Sahara

Release 2014.2.1.dev1.ga7031f6

OpenStack Foundation

October 27, 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:

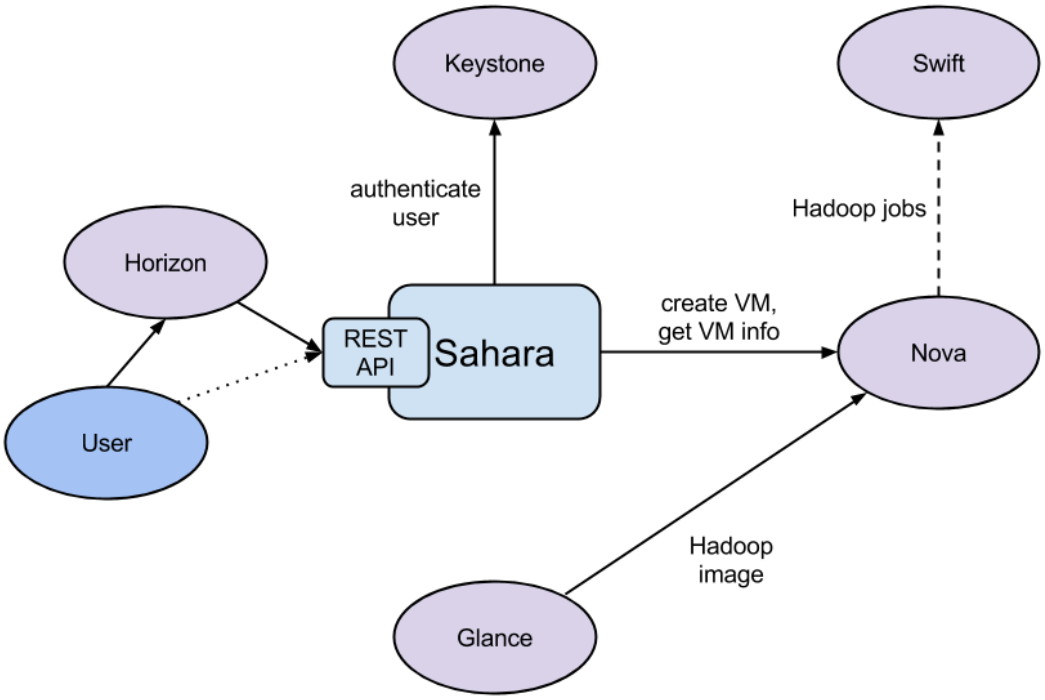
- designed as an OpenStack component;
- managed through REST API with UI available as part of OpenStack Dashboard;
- **support for different Hadoop distributions:**
  - pluggable system of Hadoop installation engines;
  - integration with vendor specific management tools, such as Apache Ambari or Cloudera 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.
- Heat - Sahara can be configured to use Heat; Heat orchestrates the required services 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.
- Cinder - can be used as a block storage.
- Neutron - provides the networking service.
- Ceilometer - used to collect measures of cluster usage for metering and monitoring purposes.



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.

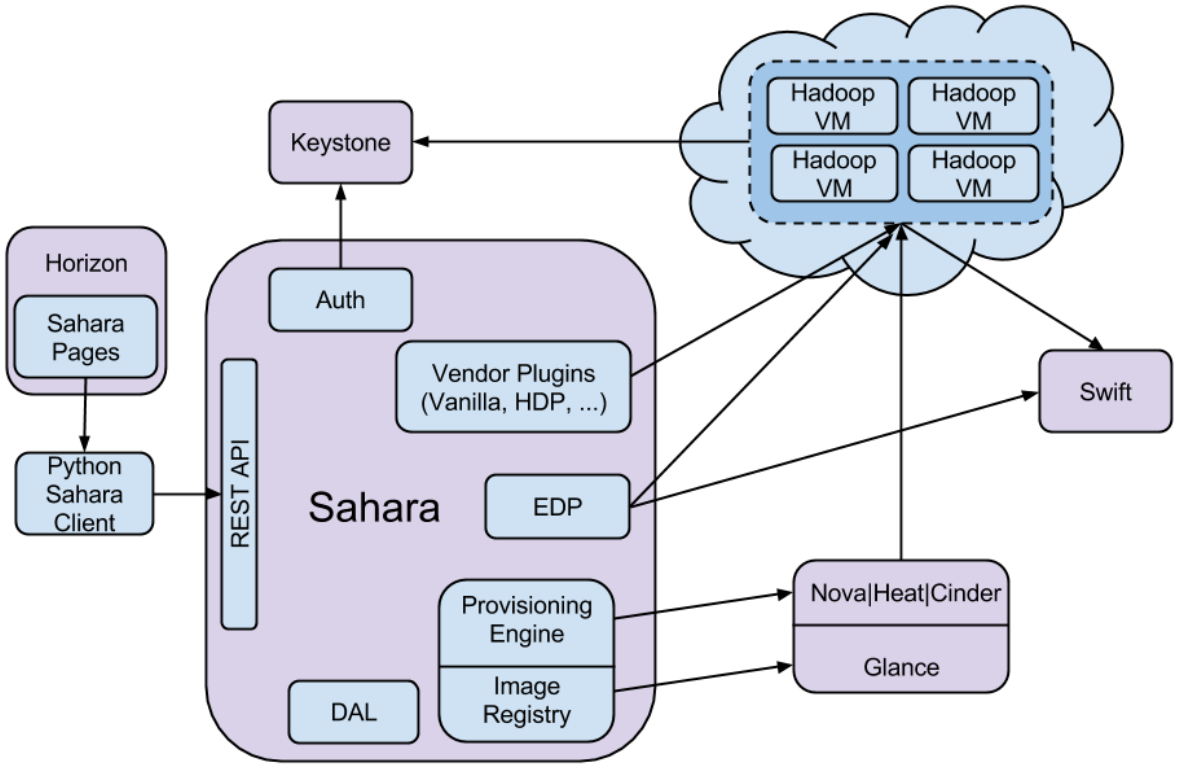
To get more information on how to enable Swift support see *Swift Integration*.

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 MOS Quickstart to install and setup OpenStack.
2. Enable Sahara service during installation.

2.1.2 To install with RDO

1. Start by following the RDO Quickstart to install and setup OpenStack.
2. Install Sahara:

```
# yum install openstack-sahara
```

3. Configure Sahara as needed. 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. Go through *Common installation steps* and make the necessary changes.
6. Start the sahara-all service:

```
# systemctl start openstack-sahara-all
```

7. *(Optional)* Enable Sahara to start on boot

```
# systemctl enable openstack-sahara-all
```

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*

2.1.4 Common installation steps

The steps below are common for both installing Sahara as part of RDO and installing it in virtual environment.

1. 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.

2. Create database schema:

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

3. To start Sahara call:

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

4. In order for Sahara to be accessible in OpenStack Dashboard and for python-saharaclient to work properly you need to register Sahara in Keystone. For example:

```
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"
```

5. To adjust OpenStack Dashboard configuration with your Sahara installation please follow the UI configuration guide `here`.

2.1.5 Notes:

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-all --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. For instance, connection string for mysql database will be:

```
connection=mysql://username:password@host:port/database
```

Switch to the `[keystone_authtoken]` section. The `auth_uri` parameter should point to the public Identity API endpoint. `identity_uri` should point to the admin Identity API endpoint. For example:

```
auth_uri=http://127.0.0.1:5000/v2.0/
identity_uri=http://127.0.0.1:35357/
```

Next specify `admin_user`, `admin_password` and `admin_tenant_name`. These parameters must specify a keystone user which has the `admin` role in the given tenant. These credentials allow Sahara to authenticate and authorize its users.

Switch to the `[DEFAULT]` section. 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.

2.2.1 Sahara notifications configuration

Sahara can send notifications to Ceilometer, if it's enabled. If you want to enable notifications you should switch to `[DEFAULT]` section and set:

```
enable_notifications = true
notification_driver = messaging
```

The current default for Sahara is to use the backend that utilizes RabbitMQ as the message broker. You should configure your backend. It's recommended to use Rabbit or Qpid.

