3.2.3 Setup local OpenStack dashboard with Sahara plugin

Sahara UI Dev Environment Setup

This page describes how to setup the Sahara dashboard UI component by either installing it as part of DevStack or installing it in an isolated environment and running from the command line.

Install as a part of DevStack

The easiest way to have a local Sahara UI environment with DevStack is to include the Sahara-Dashboard component in DevStack. This can be accomplished by modifying your DevStack `local.conf` file to enable `sahara-dashboard`. See the DevStack documentation for more information on installing and configuring DevStack.

If you are developing Sahara from an OSX environment you will need to run DevStack on a virtual machine. See Setup VM for DevStack on OSX for more information.

After Sahara-Dashboard installation as a part of DevStack, Horizon will contain a Sahara tab. Sahara-Dashboard source code will be located at `$DEST/sahara-dashboard` which is usually `/opt/stack/sahara-dashboard`.

Isolated Dashboard for Sahara

These installation steps serve two purposes:

1. Setup a dev environment
2. Setup an isolated Dashboard for Sahara

Note The host where you are going to perform installation has to be able to connect to all OpenStack endpoints. You can list all available endpoints using the following command:

```
$ keystone endpoint-list
```

1. Install prerequisites

   ```
   $ sudo apt-get update
   $ sudo apt-get install git-core python-dev gcc python-setuptools python-virtualenv node-less lib
   ```

   On Ubuntu 12.10 and higher you have to install the following lib as well:

   ```
   $ sudo apt-get install nodejs-legacy
   ```

2. Checkout Horizon from git and switch to your version of OpenStack

   Here is an example for the Icehouse release:

   ```
   $ git clone https://github.com/openstack/horizon -b stable/icehouse
   ```

   Then install the virtual environment:

   ```
   $ python tools/install_venv.py
   ```

3. Create a `local_settings.py` file

   ```
   $ cp openstack_dashboard/local/local_settings.py.example openstack_dashboard/local/local_setting
   ```

4. Modify `openstack_dashboard/local/local_settings.py`

Set the proper values for host and url variables:

```
OPENSTACK_HOST = "ip of your controller"
SAHARA_URL = "url for sahara (e.g. "http://localhost:8386/v1.1")"
```

If you are using Neutron instead of Nova-Network:

```
SAHARA_USE_NEUTRON = True
```

If you are using Nova-Network with `auto_assign_floating_ip=False` add the following parameter:

```
AUTO_ASSIGNMENT_ENABLED = False
```

5. Clone sahara-dashboard sources from `https://github.com/openstack/sahara-dashboard.git`

   ```
   $ git clone https://github.com/openstack/sahara-dashboard.git
   ```

6. Export SAHARA_DASHBOARD_HOME environment variable with a path to sahara-dashboard folder

   ```
   $ export SAHARA_DASHBOARD_HOME=$(pwd)/sahara-dashboard
   ```

7. Create a symlink to sahara-dashboard source

   ```
   $ ln -s $SAHARA_DASHBOARD_HOME/saharadashboard .venv/lib/python2.7/site-packages/saharadashboard
   ```

8. Install python-saharaclient into venv

   ```
   $ .venv/bin/pip install python-saharaclient
   ```

9. Modify `openstack_dashboard/settings.py`

   Add sahara to to the Horizon config:

   ```
   HORIZON_CONFIG = {
       'dashboards': ('nova', 'syspanel', 'settings', 'sahara'),
   ```

   and add saharadashboard to the installed apps:

   ```
   INSTALLED_APPS = (
       'saharadashboard',
       ....
   ```

10. Start Horizon

    ```
    $ tools/with_venv.sh python manage.py runserver 0.0.0.0:8080
    ```

    This will start Horizon in debug mode. That means the logs will be written to console and if any exceptions happen, you will see the stack-trace rendered as a web-page.

    Debug mode can be disabled by changing `DEBUG=True` to `False` in `local_settings.py`. In that case Horizon should be started slightly differently, otherwise it will not serve static files:

    ```
    $ tools/with_venv.sh  python manage.py runserver --insecure 0.0.0.0:8080
    ```

    Note It is not recommended to use Horizon in this mode for production.

11. Applying changes

    If you have changed any `*.py` files in `$SAHARA_DASHBOARD_HOME` directory, Horizon will notice that and reload automatically. However changes made to non-python files may not be noticed, so you have to restart Horizon again manually, as described in step 10.

3.2.4 Tips and tricks for dev environment

1. Pip speedup

Add the following lines to ~/.pip/pip.conf

```
[global]
download-cache = /home/<username>/.pip/cache
index-url = <mirror url>
```

Note! The `~/.pip/cache` folder should be created.

2. Git hook for fast checks

Just add the following lines to .git/hooks/pre-commit and do chmod +x for it.

```
#!/bin/sh
# Run fast checks (PEP8 style check and PyFlakes fast static analysis)
tools/run_fast_checks
```

You can added the same check for pre-push, for example, run_tests and run_pylint.

3. Running static analysis (PyLint)

Just run the following command

```
tools/run_pylint
```

3.3 Sahara UI Dev Environment Setup

This page describes how to setup the Sahara dashboard UI component by either installing it as part of DevStack or installing it in an isolated environment and running from the command line.

3.3.1 Install as a part of DevStack

The easiest way to have a local Sahara UI environment with DevStack is to include the Sahara-Dashboard component in DevStack. This can be accomplished by modifying your DevStack `local.conf` file to enable `sahara-dashboard`. See the DevStack documentation for more information on installing and configuring DevStack.