If you are using Rabbit as a backend, then you should set:

```
rpc_backend = rabbit
```

And after that you should specify following options: `rabbit_host`, `rabbit_port`, `rabbit_userid`, `rabbit_password`, `rabbit_virtual_host` and `rabbit_hosts`.

As example you can see default values of these options:

```
rabbit_host=localhost
rabbit_port=5672
rabbit_hosts=$rabbit_host:$rabbit_port
rabbit_userid=guest
rabbit_password=guest
rabbit_virtual_host=/
```

If you are using Qpid as backend, then you should set:

```
rpc_backend = qpid
```

And after that you should specify following options: `qpid_hostname`, `qpid_port`, `qpid_username`, `qpid_password` and `qpid_hosts`.

As example you can see default values of these options:

```
qpid_hostname=localhost
qpid_port=5672
qpid_hosts=$qpid_hostname:$qpid_port
qpid_username=
qpid_password=
```

2.3 Sahara Advanced Configuration Guide

This guide addresses specific aspects of Sahara configuration that pertain to advanced usage. It is divided into sections about various features that can be utilized, and their related configurations.

2.3.1 Domain usage for Swift proxy users

To improve security for Sahara clusters accessing Swift objects, Sahara can be configured to use proxy users and delegated trusts for access. This behavior has been implemented to reduce the need for storing and distributing user credentials.

The use of proxy users involves creating a domain in Keystone that will be designated as the home for any proxy users created. These created users will only exist for as long as a job execution runs. The domain created for the proxy users must have an identity backend that allows Sahara's admin user to create new user accounts. This new domain should contain no roles, to limit the potential access of a proxy user.

Once the domain has been created Sahara must be configured to use it by adding the domain name and any potential roles that must be used for Swift access in the sahara.conf file. With the domain enabled in Sahara, users will no longer be required to enter credentials with their Swift-backed Data Sources and Job Binaries.

Detailed instructions

First a domain must be created in Keystone to hold proxy users created by Sahara. This domain must have an identity backend that allows for Sahara to create new users. The default SQL engine is sufficient but if your Keystone identity is backed by LDAP or similar then domain specific configurations should be used to ensure Sahara's access. See the Keystone documentation for more information.

With the domain created Sahara's configuration file should be updated to include the new domain name and any potential roles that will be needed. For this example let's assume that the name of the proxy domain is `sahara_proxy` and the roles needed by proxy users will be `Member` and `SwiftUser`.

```
[DEFAULT]
use_domain_for_proxy_users=True
proxy_user_domain_name=sahara_proxy
proxy_user_role_names=Member,SwiftUser
```

A note on the use of roles. In the context of the proxy user, any roles specified here are roles intended to be delegated to the proxy user from the user with access to the Swift object store. More specifically, any roles that are required for Swift access by the project owning the object store must be delegated to the proxy user for Swift authentication to be successful.

Finally, the stack administrator must ensure that images registered with Sahara have the latest version of the Hadoop Swift filesystem plugin installed. The sources for this plugin can be found in the Sahara extra repository. For more information on images or Swift integration see the Sahara documentation sections *Building Images for Vanilla Plugin* and *Swift Integration*.

2.4 Sahara Upgrade Guide

This page contains some details about upgrading Sahara from one release to another like config file updates, db migrations, architecture changes and etc.

2.4.1 Icehouse -> Juno

Main binary renamed to sahara-all

Please, note that you should use *sahara-all* instead of *sahara-api* to start the All-In-One Sahara.

sahara.conf upgrade

We've migrated from custom auth_token middleware config options to the common config options. To update your config file you should replace the following old config opts with the new ones.

- `os_auth_protocol`, `os_auth_host`, `os_auth_port` -> `[keystone_authtoken]/auth_uri` and `[keystone_authtoken]/identity_uri`; it should be the full uri, for example: `http://127.0.0.1:5000/v2.0/`
- `os_admin_username` -> `[keystone_authtoken]/admin_user`
- `os_admin_password` -> `[keystone_authtoken]/admin_password`
- `os_admin_tenant_name` -> `[keystone_authtoken]/admin_tenant_name`

We've replaced oslo code from sahara.openstack.common.db by usage of oslo.db library.

Also sqlite database is not supported anymore. Please use MySQL or PostgreSQL db backends for Sahara. Sqlite support was dropped because it doesn't support (and not going to support, see http://www.sqlite.org/omitted.html) ALTER COLUMN and DROP COLUMN commands required for DB migrations between versions.

You can find more info about config file options in Sahara repository in file `etc/sahara/sahara.conf.sample`.

Sahara Dashboard was merged into OpenStack Dashboard

The Sahara Dashboard is not available in Juno release. Instead it's functionality is provided by OpenStack Dashboard out of the box. The Sahara UI is available in OpenStack Dashboard in "Project" -> "Data Processing" tab.

Note that you have to properly register Sahara in Keystone in order for Sahara UI in the Dashboard to work. For details see `:ref:'registering Sahara in installation guide <register-sahara-label>'`.

The sahara-dashboard project is now used solely to host Sahara UI integration tests.

VM user name changed for HEAT infrastructure engine

We've updated HEAT infrastructure engine (`infrastructure_engine=heat`) to use the same rules for instance user name as in direct engine. Before the change user name for VMs created by Sahara using HEAT engine was always 'ec2-user'. Now user name is taken from the image registry as it is described in the documentation.

Note, this change breaks Sahara backward compatibility for clusters created using HEAT infrastructure engine before the change. Clusters will continue to operate, but it is not recommended to perform scale operation over them.

Anti affinity implementation changed

Starting with Juno release anti affinity feature is implemented using server groups. There should not be much difference in Sahara behaviour from user perspective, but there are internal changes:

1. Server group object will be created if anti affinity feature is enabled

2. New implementation doesn't allow several affected instances on the same host even if they don't have common processes. So, if anti affinity enabled for 'datanode' and 'tasktracker' processes, previous implementation allowed to have instance with 'datanode' process and other instance with 'tasktracker' process on one host. New implementation guarantees that instances will be on different hosts.

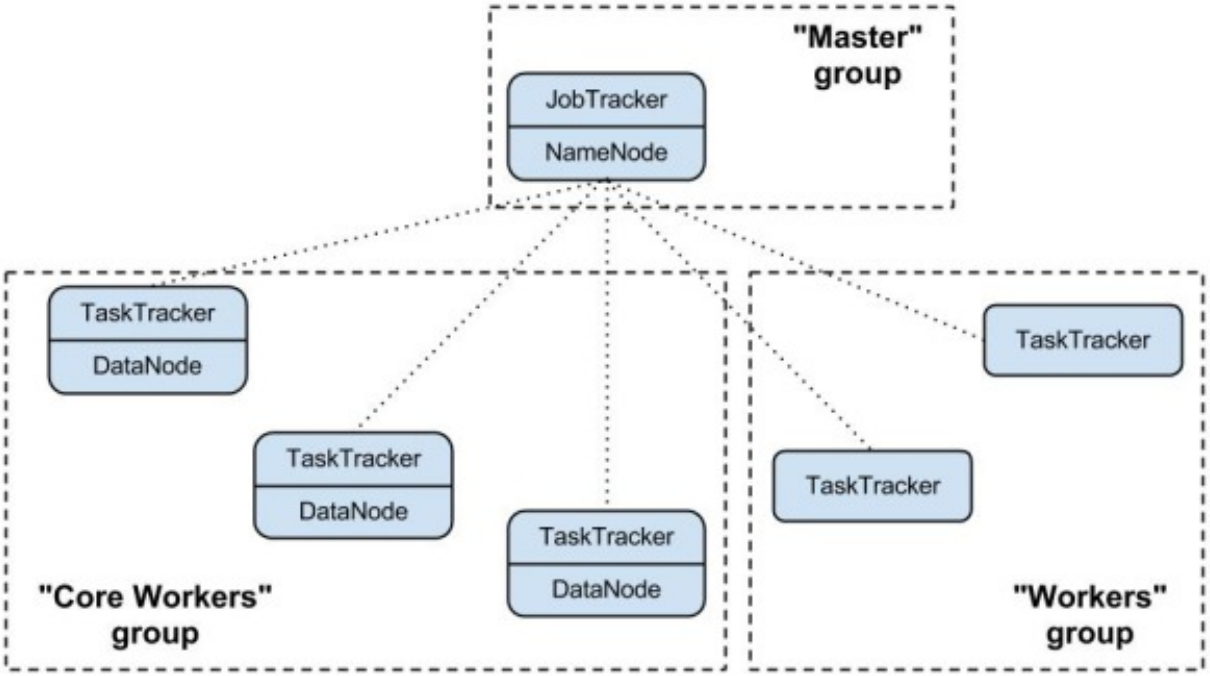
Note, new implementation will be applied for new clusters only. Old implementation will be applied if user scales cluster created in Icehouse.

How To

2.5 Getting Started

2.5.1 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.

2.5.2 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.

2.5.3 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*

2.5.4 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.

2.5.5 Features

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

2.6 Sahara (Data Processing) UI User Guide

This guide assumes that you already have the Sahara service and Horizon dashboard up and running. Don't forget to make sure that Sahara is registered in Keystone. If you require assistance with that, please see the installation guide.

2.6.1 Launching a cluster via the Sahara UI

2.6.2 Registering an Image

1. Navigate to the "Project" dashboard, then the "Data Processing" tab, 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 with Sahara
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

2.6.3 Create Node Group Templates

1. Navigate to the "Project" dashboard, then the "Data Processing" tab, 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

2.6.4 Create a Cluster Template

1. Navigate to the "Project" dashboard, then the "Data Processing" tab, 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

2.6.5 Launching a Cluster

1. Navigate to the "Project" dashboard, then the "Data Processing" tab, 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

2.6.6 Scaling a Cluster

1. From the Data Processing/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 "+" and "-" buttons

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

2.6.7 Elastic Data Processing (EDP)

2.6.8 Data Sources

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

1. From the Data Processing/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 of the the Data Source

- For a Swift object, enter <container>/<path> (ie: *mycontainer/inputfile*). Sahara will prepend *swift://* for you
- For an HDFS object, enter an absolute path, a relative path or a full URL:
  - */my/absolute/path* indicates an absolute path in the cluster HDFS
  - *my/path* indicates the path */user/hadoop/my/path* in the cluster HDFS assuming the defined HDFS user is *hadoop*
  - *hdfs://host:port/path* can be used to indicate any HDFS location

4. Enter the username and password for the Data Source (also see Additional Notes)
5. Enter an optional description
6. Click on "Create"
7. Repeat for additional Data Sources

2.6.9 Job Binaries

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

1. From the Data Processing/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", enter the URL of your binary (<container>/<path>) as well as the username and password (also see Additional Notes)
- 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

2.6.10 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 Data Processing/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
4. Choose the main binary from the dropdown
    - This is required for Hive, Pig, and Spark jobs
    - Other job types do not use a main binary
5. Enter an optional description for your Job
6. Click on the "Libs" tab and choose any libraries needed by your job
    - MapReduce and Java jobs require at least one library
    - Other job types may optionally use libraries
7. Click on "Create"

2.6.11 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 Data Processing/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

2.6.12 Example Jobs

There are sample jobs located in the sahara repository. In this section, we will give a walkthrough on how to run those jobs via the Horizon UI. These steps assume that you already have a cluster up and running (in the "Active" state).

1. Sample Pig job - https://github.com/openstack/sahara/tree/master/etc/edp-examples/pig-job

- Load the input data file from https://github.com/openstack/sahara/tree/master/etc/edp-examples/pig-job/data/input into swift
  - Click on Projet/Object Store/Containers and create a container with any name ("samplecontainer" for our purposes here)
  - Click on Upload Object and give the object a name ("piginput" in this case)
- Navigate to Data Processing/Data Sources, Click on Create Data Source
  - Name your Data Source ("pig-input-ds" in this sample)
  - Type = Swift, URL samplecontainer/piginput, fill-in the Source username/password fields with your username/password and click "Create"
- Create another Data Source to use as output for the job
  - Name = pig-output-ds, Type = Swift, URL = samplecontainer/pigoutput, Source username/password, "Create"
- Store your Job Binaries in the Sahara database
  - Navigate to Data Processing/Job Binaries, Click on Create Job Binary
  - Name = example.pig, Storage type = Internal database, click Browse and find example.pig wherever you checked out the sahara project <sahara root>/etc/edp-examples/pig-job
  - Create another Job Binary: Name = udf.jar, Storage type = Internal database, click Browse and find udf.jar wherever you checked out the sahara project <sahara root>/etc/edp-examples/pig-job
- Create a Job
  - Navigate to Data Processing/Jobs, Click on Create Job
  - Name = pigsample, Job Type = Pig, Choose "example.pig" as the main binary
  - Click on the "Libs" tab and choose "udf.jar", then hit the "Choose" button beneath the dropdown, then click on "Create"
- Launch your job
  - To launch your job from the Jobs page, click on the down arrow at the far right of the screen and choose "Launch on Existing Cluster"
  - For the input, choose "pig-input-ds", for output choose "pig-output-ds". Also choose whichever cluster you'd like to run the job on
  - For this job, no additional configuration is necessary, so you can just click on "Launch"
  - You will be taken to the "Job Executions" page where you can see your job progress through "PENDING, RUNNING, SUCCEEDED" phases
  - When your job finishes with "SUCCEEDED", you can navigate back to Object Store/Containers and browse to the samplecontainer to see your output. It should be in the "pigoutput" folder

2. Sample Spark job - https://github.com/openstack/sahara/tree/master/etc/edp-examples/edp-spark

- Store the Job Binary in the Sahara database
  - Navigate to Data Processing/Job Binaries, Click on Create Job Binary

  - Name = sparkexample.jar, Storage type = Internal database, Browse to the location <sahara root>/etc/edp-examples/edp-spark and choose spark-example.jar, Click "Create"

- Create a Job
  - Name = sparkexamplejob, Job Type = Spark, Main binary = Choose sparkexample.jar, Click "Create"
- Launch your job
  - To launch your job from the Jobs page, click on the down arrow at the far right of the screen and choose "Launch on Existing Cluster"
  - Choose whichever cluster you'd like to run the job on
  - Click on the "Configure" tab
  - Set the main class to be: org.apache.spark.examples.SparkPi
  - Under Arguments, click Add and fill in the number of "Slices" you want to use for the job. For this example, let's use 100 as the value
  - Click on Launch
  - You will be taken to the "Job Executions" page where you can see your job progress through "PENDING, RUNNING, SUCCEEDED" phases
  - When your job finishes with "SUCCEEDED", you can see your results by sshing to the Spark "master" node
  - The output is located at /tmp/spark-edp/<name of job>/<job execution id>. You can do `cat stdout` which should display something like "Pi is roughly 3.14156132"
  - It should be noted that for more complex jobs, the input/output may be elsewhere. This particular job just writes to stdout, which is logged in the folder under /tmp

2.6.13 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. In the examples above, we mention adding your username/password for the Swift Data Sources. It should be noted that it is possible to configure Sahara such that the username/password credentials are *not* required. For more information on that, please refer to: *Sahara Advanced Configuration Guide*

2.7 Features Overview

2.7.1 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.

2.7.2 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 using Swift with Sahara, including patching Hadoop and configuring Sahara, please refer to the *Swift Integration* documentation.

2.7.3 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.