If you are developing Sahara from an OSX environment you will need to run DevStack on a virtual machine. See Setup VM for DevStack on OSX for more information.

After Sahara-Dashboard installation as a part of DevStack, Horizon will contain a Sahara tab. Sahara-Dashboard source code will be located at `$DEST/sahara-dashboard` which is usually `/opt/stack/sahara-dashboard`.

3.3.2 Isolated Dashboard for Sahara

These installation steps serve two purposes:

1. Setup a dev environment
2. Setup an isolated Dashboard for Sahara

Note The host where you are going to perform installation has to be able to connect to all OpenStack endpoints. You can list all available endpoints using the following command:

```
$ keystone endpoint-list
```

1. Install prerequisites

   ```
   $ sudo apt-get update
   $ sudo apt-get install git-core python-dev gcc python-setuptools python-virtualenv node-less lib
   ```

   On Ubuntu 12.10 and higher you have to install the following lib as well:

   ```
   $ sudo apt-get install nodejs-legacy
   ```

2. Checkout Horizon from git and switch to your version of OpenStack

   Here is an example for the Icehouse release:

   ```
   $ git clone https://github.com/openstack/horizon -b stable/icehouse
   ```

   Then install the virtual environment:

   ```
   $ python tools/install_venv.py
   ```

3. Create a `local_settings.py` file

   ```
   $ cp openstack_dashboard/local/local_settings.py.example openstack_dashboard/local/local_setting
   ```

4. Modify `openstack_dashboard/local/local_settings.py`

   Set the proper values for host and url variables:

   ```
   OPENSTACK_HOST = "ip of your controller"
   SAHARA_URL = "url for sahara (e.g. "http://localhost:8386/v1.1")"
   ```

   If you are using Neutron instead of Nova-Network:

   ```
   SAHARA_USE_NEUTRON = True
   ```

   If you are using Nova-Network with `auto_assign_floating_ip=False` add the following parameter:

   ```
   AUTO_ASSIGNMENT_ENABLED = False
   ```

5. Clone sahara-dashboard sources from `https://github.com/openstack/sahara-dashboard.git`

   ```
   $ git clone https://github.com/openstack/sahara-dashboard.git
   ```

6. Export SAHARA_DASHBOARD_HOME environment variable with a path to sahara-dashboard folder

   ```
   $ export SAHARA_DASHBOARD_HOME=$(pwd)/sahara-dashboard
   ```

7. Create a symlink to sahara-dashboard source

   ```
   $ ln -s $SAHARA_DASHBOARD_HOME/saharadashboard .venv/lib/python2.7/site-packages/saharadashboard
   ```

8. Install python-saharaclient into venv

   ```
   $ .venv/bin/pip install python-saharaclient
   ```

9. Modify `openstack_dashboard/settings.py`

   Add sahara to to the Horizon config:

   ```
   HORIZON_CONFIG = {
       'dashboards': ('nova', 'syspanel', 'settings', 'sahara'),
   ```

and add saharadashboard to the installed apps:

```
INSTALLED_APPS = (
    'saharadashboard',
    ....
```

10. Start Horizon

    ```
    $ tools/with_venv.sh python manage.py runserver 0.0.0.0:8080
    ```

    This will start Horizon in debug mode. That means the logs will be written to console and if any exceptions happen, you will see the stack-trace rendered as a web-page.

    Debug mode can be disabled by changing DEBUG=True to False in local_settings.py. In that case Horizon should be started slightly differently, otherwise it will not serve static files:

    ```
    $ tools/with_venv.sh  python manage.py runserver --insecure 0.0.0.0:8080
    ```

    Note It is not recommended to use Horizon in this mode for production.

11. Applying changes

    If you have changed any *.py files in $SAHARA_DASHBOARD_HOME directory, Horizon will notice that and reload automatically. However changes made to non-python files may not be noticed, so you have to restart Horizon again manually, as described in step 10.

3.4 Quickstart guide

This guide will help you to setup vanilla Hadoop cluster using *Sahara REST API v1.0*.

3.4.1 1. Install Sahara

- If you want to hack the code follow *Setting Up a Development Environment*.
- If you just want to install and use Sahara follow *Sahara Installation Guide*.

3.4.2 2. Keystone endpoints setup

To use CLI tools, such as OpenStack's python clients, we should specify environment variables with addresses and credentials. Let's mind that we have keystone at 127.0.0.1:5000 with tenant admin, credentials admin:nova and Sahara API at 127.0.0.1:8386. Here is a list of commands to set env:

```
$ export OS_AUTH_URL=http://127.0.0.1:5000/v2.0/
$ export OS_TENANT_NAME=admin
$ export OS_USERNAME=admin
$ export OS_PASSWORD=nova
```

You can append these lines to the .bashrc and execute source .bashrc. Now you can get authentication token from OpenStack Keystone service.

```
$ keystone token-get
```

If authentication succeed, output will be as follows:

```
+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|  expires  |       2013-07-08T15:21:18Z       |
|     id    | dd92e3cdb4e1462690cd444d6b01b746 |
| tenant_id | 62bd2046841e4e94a87b4a22aa886c13 |
|  user_id  | 720fb87141a14fd0b204f977f5f02512 |
+-----------+----------------------------------+
```

Save `tenant_id` which is obviously your Tenant ID and `id` which is your authentication token (X-Auth-Token):

```
$ export AUTH_TOKEN="dd92e3cdb4e1462690cd444d6b01b746"
$ export TENANT_ID="62bd2046841e4e94a87b4a22aa886c13"
```

3.4.3 3. Upload image to Glance

You can download pre-built images with vanilla Apache Hadoop or build this images yourself:

- Download and install pre-built image with Ubuntu 13.10

```
$ ssh user@hostname
$ wget http://sahara-files.mirantis.com/sahara-icehouse-vanilla-1.2.1-ubuntu-13.10.qcow2
$ glance image-create --name=sahara-icehouse-vanilla-1.2.1-ubuntu-13.10 \
  --disk-format=qcow2 --container-format=bare < ./sahara-icehouse-vanilla-1.2.1-ubuntu-13.10.qcow2
```

- OR with Fedora 20

```
$ ssh user@hostname
$ wget http://sahara-files.mirantis.com/sahara-icehouse-vanilla-1.2.1-fedora-20.qcow2
$ glance image-create --name=sahara-icehouse-vanilla-1.2.1-fedora-20 \
  --disk-format=qcow2 --container-format=bare < ./sahara-icehouse-vanilla-1.2.1-fedora-20.qcow2
```

- OR build image using *Building Images for Vanilla Plugin*.

Save image id. You can get image id from command `glance image-list`:

```
$ glance image-list --name sahara-icehouse-vanilla-1.2.1-ubuntu-13.10
+--------------------------------------+---------------------------------------------+
| ID                                   | Name                                        |
+--------------------------------------+---------------------------------------------+
| 3f9fc974-b484-4756-82a4-bff9e116919b | sahara-icehouse-vanilla-1.2.1-ubuntu-13.10  |
+--------------------------------------+---------------------------------------------+

$ export IMAGE_ID="3f9fc974-b484-4756-82a4-bff9e116919b"
```

3.4.4 4. Register image in Image Registry

- Now we will actually start to interact with Sahara.

```
$ export SAHARA_URL="http://localhost:8386/v1.0/$TENANT_ID"
```

- Install `httpie` REST client

```
$ sudo pip install httpie
```

- Send POST request to Sahara API to register image with username `ubuntu`.

```
$ http POST $SAHARA_URL/images/$IMAGE_ID X-Auth-Token:$AUTH_TOKEN \
 username=ubuntu
```

- Tag the image:

```
$ http $SAHARA_URL/images/$IMAGE_ID/tag X-Auth-Token:$AUTH_TOKEN \
 tags:=’["vanilla", "1.2.1", "ubuntu"]’
```

- Make sure that image is registered correctly:

```
$ http $SAHARA_URL/images X-Auth-Token:$AUTH_TOKEN
```

- Output should look like:

```
{
    "images": [
        {
            "OS-EXT-IMG-SIZE:size": 550744576,
            "created": "2013-07-07T15:18:50Z",
            "description": "None",
            "id": "3f9fc974-b484-4756-82a4-bff9e116919b",
            "metadata": {
                "_sahara_description": "None",
                "_sahara_tag_1.2.1": "True",
                "_sahara_tag_ubuntu": "True",
                "_sahara_tag_vanilla": "True",
                "_sahara_username": "ubuntu"
            },
            "minDisk": 0,
            "minRam": 0,
            "name": "sahara-icehouse-vanilla-1.2.1-ubuntu-13.10",
            "progress": 100,
            "status": "ACTIVE",
            "tags": [
                "vanilla",
                "ubuntu",
                "1.2.1"
            ],
            "updated": "2013-07-07T16:25:19Z",
            "username": "ubuntu"
        }
    ]
}
```

3.4.5 5. Setup NodeGroup templates

Create file with name `ng_master_template_create.json` and fill it with the following content:

```
{
    "name": "test-master-tmpl",
    "flavor_id": "2",
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_processes": ["jobtracker", "namenode"]
}
```

Create file with name `ng_worker_template_create.json` and fill it with the following content:

```
{
    "name": "test-worker-tmpl",
    "flavor_id": "2",
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_processes": ["tasktracker", "datanode"]
}
```

Send POST requests to Sahara API to upload NodeGroup templates:

```
$ http $SAHARA_URL/node-group-templates X-Auth-Token:$AUTH_TOKEN \
 < ng_master_template_create.json

$ http $SAHARA_URL/node-group-templates X-Auth-Token:$AUTH_TOKEN \
 < ng_worker_template_create.json
```

You can list available NodeGroup templates by sending the following request to Sahara API:

```
$ http $SAHARA_URL/node-group-templates X-Auth-Token:$AUTH_TOKEN
```

Output should look like:

```
{
    "node_group_templates": [
        {
            "created": "2013-07-07T18:53:55",
            "flavor_id": "2",
            "hadoop_version": "1.2.1",
            "id": "b38227dc-64fe-42bf-8792-d1456b453ef3",
            "name": "demo-master",
            "node_configs": {},
            "node_processes": [
                "jobtracker",
                "namenode"
            ],
            "plugin_name": "vanilla",
            "updated": "2013-07-07T18:53:55",
            "volume_mount_prefix": "/volumes/disk",
            "volumes_per_node": 0,
            "volumes_size": 10
        },
        {
            "created": "2013-07-07T18:54:00",
            "flavor_id": "2",
            "hadoop_version": "1.2.1",
            "id": "634827b9-6a18-4837-ae15-5371d6ecf02c",
            "name": "demo-worker",
            "node_configs": {},
            "node_processes": [
                "tasktracker",
                "datanode"
            ],
            "plugin_name": "vanilla",
            "updated": "2013-07-07T18:54:00",
            "volume_mount_prefix": "/volumes/disk",
            "volumes_per_node": 0,
            "volumes_size": 10
        }
    ]
}
```

Save id for the master and worker NodeGroup templates. For example:

- Master NodeGroup template id: `b38227dc-64fe-42bf-8792-d1456b453ef3`
- Worker NodeGroup template id: `634827b9-6a18-4837-ae15-5371d6ecf02c`

3.4.6 6. Setup Cluster Template

Create file with name `cluster_template_create.json` and fill it with the following content:

```
{
    "name": "demo-cluster-template",
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "node_groups": [
        {
            "name": "master",
            "node_group_template_id": "b1ac3f04-c67f-445f-b06c-fb722736ccc6",
            "count": 1
        },
        {
            "name": "workers",
            "node_group_template_id": "dbc6147e-4020-4695-8b5d-04f2efa978c5",
            "count": 2
        }
    ]
}
```

Send POST request to Sahara API to upload Cluster template:

```
$ http $SAHARA_URL/cluster-templates X-Auth-Token:$AUTH_TOKEN \
 < cluster_template_create.json
```

Save template id. For example `ce897df2-1610-4caa-bdb8-408ef90561cf`.

3.4.7 7. Create cluster

Create file with name `cluster_create.json` and fill it with the following content:

```
{
    "name": "cluster-1",
    "plugin_name": "vanilla",
    "hadoop_version": "1.2.1",
    "cluster_template_id" : "ce897df2-1610-4caa-bdb8-408ef90561cf",
    "user_keypair_id": "stack",
    "default_image_id": "3f9fc974-b484-4756-82a4-bff9e116919b"
}
```

There is a parameter `user_keypair_id` with value `stack`. You can create your own keypair in in Horizon UI, or using the command line client:

```
nova keypair-add stack --pub-key $PATH_TO_PUBLIC_KEY
```

Send POST request to Sahara API to create and start the cluster:

```
$ http $SAHARA_URL/clusters X-Auth-Token:$AUTH_TOKEN \
 < cluster_create.json
```

Once cluster started, you'll get similar output:

```
{
    "clusters": [
        {
            "anti_affinity": [],
            "cluster_configs": {},
            "cluster_template_id": "ce897df2-1610-4caa-bdb8-408ef90561cf",
            "created": "2013-07-07T19:01:51",
            "default_image_id": "3f9fc974-b484-4756-82a4-bff9e116919b",
            "hadoop_version": "1.2.1",
            "id": "c5e755a2-b3f9-417b-948b-e99ed7fbf1e3",
            "info": {
                "HDFS": {
                    "Web UI": "http://172.24.4.225:50070"
                },
                "MapReduce": {
                    "Web UI": "http://172.24.4.225:50030"
                }
            },
            "name": "cluster-1",
            "node_groups": [
                {
                    "count": 1,
                    "created": "2013-07-07T19:01:51",
                    "flavor_id": "999",
                    "instances": [
                        {
                            "created": "2013-07-07T19:01:51",
                            "instance_id": "4f6dc715-9c65-4d74-bddd-5f1820e6ce02",
                            "instance_name": "cluster-1-master-001",
                            "internal_ip": "10.0.0.5",
                            "management_ip": "172.24.4.225",
                            "updated": "2013-07-07T19:06:07",
                            "volumes": []
                        }
                    ],
                    "name": "master",
                    "node_configs": {},
                    "node_group_template_id": "b38227dc-64fe-42bf-8792-d1456b453ef3",
                    "node_processes": [
                        "jobtracker",
                        "namenode"
                    ],
                    "updated": "2013-07-07T19:01:51",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_per_node": 0,
                    "volumes_size": 10
                },
                {
                    "count": 2,
                    "created": "2013-07-07T19:01:51",
                    "flavor_id": "999",
                    "instances": [
                        {
                            "created": "2013-07-07T19:01:52",
                            "instance_id": "11089dd0-8832-4473-a835-d3dd36bc3d00",
                            "instance_name": "cluster-1-workers-001",
                            "internal_ip": "10.0.0.6",
                            "management_ip": "172.24.4.227",
                            "updated": "2013-07-07T19:06:07",
                            "volumes": []
                        },
                        {
                            "created": "2013-07-07T19:01:52",
                            "instance_id": "d59ee54f-19e6-401b-8662-04a156ba811f",
                            "instance_name": "cluster-1-workers-002",
                            "internal_ip": "10.0.0.7",
                            "management_ip": "172.24.4.226",
                            "updated": "2013-07-07T19:06:07",
                            "volumes": []
                        }
                    ],
                    "name": "workers",
                    "node_configs": {},
                    "node_group_template_id": "634827b9-6a18-4837-ae15-5371d6ecf02c",
                    "node_processes": [
                        "tasktracker",
                        "datanode"
                    ],
                    "updated": "2013-07-07T19:01:51",
                    "volume_mount_prefix": "/volumes/disk",
                    "volumes_per_node": 0,
                    "volumes_size": 10
                }
            ],
            "plugin_name": "vanilla",
            "status": "Active",
            "updated": "2013-07-07T19:06:24",
            "user_keypair_id": "stack"
        }
    ]
}
```

3.4.8 8. Run MapReduce job

To check that your Hadoop installation works correctly:

- Go to NameNode via ssh:

```
$ ssh ubuntu@<namenode_ip>
```

- Switch to hadoop user:

```
$ sudo su hadoop
```

- Go to hadoop home directory and run the simpliest MapReduce example:

```
$ cd /usr/share/hadoop
$ hadoop jar hadoop-examples-1.2.1.jar pi 10 100
```

Congratulations! Now you have Hadoop cluster ready on the OpenStack cloud!