2.7.4 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
```

2.7.5 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.

2.7.6 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.

Starting with Juno release Sahara creates server groups with `anti-affinity` policy to enable anti affinity feature. Sahara creates one server group per cluster and assigns all instances with affected processes to this server group. Refer to Nova documentation on how server groups work.

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

2.7.7 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.

2.7.8 Security group management

Sahara allows you to control which security groups will be used for created instances. This can be done by providing the `security_groups` parameter for the Node Group or Node Group Template. By default an empty list is used that will result in using the default security group.

Sahara may also create a security group for instances in node group automatically. This security group will only have open ports which are required by instance processes or the Sahara engine. This option is useful for development and secured from outside environments, but for production environments it is recommended to control security group policy manually.

2.7.9 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
```

There is a feature parity between direct and heat infrastructure engines. It is recommended to use heat engine since direct engine will be deprecated at some point.

2.7.10 Plugin Capabilities

The below tables provides a plugin capability matrix:

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

2.7.11 Running Sahara in Distributed Mode

Warning: Currently distributed mode for Sahara is in alpha state. We do not recommend using it in production environment.

The installation guide suggests to launch Sahara as a single 'sahara-all' process. It is also possible to run Sahara in distributed mode with 'sahara-api' and 'sahara-engine' processes running on several machines simultaneously.

Sahara-api works as a frontend and serves users' requests. It offloads 'heavy' tasks to sahara-engine via RPC mechanism. While sahara-engine could be loaded, sahara-api by design stays free and hence may quickly respond on user queries.

If Sahara runs on several machines, the API requests could be balanced between several sahara-api instances using a load balancer. It is not required to balance load between different sahara-engine instances, as that will be automatically done via a message queue.

If a single machine goes down, others will continue serving users' requests. Hence a better scalability is achieved and some fault tolerance as well. Note that the proposed solution is not a true High Availability. While failure of a single machine does not affect work of other machines, all of the operations running on the failed machine will stop. For example, if a cluster scaling is interrupted, the cluster will be stuck in a half-scaled state. The cluster will probably continue working, but it will be impossible to scale it further or run jobs on it via EDP.

To run Sahara in distributed mode pick several machines on which you want to run Sahara services and follow these steps:

- On each machine install and configure Sahara using the installation guide except:
  - Do not run 'sahara-db-manage' or launch Sahara with 'sahara-all'

  - Make sure sahara.conf provides database connection string to a single database on all machines.
- Run 'sahara-db-manage' as described in the installation guide, but only on a single (arbitrarily picked) machine.
- sahara-api and sahara-engine processes use oslo.messaging to communicate with each other. You need to configure it properly on each node (see below).
- run sahara-api and sahara-engine on the desired nodes. On a node you can run both sahara-api and sahara-engine or you can run them on separate nodes. It does not matter as long as they are configured to use the same message broker and database.

To configure oslo.messaging, first you need to pick the driver you are going to use. Right now three drivers are provided: Rabbit MQ, Qpid or Zmq. To use Rabbit MQ or Qpid driver, you will have to setup messaging broker. The picked driver must be supplied in `sahara.conf` in `[DEFAULT]/rpc_backend` parameter. Use one the following values: `rabbit`, `qpid` or `zmq`. Next you have to supply driver-specific options.

Unfortunately, right now there is no documentation with description of drivers' configuration. The options are available only in source code.

- For Rabbit MQ see
  - rabbit_opts variable in impl_rabbit.py
  - amqp_opts variable in amqp.py
- For Qpid see
  - qpid_opts variable in impl_qpid.py
  - amqp_opts variable in amqp.py
- For Zmq see
  - zmq_opts variable in impl_zmq.py
  - matchmaker_opts variable in matchmaker.py
  - matchmaker_redis_opts variable in matchmaker_redis.py
  - matchmaker_opts variable in matchmaker_ring.py

You can find the same options defined in `sahara.conf.sample`. You can use it to find section names for each option (matchmaker options are defined not in `[DEFAULT]`)

2.8 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

2.9 Provisioning Plugins

This page lists all available provisioning plugins. In general a plugin enables Sahara to deploy a specific data intensive application (Hadoop, Spark) distribution in various topologies and with management/monitoring tools.

- *Vanilla Plugin* - deploys Vanilla Apache Hadoop
- *Hortonworks Data Platform Plugin* - deploys Hortonworks Data Platform
- *Spark Plugin* - deploys Apache Spark with Cloudera HDFS

2.10 Vanilla Plugin

The vanilla plugin is a reference implementation which allows users to operate a cluster with Apache Hadoop.

For cluster provisioning prepared images should be used. They already have Apache Hadoop 1.2.1 and Apache Hadoop 2.4.1 installed. Prepared images can be found at the following locations:

- http://sahara-files.mirantis.com/sahara-juno-vanilla-1.2.1-ubuntu-14.04.qcow2
- http://sahara-files.mirantis.com/sahara-juno-vanilla-1.2.1-centos-6.5.qcow2
- http://sahara-files.mirantis.com/sahara-juno-vanilla-1.2.1-fedora-20.qcow2
- http://sahara-files.mirantis.com/sahara-juno-vanilla-2.4.1-ubuntu-14.04.qcow2
- http://sahara-files.mirantis.com/sahara-juno-vanilla-2.4.1-centos-6.5.qcow2
- http://sahara-files.mirantis.com/sahara-juno-vanilla-2.4.1-fedora-20.qcow2

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

The default username specified for these images is different for each distribution:

OS	username
Ubuntu 14.04	ubuntu
Fedora 20	fedora
CentOS 6.5	cloud-user

2.10.1 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

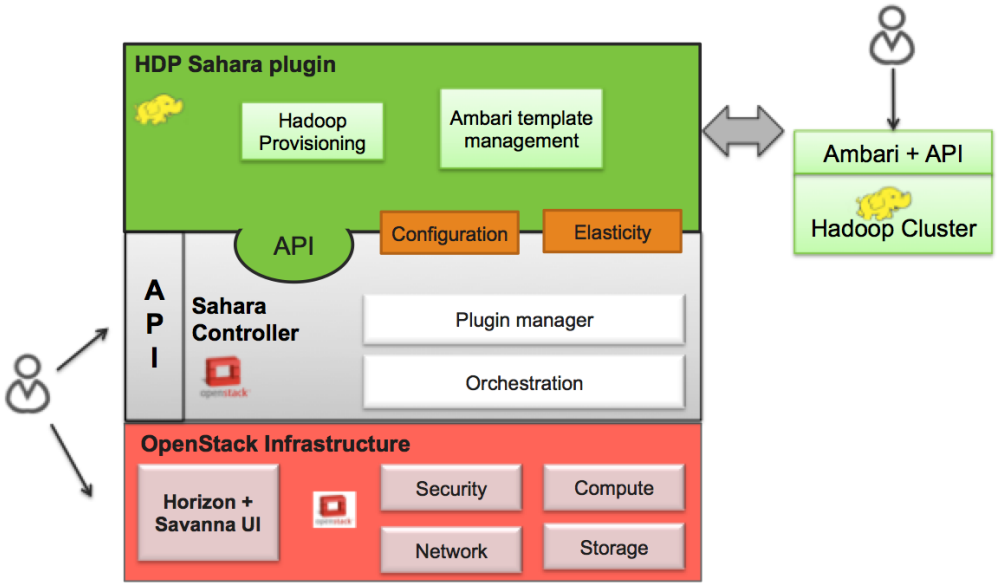
- Cluster can't have tasktracker nodes if it doesn't have jobtracker

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

2.11 Hortonworks Data Platform 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.

2.11.1 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.

2.11.2 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

2.11.3 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. sahara-juno-hdp-1.3.2-centos-6.5.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. sahara-juno-hdp-2.0.6-centos-6.5.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. sahara-juno-hdp-plain-centos-6.5.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 'cloud-user'.

2.11.4 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.

2.11.5 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.

2.11.6 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

2.11.7 The HDP Plugin and Sahara Support

For more information, please contact Hortonworks.

2.12 Spark Plugin

The Spark Sahara plugin provides a way to provision Apache Spark clusters on OpenStack in a single click and in an easily repeatable fashion.

Currently Spark is installed in standalone mode, with no YARN or Mesos support.

2.12.1 Images

For cluster provisioning prepared images should be used. The Spark plugin has been developed and tested with the images generated by the *Building Images for Vanilla Plugin*. Those Ubuntu images already have Cloudera CDH4 HDFS and Apache Spark installed. A prepared image can be found at the following location:

- http://sahara-files.mirantis.com/sahara-juno-spark-1.0.0-ubuntu-14.04.qcow2

The Spark plugin requires an image to be tagged in Sahara Image Registry with two tags: 'spark' and '<Spark version>' (e.g. '1.0.0').

Also you should specify the username of the default cloud-user used in the image. For images generated with the DIB it is 'ubuntu'.

Note that the Spark cluster is deployed using the scripts available in the Spark distribution, which allow to start all services (master and slaves), stop all services and so on. As such (and as opposed to CDH HDFS daemons), Spark is not deployed as a standard Ubuntu service and if the virtual machines are rebooted, Spark will not be restarted.

2.12.2 Spark configuration

Spark needs few parameters to work and has sensible defaults. If needed they can be changed when creating the Sahara cluster template. No node group options are available.

Once the cluster is ready, connect with ssh to the master using the 'ubuntu' user and the appropriate ssh key. Spark is installed in /opt/spark and should be completely configured and ready to start executing jobs. At the bottom of the cluster information page from the OpenStack dashboard, a link to the Spark web interface is provided.

2.12.3 Cluster Validation

When a user creates an Hadoop cluster using the Spark plugin, the cluster topology requested by user is verified for consistency.

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

- Cluster must contain exactly one HDFS namenode
- Cluster must contain exactly one Spark master
- Cluster must contain at least one Spark slave
- Cluster must contain at least one HDFS datanode

The tested configuration puts the NameNode co-located with the master and a DataNode with each slave to maximize data locality.

2.12.4 Limitations

Swift support is not available in Spark. Once it is developed there, it will be possible to add it to this plugin.

Elastic Data Processing

2.13 Elastic Data Processing (EDP)

2.13.1 Overview

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

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

2.13.2 Interfaces

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

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

2.13.3 EDP Concepts

Sahara EDP uses a collection of simple objects to define and execute 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 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 unless Swift proxy users are configured as described in *Sahara Advanced Configuration Guide*. 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
`MapReduce.Streaming`	not used	optional
`Java`	not used	required
`Spark`	required	optional

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 Hadoop jobs, Sahara generates an Oozie 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.

For Spark jobs, Sahara uses the *spark-submit* shell script and executes the Spark job from the master node. Logs of spark jobs run by Sahara can be found on the master node under the */tmp/spark-edp* directory.

2.13.4 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 or Spark job types. See Additional Details for Java jobs and Additional Details for Spark 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
`MapReduce.Streaming`	Yes	No	No
`Java`	Yes	No	Yes
`Spark`	Yes	No	Yes

- *Configuration values* are key/value pairs.
  - The EDP configuration values have names beginning with *edp.* and are consumed by Sahara
  - Other configuration values may be read at runtime by Hadoop jobs
  - Currently additional configuration values are not available to Spark jobs at runtime
- *Parameters* are key/value pairs. They supply values for the Hive and Pig parameter substitution mechanisms.
- *Arguments* are strings passed as command line arguments to a shell or main program

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 Swift proxy users are not configured (see *Sahara Advanced Configuration Guide*) and 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. Note, the UI will not prompt the user for these required values, they must be added manually with the `Configure` tab. Make sure to add these values with the correct names:

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

Additional Details for MapReduce.Streaming jobs

Important!

If the job type is MapReduce.Streaming, the streaming mapper and reducer classes *must* be specified.

In this case, the UI *will* prompt the user to enter mapper and reducer values on the form and will take care of adding them to the job configuration with the appropriate names. If using the python client, however, be certain to add these values to the job configuration manually with the correct names:

Name	Example Value
edp.streaming.mapper	/bin/cat
edp.streaming.reducer	/usr/bin/wc

Additional Details for Java jobs

Java jobs use two special configuration values:

- `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.

Additional Details for Spark jobs

Spark jobs use a special configuration value:

- `edp.java.main_class` (required) Specifies the class containing the Java or Scala main method:
  - `main(String[] args)` for Java
  - `main(args: Array[String]` for Scala

A Spark job will execute the `main` method of the specified main class. Values may be passed to the main method through the `args` array. Any arguments set during job launch will be passed to the program as commandline arguments by *spark-submit*.

Data Source objects are not used with Spark job types. Instead, any input or output paths must be passed to the `main` method as arguments. Remember that Swift paths are not supported for Spark jobs currently.

The `edp-spark` example bundled with Sahara contains a Spark program for estimating Pi.

2.13.5 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 passed to running jobs as input or output sources include a ".sahara" suffix on the container, for example:

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

You may notice these Swift URLs in job logs, however, you do not need to add the suffix to the containers yourself. Sahara will add the suffix if necessary, so when using the UI or the python client you may write the above URL simply as:

```
swift://container/object
```

Sahara internal database URLs have the form:

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

This indicates a file object in the Sahara database which has the given uuid as a key

2.14 EDP Requirements

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

2.14.1 OpenStack Services

When a Hadoop job is executed, binaries are first uploaded to a cluster node and then moved from the node 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

2.14.2 Cluster Processes

Requirements for EDP support depend on the EDP job type and plugin used for the cluster. For example a Vanilla Sahara cluster must run at least one 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

2.15 EDP Technical Considerations

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

2.15.1 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

2.16 Sahara REST API docs

2.16.1 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/{template-name}	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",
                "2.3.0",
                "2.4.1"
            ],
            "name": "vanilla",
            "title": "Vanilla Apache Hadoop"
        },
        {
            "description": "The Hortonworks OpenStack plugin works with project Sahara to automa
            "versions": [
                "1.3.2",
                "2.0.6"
            ],
            "name": "hdp",
            "title": "Hortonworks Data Platform"
        },
        {
            "description": "This plugin provides an ability to launch Spark on Hadoop CDH cluste
            "versions": [
```

```
                "1.0.0",
                "0.9.1"
            ],
            "name": "spark",
            "title": "Apache Spark"
        },
        {
            "description": "This plugin provides an ability to launch CDH clusters with Cloudera
            "versions": [
                "5"
            ],
            "name": "cdh",
            "title": "Cloudera Plugin"
        }
    ]
}
```

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",
            "2.3.0",
            "2.4.1"
        ]
    }
}
```

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/2.4.1
```

response

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

{
    "plugin": {
        "node_processes": {
            "HDFS": [
                "namenode",
                "datanode",
                "secondarynamenode"
            ],
            "JobFlow": [
                "oozie"
            ],
            "Hadoop": [],
            "YARN": [
                "resourcemanager",
                "nodemanager"
            ],
            "MapReduce": [
                "historyserver"
            ]
        },
        "description": "This plugin provides an ability to launch vanilla Apache Hadoop cluster
        "versions": [
            "1.2.1",
            "2.3.0",
            "2.4.1"
        ],
        "required_image_tags": [
            "vanilla",
            "2.4.1"
        ],
        "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."
                },
                {
                    "default_value": 1024,
                    "name": "NodeManager Heap Size",
                    "config_values": null,
                    "priority": 1,
                    "config_type": "int",
                    "applicable_target": "YARN",
                    "is_optional": false,
                    "scope": "node",
                    "description": null
                },
                {
                    "default_value": true,
                    "name": "Enable Swift",
                    "config_values": null,
                    "priority": 1,
                    "config_type": "bool",
                    "applicable_target": "general",
                    "is_optional": false,
                    "scope": "cluster",
                    "description": null
                },
                {
                    "default_value": true,
                    "name": "Enable MySQL",
                    "config_values": null,
                    "priority": 1,
                    "config_type": "bool",
                    "applicable_target": "general",
                    "is_optional": true,
                    "scope": "cluster",
                    "description": null
                }
            ],
            "title": "Vanilla Apache Hadoop",
            "name": "vanilla"
        }
    }
```


2.4 Convert configuration file

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

Normal Response Code: 202 (ACCEPTED)

Errors: none

This operation returns Sahara's JSON representation of a cluster template created from the posted configuration.