3.5 How to Participate

3.5.1 Getting started

- Create account on Github (if you don't have one)
    - Make sure that your local git is properly configured by executing `git config --list`. If not, configure `user.name`, `user.email`
- Create account on Launchpad (if you don't have one)
- Subscribe to OpenStack general mail-list
- Subscribe to OpenStack development mail-list
- Create OpenStack profile
- Login to OpenStack Gerrit with your Launchpad id
    - Sign OpenStack Individual Contributor License Agreement
    - Make sure that your email is listed in identities
- Subscribe to code-reviews. Go to your settings on http://review.openstack.org
    - Go to `watched projects`
    - Add `openstack/sahara`, `openstack/sahara-dashboard`, `openstack/sahara-extra`, `openstack/python-saharaclient`, `openstack/sahara-image-elements`

3.5.2 How to stay in touch with the community?

- If you have something to discuss use OpenStack development mail-list. Prefix mail subject with `[Sahara]`
- Join `#openstack-sahara` IRC channel on freenode
- Join public weekly meetings on *Thursdays at 18:00 UTC* on `#openstack-meeting-alt` IRC channel

3.5.3 How to send your first patch on review?

- Checkout Sahara code from Github
- Carefully read https://wiki.openstack.org/wiki/Gerrit_Workflow
    - Pay special attention to https://wiki.openstack.org/wiki/Gerrit_Workflow#Committing_Changes
- Apply and commit your changes
- Make sure that your code passes `PEP8` checks and unit-tests. See *Development Guidelines*
- Send your patch on review
- Monitor status of your patch review on https://review.openstack.org/#/

3.6 How to build Oozie

Note: Apache does not make Oozie builds, so it has to be built manually.

3.6.1 Prerequisites

- Maven
- JDK 1.6 (1.7 is not allowed there)
- Downloaded Oozie distribution from Apache mirror
- Downloaded ext-2.2.zip (it is needed for enable Oozie web console)
- All Hadoop jar files (either on hadoop cluster or simply from any repository)

Note: Name of extJS archive should be only `ext-2.2.zip`, there is a check in oozie-setup.sh

To build oozie.tar.gz you should follow the steps below:

- Make package:

```
$ bin/mkdistro.sh -DskipTests
```

- Unpack file distro/target/oozie-x.x.x-distro.tar.gz
- Create `libext` directory in <oozie-path>
- Copy hadoop jars (including hadoop-core, hadoop-client, hadoop-auth) and `ext-2.2.zip` to `libext` directory
- Prepare war for Oozie web console:

```
$ bin/oozie-setup.sh prepare-war
```

Then your Oozie package is ready, pack it to tar.gz:

```
$ tar -czf oozie.tar.gz <oozie-dir>
```

Similar instruction to build oozie.tar.gz you may find there: http://oozie.apache.org/docs/4.0.0/DG_QuickStart.html#Building_Oozie

3.7 Adding Database Migrations

The migrations in `sahara/db/migration/alembic_migrations/versions` contain the changes needed to migrate between Sahara database revisions. A migration occurs by executing a script that details the changes needed to upgrade or downgrade the database. The migration scripts are ordered so that multiple scripts can run sequentially. The scripts are executed by Sahara's migration wrapper which uses the Alembic library to manage the migration. Sahara supports migration from Icehouse or later.

Any code modifications that change the structure of the database require a migration script so that previously existing databases will continue to function when the new code is released. This page gives a brief overview of how to add the migration.

3.7.1 Generate a New Migration Script

New migration scripts can be generated using the `sahara-db-manage` command.

To generate a migration stub to be filled in by the developer:

```
$ sahara-db-manage --config-file /path/to/sahara.conf revision -m "description of revision"
```

To autogenerate a migration script that reflects the current structure of the database:

```
$ sahara-db-manage --config-file /path/to/sahara.conf revision -m "description of revision" --autogen
```

Each of these commands will create a file of the form `revision_description` where `revision` is a string generated by Alembic and `description` is based on the text passed with the `-m` option.

3.7.2 Follow the Sahara Naming Convention

By convention Sahara uses 3-digit revision numbers, and this scheme differs from the strings generated by Alembic. Consequently, it’s necessary to rename the generated script and modify the revision identifiers in the script.

Open the new script and look for the variable `down_revision`. The value should be a 3-digit numeric string, and it identifies the current revision number of the database. Set the `revision` value to the `down_revision` value + 1. For example, the lines:

```
# revision identifiers, used by Alembic.
revision = ’507eb70202af’
down_revision = ’006’
```

will become:

```
# revision identifiers, used by Alembic.
revision = ’007’
down_revision = ’006’
```

Modify any comments in the file to match the changes and rename the file to match the new revision number:

```
$ mv 507eb70202af_my_new_revision.py 007_my_new_revision.py
```

3.7.3 Add Alembic Operations to the Script

The migration script contains two methods, `upgrade()` and `downgrade()`. Fill in these methods with the appropriate Alembic operations to perform upgrades or downgrades. In the above example, an upgrade will move from revision ‘006’ to revision ‘007’ and a downgrade will move from revision ‘007’ to revision ‘006’.