The request body should contain configuration file.

Example: request

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

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,
                "security_groups": [],
                "auto_security_group": False,
                "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,
                "security_groups": [],
                "auto_security_group": False,
                "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": "ubuntu",
            "updated": "2014-08-26T07:29:36Z",
            "OS-EXT-IMG-SIZE:size": 965476352,
            "name": "ubuntu-sahara-vanilla-2.4.1",
            "created": "2014-08-26T07:16:40Z",
            "tags": [
                "2.4.1",
                "vanilla"
            ],
```

```
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_username": "ubuntu",
                "_sahara_tag_2.4.1": "True",
                "_sahara_description": "Ubuntu image for Hadoop 2.4.1",
                "_sahara_tag_vanilla": "True"
            },
            "id": "5880a275-df8e-49cc-991a-e3a0b1fcf8ea",
            "description": "Ubuntu image for Hadoop 2.4.1"
        },
        {
            "status": "ACTIVE",
            "username": "ubuntu",
            "updated": "2014-08-08T12:45:37Z",
            "OS-EXT-IMG-SIZE:size": 962658304,
            "name": "sahara-icehouse-vanilla-1.2.1-ubuntu-13.10",
            "created": "2014-08-08T12:43:47Z",
            "tags": [
                "vanilla",
                "1.2.1"
            ],
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_username": "ubuntu",
                "_sahara_tag_vanilla": "True",
                "_sahara_tag_1.2.1": "True"
            },
            "id": "d62ad147-5c10-418c-a21a-3a6597044f29",
            "description": null
        }
    ]
}
```


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=2.4.1
```

response

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

```
{
    "images": [
        {
            "status": "ACTIVE",
            "username": "ubuntu",
            "updated": "2014-08-26T07:29:36Z",
            "OS-EXT-IMG-SIZE:size": 965476352,
            "name": "ubuntu-sahara-vanilla-2.4.1",
            "created": "2014-08-26T07:16:40Z",
            "tags": [
                "2.4.1",
                "vanilla"
            ],
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_username": "ubuntu",
                "_sahara_tag_2.4.1": "True",
                "_sahara_description": "Ubuntu image for Hadoop 2.4.1",
                "_sahara_tag_vanilla": "True"
            },
            "id": "5880a275-df8e-49cc-991a-e3a0b1fcf8ea",
            "description": "Ubuntu image for Hadoop 2.4.1"
        }
    ]
}
```


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/d62ad147-5c10-418c-a21a-3a6597044f29
```

response

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


```
{
    "image": {
        "status": "ACTIVE",
        "username": "ubuntu",
        "updated": "2014-08-08T12:45:37Z",
        "OS-EXT-IMG-SIZE:size": 962658304,
        "name": "sahara-icehouse-vanilla-1.2.1-ubuntu-13.10",
        "created": "2014-08-08T12:43:47Z",
        "tags": [
            "vanilla",
```

```
            "1.2.1"
        ],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ubuntu",
            "_sahara_tag_vanilla": "True",
            "_sahara_tag_1.2.1": "True"
        },
        "id": "d62ad147-5c10-418c-a21a-3a6597044f29",
        "description": null
    }
}
```

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/5880a275-df8e-49cc-991a-e3a0b1fcf8ea

{
    "username": "ubuntu",
    "description": "Ubuntu image for Hadoop 2.4.1"
}
```

response

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

{
    "image": {
        "status": "ACTIVE",
        "username": "ubuntu",
        "updated": "2014-08-26T07:24:02Z",
        "OS-EXT-IMG-SIZE:size": 965476352,
        "name": "ubuntu-sahara-vanilla-2.4.1",
        "created": "2014-08-26T07:16:40Z",
        "tags": [],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ubuntu",
            "_sahara_description": "Ubuntu image for Hadoop 2.4.1"
        },
        "id": "5880a275-df8e-49cc-991a-e3a0b1fcf8ea",
        "description": "Ubuntu image for Hadoop 2.4.1"
    }
}
```

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/5880a275-df8e-49cc-991a-e3a0b1fcf8ea
```

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/5880a275-df8e-49cc-991a-e3a0b1fcf8ea/tag

{
    "tags": ["vanilla", "2.4.1", "some_other_tag"]
}
```

response

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

{
    "image": {
        "status": "ACTIVE",
        "username": "ubuntu",
        "updated": "2014-08-26T07:27:10Z",
        "OS-EXT-IMG-SIZE:size": 965476352,
        "name": "ubuntu-sahara-vanilla-2.4.1",
        "created": "2014-08-26T07:16:40Z",
        "tags": [
            "some_other_tag",
            "vanilla",
            "2.4.1"
        ],
```

```
            "minDisk": 0,
            "progress": 100,
            "minRam": 0,
            "metadata": {
                "_sahara_username": "ubuntu",
                "_sahara_tag_some_other_tag": "True",
                "_sahara_tag_vanilla": "True",
                "_sahara_description": "Ubuntu image for Hadoop 2.4.1",
                "_sahara_tag_2.4.1": "True"
            },
            "id": "5880a275-df8e-49cc-991a-e3a0b1fcf8ea",
            "description": "Ubuntu image for Hadoop 2.4.1"
        }
    }
```

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/5880a275-df8e-49cc-991a-e3a0b1fcf8ea/untag

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

response

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

{
    "image": {
        "status": "ACTIVE",
        "username": "ubuntu",
        "updated": "2014-08-26T07:29:36Z",
        "OS-EXT-IMG-SIZE:size": 965476352,
        "name": "ubuntu-sahara-vanilla-2.4.1",
        "created": "2014-08-26T07:16:40Z",
        "tags": [
            "2.4.1",
            "vanilla"
        ],
        "minDisk": 0,
        "progress": 100,
        "minRam": 0,
        "metadata": {
            "_sahara_username": "ubuntu",
            "_sahara_tag_2.4.1": "True",
            "_sahara_description": "Ubuntu image for Hadoop 2.4.1",
```

```
            "_sahara_tag_vanilla": "True"
        },
        "id": "5880a275-df8e-49cc-991a-e3a0b1fcf8ea",
        "description": "Ubuntu image for Hadoop 2.4.1"
    }
}
```

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": [
        {
            "hadoop_version": "2.4.1",
            "security_groups": null,
            "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
            "name": "worker",
            "updated_at": null,
            "description": null,
            "plugin_name": "vanilla",
```

```
            "image_id": null,
            "volumes_size": 0,
            "id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
            "auto_security_group": null,
            "volumes_per_node": 0,
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "node_processes": [
                "datanode",
                "nodemanager"
            ],
            "created_at": "2014-09-02 23:02:29",
            "node_configs": {
                "HDFS": {
                    "DataNode Heap Size": 1024
                },
                "YARN": {
                    "NodeManager Heap Size": 2048
                }
            },
            "volume_mount_prefix": "/volumes/disk",
            "flavor_id": "3"
        },
        {
            "hadoop_version": "2.4.1",
            "security_groups": null,
            "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
            "name": "master",
            "updated_at": null,
            "description": null,
            "plugin_name": "vanilla",
            "image_id": null,
            "volumes_size": 0,
            "id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
            "auto_security_group": null,
            "volumes_per_node": 0,
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "node_processes": [
                "namenode",
                "resourcemanager",
                "oozie",
                "historyserver"
            ],
            "created_at": "2014-09-02 23:01:33",
            "node_configs": {},
            "volume_mount_prefix": "/volumes/disk",
            "flavor_id": "3"
        }
    ]
}
```


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/b900b4dc-d3ee-4341-99c3-ac078301f9d8
```

response

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

{
    "node_group_template": {
        "hadoop_version": "2.4.1",
        "security_groups": null,
        "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
        "name": "master",
        "updated_at": null,
        "description": null,
        "plugin_name": "vanilla",
        "image_id": null,
        "volumes_size": 0,
        "id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
        "auto_security_group": null,
        "volumes_per_node": 0,
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "node_processes": [
            "namenode",
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "created_at": "2014-09-02 23:01:33",
        "node_configs": {},
        "volume_mount_prefix": "/volumes/disk",
        "flavor_id": "3"
    }
}
```

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": "2.4.1",
    "node_processes": [
        "namenode",
```

```
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "name": "master",
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "flavor_id": "3"
}
```

response

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

{
    "node_group_template": {
        "name": "master",
        "volume_mount_prefix": "/volumes/disk",
        "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
        "created_at": "2014-08-26 08:14:46.119233",
        "plugin_name": "vanilla",
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "volumes_size": 0,
        "node_processes": [
            "namenode",
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "flavor_id": "3",
        "volumes_per_node": 0,
        "auto_security_group": null,
        "hadoop_version": "2.4.1",
        "id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
        "security_groups": null
    }
}
```

Example with configurations: request

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

{
    "plugin_name": "vanilla",
    "hadoop_version": "2.4.1",
    "node_processes": [
        "datanode",
        "nodemanager"
    ],
    "name": "worker",
    "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
    "flavor_id": "3",
    "node_configs": {
        "HDFS": {
            "DataNode Heap Size": 1024
        },
        "YARN": {
            "NodeManager Heap Size": 2048
        }
```

```
    }
}
```

response

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

{
    "node_group_template": {
        "name": "worker",
        "volume_mount_prefix": "/volumes/disk",
        "tenant_id": "28a4d0e49b024dc0875ed6a862b129f0",
        "created_at": "2014-08-26 08:23:06.740466",
        "plugin_name": "vanilla",
        "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
        "volumes_size": 0,
        "node_processes": [
            "datanode",
            "nodemanager"
        ],
        "flavor_id": "3",
        "volumes_per_node": 0,
        "security_groups": [],
        "auto_security_group": False,
        "node_configs": {
            "HDFS": {
                "DataNode Heap Size": 1024
            },
            "YARN": {
                "NodeManager Heap Size": 2048
            }
        },
        "hadoop_version": "2.4.1",
        "id": "3b975888-42d4-43d3-be70-8e4401e3cb65",
        "security_groups": null
    }
}
```

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": [
        {
            "hadoop_version": "2.4.1",
            "default_image_id": null,
            "name": "cluster-template",
            "updated_at": null,
            "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
            "plugin_name": "vanilla",
            "anti_affinity": [],
            "description": null,
            "id": "1beae95b-fd20-47c0-a745-5125dccbd560",
            "node_groups": [
                {
                    "security_groups": null,
```

```
                "name": "master",
                "updated_at": null,
                "count": 1,
                "node_processes": [
                    "namenode",
                    "resourcemanager",
                    "oozie",
                    "historyserver"
                ],
                "node_configs": {},
                "volumes_size": 0,
                "auto_security_group": null,
                "volumes_per_node": 0,
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "node_group_template_id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
                "created_at": "2014-09-02 23:05:23",
                "image_id": null,
                "volume_mount_prefix": "/volumes/disk",
                "flavor_id": "3"
            },
            {
                "security_groups": null,
                "name": "worker",
                "updated_at": null,
                "count": 3,
                "node_processes": [
                    "datanode",
                    "nodemanager"
                ],
                "node_configs": {
                    "HDFS": {
                        "DataNode Heap Size": 1024
                    },
                    "YARN": {
                        "NodeManager Heap Size": 2048
                    }
                },
                "volumes_size": 0,
                "auto_security_group": null,
                "volumes_per_node": 0,
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
                "created_at": "2014-09-02 23:05:23",
                "image_id": null,
                "volume_mount_prefix": "/volumes/disk",
                "flavor_id": "3"
            }
        ],
        "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
        "created_at": "2014-09-02 23:05:23",
        "cluster_configs": {}
    },
    {
        "hadoop_version": "2.4.1",
        "default_image_id": null,
        "name": "cluster-template-3",
        "updated_at": null,
        "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
```

```
"plugin_name": "vanilla",
"anti_affinity": [],
"description": null,
"id": "3d5bdb90-c8c5-4100-81b8-81d23cecaab2",
"node_groups": [
    {
        "security_groups": null,
        "name": "master",
        "updated_at": null,
        "count": 1,
        "node_processes": [
            "namenode",
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "node_configs": {},
        "volumes_size": 0,
        "auto_security_group": null,
        "volumes_per_node": 0,
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "node_group_template_id": null,
        "created_at": "2014-09-02 23:06:39",
        "image_id": null,
        "volume_mount_prefix": "/volumes/disk",
        "flavor_id": "3"
    },
    {
        "security_groups": null,
        "name": "worker",
        "updated_at": null,
        "count": 3,
        "node_processes": [
            "datanode",
            "nodemanager"
        ],
        "node_configs": {
            "HDFS": {
                "DataNode Heap Size": 1024
            },
            "YARN": {
                "NodeManager Heap Size": 2048
            }
        },
        "volumes_size": 0,
        "auto_security_group": null,
        "volumes_per_node": 0,
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "node_group_template_id": null,
        "created_at": "2014-09-02 23:06:38",
        "image_id": null,
        "volume_mount_prefix": "/volumes/disk",
        "flavor_id": "3"
    }
],
"neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
"created_at": "2014-09-02 23:06:38",
"cluster_configs": {
```

```
                    "HDFS": {
                        "dfs.replication": 3
                    },
                    "general": {
                        "Enable Swift": true,
                        "Enable MySQL": true
                    }
                }
            }
        ]
    }
```

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/1beae95b-fd20-47c0-a745-5125dccbd560
```

response

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

{
    "cluster_template": {
        "hadoop_version": "2.4.1",
        "default_image_id": null,
        "name": "cluster-template",
        "updated_at": null,
        "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "description": null,
        "id": "1beae95b-fd20-47c0-a745-5125dccbd560",
        "node_groups": [
            {
                "security_groups": null,
                "name": "master",
                "updated_at": null,
                "count": 1,
                "node_processes": [
                    "namenode",
                    "resourcemanager",
                    "oozie",
                    "historyserver"
                ],
                "node_configs": {},
                "volumes_size": 0,
                "auto_security_group": null,
```

```
                    "volumes_per_node": 0,
                    "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                    "node_group_template_id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
                    "created_at": "2014-09-02 23:05:23",
                    "image_id": null,
                    "volume_mount_prefix": "/volumes/disk",
                    "flavor_id": "3"
                },
                {
                    "security_groups": null,
                    "name": "worker",
                    "updated_at": null,
                    "count": 3,
                    "node_processes": [
                        "datanode",
                        "nodemanager"
                    ],
                    "node_configs": {
                        "HDFS": {
                            "DataNode Heap Size": 1024
                        },
                        "YARN": {
                            "NodeManager Heap Size": 2048
                        }
                    },
                    "volumes_size": 0,
                    "auto_security_group": null,
                    "volumes_per_node": 0,
                    "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                    "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
                    "created_at": "2014-09-02 23:05:23",
                    "image_id": null,
                    "volume_mount_prefix": "/volumes/disk",
                    "flavor_id": "3"
                }
            ],
            "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
            "created_at": "2014-09-02 23:05:23",
            "cluster_configs": {}
        }
    }
```


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": "2.4.1",
    "node_groups": [
        {
            "name": "worker",
            "count": 3,
            "node_group_template_id": "3b975888-42d4-43d3-be70-8e4401e3cb65"
        },
        {
            "name": "master",
            "count": 1,
            "node_group_template_id": "208f2d53-69c3-48c3-9830-986db4c29c95"
        }
    ],
    "name": "cluster-template",
    "neutron_management_network": "0b001fb7-b172-43f0-8c99-444672fd0513",
    "cluster_configs": {}
}
```

response

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

{
    "cluster_template": {
        "neutron_management_network": "0b001fb7-b172-43f0-8c99-444672fd0513",
        "description": null,
        "cluster_configs": {},
        "created_at": "2014-08-28 20:00:40",
        "default_image_id": null,
        "updated_at": null,
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "tenant_id": "28a4d0e49b024dc0875ed6a862b129f0",
        "node_groups": [
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "auto_security_group": null,
                "created_at": "2014-08-28 20:00:40",
                "updated_at": null,
                "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
                "image_id": null,
                "volumes_size": 0,
                "node_processes": [
                    "datanode",
                    "nodemanager"
                ],
                "node_group_template_id": "3b975888-42d4-43d3-be70-8e4401e3cb65",
                "volumes_per_node": 0,
                "node_configs": {
                    "HDFS": {
                        "DataNode Heap Size": 1024
                    },
                    "YARN": {
                        "NodeManager Heap Size": 2048
                    }
```