3.7.4 Command Summary for sahara-db-manage

You can upgrade to the latest database version via:

```
$ sahara-db-manage --config-file /path/to/sahara.conf upgrade head
```

To check the current database version:

```
$ sahara-db-manage --config-file /path/to/sahara.conf current
```

To create a script to run the migration offline:

```
$ sahara-db-manage --config-file /path/to/sahara.conf upgrade head --sql
```

To run the offline migration between specific migration versions:

```
$ sahara-db-manage --config-file /path/to/sahara.conf upgrade <start version>:<end version> --sql
```

Upgrade the database incrementally:

```
$ sahara-db-manage --config-file /path/to/sahara.conf upgrade --delta <# of revs>
```

Downgrade the database by a certain number of revisions:

```
$ sahara-db-manage --config-file /path/to/sahara.conf downgrade --delta <# of revs>
```

Create new revision:

```
$ sahara-db-manage --config-file /path/to/sahara.conf revision -m "description of revision" --autoger
```

Create a blank file:

```
$ sahara-db-manage --config-file /path/to/sahara.conf revision -m "description of revision"
```

This command does not perform any migrations, it only sets the revision. Revision may be any existing revision. Use this command carefully:

```
$ sahara-db-manage --config-file /path/to/sahara.conf stamp <revision>
```

To verify that the timeline does branch, you can run this command:

```
$ sahara-db-manage --config-file /path/to/sahara.conf check_migration
```

If the migration path does branch, you can find the branch point via:

```
$ sahara-db-manage --config-file /path/to/sahara.conf history
```

3.8 Sahara Testing

We have a bunch of different tests for Sahara.

3.8.1 Unit Tests

In most Sahara sub repositories we have *_package_/tests/unit* or *_package_/tests* that contains Python unit tests.

3.8.2 Integration tests

We have integration tests for the main Sahara service and they are located in *sahara/tests/integration*. The main purpose of these integration tests is to run some kind of scenarios to test Sahara using all plugins. You can find more info about it in *sahara/tests/integration/README.rst*.

3.8.3 Tempest tests

We have some tests in Tempest (https://github.com/openstack/tempest) that are testing Sahara. Here is a list of currently implemented tests:

- REST API tests are checking how the Sahara REST API works.

The only part that is not tested is cluster creation, more info about api tests - http://docs.openstack.org/developer/tempest/field_guide/api.html

- CLI tests are checking read-only operations using the Sahara CLI, more info -

http://docs.openstack.org/developer/tempest/field_guide/cli.html

3.8.4 Selenium Integration tests

We have a bunch of Selenium-based UI tests for sahara-dashboard in *saharadashboard/tests*. The following UI parts are covered:

- Clusters
- Cluster templates
- Node group templates
- Image registry
- Data sources
- Job binaries
- Jobs
- Job executions

Background Concepts for Sahara

3.9 Pluggable Provisioning Mechanism

Sahara could be integrated with 3rd party management tools like Apache Ambari and Cloudera Management Console. The integration is achieved using plugin mechanism.

In short, responsibilities are divided between Sahara core and plugin as follows. Sahara interacts with user and provisions infrastructure (VMs). Plugin installs and configures Hadoop cluster on the VMs. Optionally Plugin could deploy management and monitoring tools for the cluster. Sahara provides plugin with utility methods to work with VMs.

A plugin must extend *sahara.plugins.provisioning:ProvisioningPluginBase* class and implement all the required methods. Read *Plugin SPI* for details.

The *instance* objects provided by Sahara have *remote* property which could be used to work with VM. The *remote* is a context manager so you can use it in *with instance.remote:* statements. The list of available commands could be found in *sahara.utils.remote.InstanceInteropHelper*. See Vanilla plugin source for usage examples.

3.10 Plugin SPI

3.10.1 Plugin interface

get_versions()

Returns all versions of Hadoop that could be used with the plugin. It is responsibility of the plugin to make sure that all required images for each hadoop version are available, as well as configs and whatever else that plugin needs to create the Hadoop cluster.

Returns: list of strings - Hadoop versions

Example return value: (“Apache Hadoop 1.1.1”, “CDH 3”, “HDP 1.2”)

get_configs(hadoop_version)

Lists all configs supported by plugin with descriptions, defaults and targets for which this config is applicable.

Returns: list of configs

Example return value: ((“JobTracker heap size”, “JobTracker heap size, in MB”, “int”, “512”, *“mapreduce”*, “node”, True, 1))

get_node_processes(hadoop_version)

Returns all supported services and node processes for a given Hadoop version. Each node process belongs to a single service and that relationship is reflected in the returned dict object. See example for details.

Returns: dictionary having entries (service -> list of processes)

Example return value: {“mapreduce”: [”tasktracker”, “jobtracker”], “hdfs”: [”datanode”, “namenode”]}

get_required_image_tags(hadoop_version)

Lists tags, that should be added to OpenStack Image via Image Registry. Tags are used to filter Images by plugin and hadoop version.

Returns: list of tags

Example return value: [”tag1”, “some_other_tag”, ...]

validate(cluster)

Validates a given cluster object. Raises *SaharaException* with meaningful message.

Returns: None

Example exception: <NotSingleNameNodeException {code=’NOT_SINGLE_NAME_NODE’, message=’Hadoop cluster should contain only 1 NameNode instance. Actual NN count is 2’ }>

validate_edp(cluster)

Validates that given cluster can be used to run EDP jobs. In case of incompatibility raises *SaharaException* with meaningful message.

Returns: None

validate_scaling(cluster, existing, additional)

To be improved.

Validates a given cluster before scaling operation.

Returns: list of validation_errors

update_infra(cluster)

Plugin has a chance to change cluster description here. Specifically, plugin must specify image for VMs could change VMs specs in any way it needs. For instance, plugin can ask for additional VMs for the management tool.

Returns: None

configure_cluster(cluster)

Configures cluster on provisioned by Sahara VMs. In this function plugin should perform all actions like adjusting OS, installing required packages (including Hadoop, if needed), configuring Hadoop, etc.

Returns: None

start_cluster(cluster)

Start already configured cluster. This method is guaranteed to be called only on cluster which was already prepared with configure_cluster(...) call.

Returns: None

scale_cluster(cluster, instances)

Scale an existing Cluster with additional instances. Instances argument is a list of ready-to-configure instances. Plugin should do all configuration operations in this method and start all services on those instances.

Returns: None

decommission_nodes(cluster, instances)

Scale cluster down by removing a list of instances. Plugin should stop services on a provided list of instances. Plugin also may want to update some configurations on other instances, so this method is the right place to do that.

Returns: None

convert(config, plugin_name, version, template_name, cluster_template_create)

Provides plugin with ability to create cluster based on plugin-specific config. Sahara expects plugin to fill in all the required fields. The last argument is the function that plugin should call to save the Cluster Template. See “Cluster Lifecycle for Config File Mode” section below for clarification.

on_terminate_cluster(cluster)

When user terminates cluster, Sahara simply shuts down all the cluster VMs. This method is guaranteed to be invoked before that, allowing plugin to do some clean-up.

Returns: None

get_oozie_server(cluster)

Returns the instance object for the host running the Oozie server (this service may be referenced by a vendor-dependent identifier)

Returns: The Oozie server instance object

get_resource_manager_uri(cluster)

Returns the URI for access to the mapred resource manager (e.g Hadoop 1.x - jobtracker, Hadoop 2.x - yarn resource manager)

Returns: The resource manager URI

3.11 Object Model

Here is a description of all the objects involved in the API.

Notes:

- cluster and node_group have 'extra' field allowing plugin to persist any complementary info about the cluster.
- node_process is just a process that runs at some node in cluster.

Example list of node processes:

1. jobtracker
2. namenode
3. tasktracker
4. datanode

- Each plugin may have different names for the same processes.

3.11.1 Config

An object, describing one configuration entry

Property	Type	Description
name	string	Config name.
description	string	A hint for user, what this config is used for.
config_type	enum	possible values are: 'string', 'integer', 'boolean', 'enum'.
config_values	list	List of possible values, if config_type is enum.
default_value	string	Default value for config.
applicable_target	string	The target could be either a service returned by get_node_processes(...) call in form of 'service:<service name>', or 'general'.
scope	enum	Could be either 'node' or 'cluster'.
is_optional	bool	If is_optional is False and no default_value is specified, user should provide a value.
priority	int	1 or 2. A Hint for UI. Configs with priority *1* are always displayed. Priority *2* means user should click a button to see the config.

3.11.2 User Input

Value provided by user for a specific config.

Property	Type	Description
config	config	A config object for which this user_input is provided.
value	...	Value for the config. Type depends on Config type.

3.11.3 Instance

An instance created for cluster.

Property	Type	Description
instance_id	string	Unique instance identifier.
instance_name	string	OpenStack Instance name.
internal_ip	string	IP to communicate with other instances.
management_ip	string	IP of instance, accessible outside of internal network.
volumes	list	List of volumes attached to instance. Empty if ephemeral drive is used.
nova_info	object	Nova Instance object.
username	string	Username, that Sahara uses for establishing remote connections to instance.
hostname	string	Same as instance_name.
fqdn	string	Fully qualified domain name for this instance.
remote	helpers	Object with helpers for performing remote operations

3.11.4 Node Group

Group of instances.

Property	Type	Description
name	string	Name of this Node Group in Cluster.
flavor_id	string	OpenStack Flavor used to boot instances.
image_id	string	Image id used to boot instances.
node_processes	list	List of processes running on each instance.
node_configs	dict	Configs dictionary, applied to instances.
volumes_per_node	int	Number of volumes mounted to each instance. 0 means use ephemeral drive.
volumes_size	int	Size of each volume (GB).
volumes_mount_prefix	string	Prefix added to mount path of each volume.
floating_ip_pool	string	Floating IP Pool name. All instances in the Node Group will have Floating IPs assigned from this pool.
count	int	Number of instances in this Node Group.
username	string	Username used by Sahara to establish remote connections to instances.
configuration	dict	Merged dictionary of node configurations and cluster configurations.
storage_paths	list	List of directories where storage should be placed.

3.11.5 Cluster

Contains all relevant info about cluster. This object is is provided to the plugin for both cluster creation and scaling. The “Cluster Lifecycle” section below further specifies which fields are filled at which moment.

Property	Type	Description
name	string	Cluster name.
tenant_id	string	OpenStack Tenant id where this Cluster is available.
plugin_name	string	Plugin name.
hadoop_version	string	Hadoop version running on instances.
default_image_id	string	OpenStack image used to boot instances.
node_groups	list	List of Node Groups.
cluster_configs	dict	Dictionary of Cluster scoped configurations.
cluster_template_id	string	Cluster Template used for Node Groups and Configurations.
user_keypair_id	string	OpenStack keypair added to instances to make them accessible for user.
neutron_management_network	string	Neutron network ID. Instances will get fixed IPs in this network if 'use_neutron' config is set to True.
anti_affinity	list	List of processes that will be run on different hosts.
description	string	Cluster Description.
info	dict	Dictionary for additional information.

3.11.6 Validation Error

Describes what is wrong with one of the values provided by user.

Property	Type	Description
config	config	A config object that is not valid.
error_message	string	Message that describes what exactly is wrong.

3.12 Elastic Data Processing (EDP) SPI

3.12.1 Coming soon!

3.13 Sahara Cluster Statuses Overview

All Sahara Cluster operations are performed in multiple steps. A Cluster object has a `Status` attribute which changes when Sahara finishes one step of operations and starts another one.

Sahara supports three types of Cluster operations:

- Create a new Cluster
- Scale/Shrink an existing Cluster
- Delete an existing Cluster

3.13.1 Creating a new Cluster

1. Validating

Before performing any operations with OpenStack environment, Sahara validates user input.

There are two types of validations, that are done:

- Check that a request contains all necessary fields and request does not violate

any constraints like unique naming and etc.

- Plugin check (optional). The provisioning Plugin may also perform any specific checks like Cluster topology validation.

If any of validations fails, the Cluster will still be kept in database with `Error` status.

2. InfraUpdating

This status means that the Provisioning plugin performs some infrastructural updates.

3. Spawning

Sahara sends requests to OpenStack for all resources to be created:

- VMs
- Volumes
- Floating IPs (if Sahara is configured to use Floating IPs)

It takes some time for OpenStack to schedule all required VMs and Volumes, so Sahara wait until all of them are in `Active` state.

4. Waiting

Sahara waits while VMs' operating systems boot up and all internal infrastructure components like networks and volumes are attached and ready to use.

5. Preparing

Sahara preparers a Cluster for starting. This step includes generating `/etc/hosts` file, so that all instances could access each other by a hostname. Also Sahara updates `authorized_keys` file on each VM, so that communications could be done without passwords.

6. Configuring

Sahara pushes service configurations to VMs. Both XML based configurations and environmental variables are set on this step.

7. Starting

Sahara is starting Hadoop services on Cluster's VMs.

8. Active

Active status means that a Cluster has started successfully and is ready to run Jobs.

3.13.2 Scaling/Shrinking an existing Cluster

1. Validating

Sahara checks the scale/shrink request for validity. The Plugin method called for performing Plugin specific checks is different from creation validation method.

2. Scaling

Sahara performs database operations updating all affected existing Node Groups and creating new ones.

3. Adding Instances

State similar to `Spawning` while Custer creation. Sahara adds required amount of VMs to existing Node Groups and creates new Node Groups.

4. Configuring

State similar to `Configuring` while Cluster creation. New instances are being configured in the same manner as already existing ones. Existing Cluster VMs are also updated with a new `/etc/hosts` file.

5. Decommissioning

Sahara stops Hadoop services on VMs that will be deleted from a Cluster. Decommissioning Data Node may take some time because Hadoop rearranges data replicas around the Cluster, so that no data will be lost after tht VM is deleted.

6. Deleting Instances

Sahara sends requests to OpenStack to release unneeded resources:

- VMs
- Volumes
- Floating IPs (if they are used)

7. Active

The same `Active` as after Cluster creation.

3.13.3 Deleting an existing Cluster

1. Deleting

The only step, that releases all Cluster's resources and removes it form database.

3.13.4 Error State

If Cluster creation fails, the Cluster will get into `Error` state. This state means the Cluster may not be able to perform any operations normally. This cluster will stay in database until it is manually deleted. The reason of failure may be found in Sahara logs.

If an error occurs during `Adding Instances` operation, Sahara will first try to rollback this operation. If rollback is impossible or fails itself, then the Cluster will also get into `Error` state.

Other Resources

3.14 Project hosting with Launchpad

Launchpad hosts the Sahara project. The Sahara project homepage on Launchpad is http://launchpad.net/sahara.

3.14.1 Launchpad credentials

Creating a login on Launchpad is important even if you don't use the Launchpad site itself, since Launchpad credentials are used for logging in on several OpenStack-related sites. These sites include:

- Wiki
- Gerrit (see *Code Reviews with Gerrit*)
- Jenkins (see *Continuous Integration with Jenkins*)

3.14.2 Mailing list

The mailing list email is `sahara-all@lists.launchpad.net`. To participate in the mailing list:

1. Join the Sahara Team on Launchpad.
2. Subscribe to the list on the Sahara Team page on Launchpad.

The mailing list archives are at https://lists.launchpad.net/sahara-all

3.14.3 Bug tracking

Report Sahara bugs at https://bugs.launchpad.net/sahara

3.14.4 Feature requests (Blueprints)

Sahara uses Launchpad Blueprints to track feature requests. Blueprints are at https://blueprints.launchpad.net/sahara.

3.14.5 Technical support (Answers)

Sahara uses Launchpad Answers to track Sahara technical support questions. The Sahara Answers page is at https://answers.launchpad.net/sahara

3.15 Code Reviews with Gerrit

Sahara uses the Gerrit tool to review proposed code changes. The review site is http://review.openstack.org.

Gerrit is a complete replacement for Github pull requests. *All Github pull requests to the Sahara repository will be ignored*.

See Gerrit Workflow Quick Reference for information about how to get started using Gerrit. See Gerrit, Jenkins and Github for more detailed documentation on how to work with Gerrit.

3.16 Continuous Integration with Jenkins

Each change made to Sahara core code is tested with unit and integration tests and style checks flake8.

Unit tests and style checks are performed on public OpenStack Jenkins managed by Zuul. Unit tests are checked using both python 2.6 and python 2.7.

The result of those checks and Unit tests are +1 or -1 to *Verify* column in a code review from *Jenkins* user.

Integration tests check CRUD operations for Image Registry, Templates and Clusters. Also a test job is launched on a created Cluster to verify Hadoop work.

All integration tests are launched by Jenkins on internal Mirantis OpenStack Lab. Jenkins keeps a pool of VMs to run tests in parallel. Still integration testing may take a while.

The integration tests result is +1 or -1 to *Verify* column in a code review from *savanna-ci* user.

HTTP Routing Table