```
                },
                "security_groups": null,
                "flavor_id": "3"
            },
            {
                "count": 1,
                "name": "master",
                "volume_mount_prefix": "/volumes/disk",
                "auto_security_group": null,
                "created_at": "2014-08-28 20:00:40",
                "updated_at": null,
                "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
                "image_id": null,
                "volumes_size": 0,
                "node_processes": [
                    "namenode",
                    "resourcemanager",
                    "oozie",
                    "historyserver"
                ],
                "node_group_template_id": "208f2d53-69c3-48c3-9830-986db4c29c95",
                "volumes_per_node": 0,
                "node_configs": {},
                "security_groups": null,
                "flavor_id": "3"
            }
        ],
        "hadoop_version": "2.4.1",
        "id": "1beae95b-fd20-47c0-a745-5125dccbd560",
        "name": "cluster-template"
    }
}
```


Example with configurations and no Node Group Templates: request

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


```
{
    "plugin_name": "vanilla",
    "hadoop_version": "2.4.1",
    "name": "cluster-template-3",
    "neutron_management_network": "0b001fb7-b172-43f0-8c99-444672fd0513",
    "cluster_configs": {
        "general": {
          "Enable Swift": true,
          "Enable MySQL": true
        },
        "HDFS": {
            "dfs.replication": 3
        }
    },
    "node_groups": [
        {
            "count": 3,
            "name": "worker",
            "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
            "node_processes": [
                "datanode",
```

```
                "nodemanager"
            ],
            "node_configs": {
                "HDFS": {
                    "DataNode Heap Size": 1024
                },
                "YARN": {
                    "NodeManager Heap Size": 2048
                }
            },
            "flavor_id": "3"
        },
        {
            "count": 1,
            "name": "master",
            "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
            "node_processes": [
                "namenode",
                "resourcemanager",
                "oozie",
                "historyserver"
            ],
            "flavor_id": "3"
        }
    ]
}
```

response

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

{
    "cluster_template": {
        "neutron_management_network": "0b001fb7-b172-43f0-8c99-444672fd0513",
        "description": null,
        "cluster_configs": {
            "HDFS": {
                "dfs.replication": 3
            },
            "general": {
                "Enable MySQL": true,
                "Enable Swift": true
            }
        },
        "created_at": "2014-08-28 20:20:38",
        "default_image_id": null,
        "updated_at": null,
        "plugin_name": "vanilla",
        "anti_affinity": [],
        "tenant_id": "28a4d0e49b024dc0875ed6a862b129f0",
        "node_groups": [
            {
                "count": 3,
                "name": "worker",
                "volume_mount_prefix": "/volumes/disk",
                "auto_security_group": null,
                "created_at": "2014-08-28 20:20:38",
```

```
                    "updated_at": null,
                    "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
                    "image_id": null,
                    "volumes_size": 0,
                    "node_processes": [
                        "datanode",
                        "nodemanager"
                    ],
                    "node_group_template_id": null,
                    "volumes_per_node": 0,
                    "node_configs": {
                        "HDFS": {
                            "DataNode Heap Size": 1024
                        },
                        "YARN": {
                            "NodeManager Heap Size": 2048
                        }
                    },
                    "security_groups": null,
                    "flavor_id": "3"
                },
                {
                    "count": 1,
                    "name": "master",
                    "volume_mount_prefix": "/volumes/disk",
                    "auto_security_group": null,
                    "created_at": "2014-08-28 20:20:38",
                    "updated_at": null,
                    "floating_ip_pool": "cdeaa720-5517-4878-860e-71a1926744aa",
                    "image_id": null,
                    "volumes_size": 0,
                    "node_processes": [
                        "namenode",
                        "resourcemanager",
                        "oozie",
                        "historyserver"
                    ],
                    "node_group_template_id": null,
                    "volumes_per_node": 0,
                    "node_configs": {},
                    "security_groups": null,
                    "flavor_id": "3"
                }
            ],
            "hadoop_version": "2.4.1",
            "id": "3a9c68e5-47f0-479b-9ee9-f86ccb0be68c",
            "name": "cluster-template-3"
        }
    }
```


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/3a9c68e5-47f0-479b-9ee9-f86ccb0be68c
```

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": "Active",
            "info": {
```

```
        "HDFS": {
            "NameNode": "hdfs://doc-cluster-master-001:9000",
            "Web UI": "http://172.18.168.227:50070"
        },
        "JobFlow": {
            "Oozie": "http://172.18.168.227:11000"
        },
        "MapReduce JobHistory Server": {
            "Web UI": "http://172.18.168.227:19888"
        },
        "YARN": {
            "Web UI": "http://172.18.168.227:8088"
        }
    },
    "cluster_template_id": "1beae95b-fd20-47c0-a745-5125dccbd560",
    "is_transient": false,
    "description": null,
    "cluster_configs": {},
    "created_at": "2014-09-02 23:13:50",
    "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
    "user_keypair_id": "doc-keypair",
    "updated_at": "2014-09-02 23:17:22",
    "plugin_name": "vanilla",
    "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
    "anti_affinity": [],
    "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
    "node_groups": [
        {
            "count": 1,
            "name": "master",
            "auto_security_group": null,
            "instances": [
                {
                    "instance_name": "doc-cluster-master-001",
                    "created_at": "2014-09-02 23:13:53",
                    "updated_at": "2014-09-02 23:14:27",
                    "instance_id": "59dd622c-787d-4bb8-98a2-33887dfc5b41",
                    "management_ip": "172.18.168.227",
                    "volumes": [],
                    "internal_ip": "10.50.0.55",
                    "id": "a01cd5a1-5c4e-419e-9718-c8e839995150"
                }
            ],
            "volume_mount_prefix": "/volumes/disk",
            "created_at": "2014-09-02 23:13:50",
            "updated_at": "2014-09-02 23:13:53",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "image_id": null,
            "volumes_size": 0,
            "node_configs": {},
            "node_group_template_id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
            "volumes_per_node": 0,
            "node_processes": [
                "namenode",
                "resourcemanager",
                "oozie",
                "historyserver"
            ],
```

```
            "security_groups": null,
            "flavor_id": "3"
        },
        {
            "count": 3,
            "name": "worker",
            "auto_security_group": null,
            "instances": [
                {
                    "instance_name": "doc-cluster-worker-001",
                    "created_at": "2014-09-02 23:13:52",
                    "updated_at": "2014-09-02 23:14:27",
                    "instance_id": "be59bf7b-5b63-4e63-ba13-fbfd94078885",
                    "management_ip": "172.18.168.226",
                    "volumes": [],
                    "internal_ip": "10.50.0.53",
                    "id": "b4e9d4ad-e421-4bf1-8b4d-756154f7396a"
                },
                {
                    "instance_name": "doc-cluster-worker-002",
                    "created_at": "2014-09-02 23:13:52",
                    "updated_at": "2014-09-02 23:14:28",
                    "instance_id": "19c55dea-2a03-41c6-adba-2512a38bc708",
                    "management_ip": "172.18.168.228",
                    "volumes": [],
                    "internal_ip": "10.50.0.56",
                    "id": "e1cb99d6-bce5-4df6-8725-522224154119"
                },
                {
                    "instance_name": "doc-cluster-worker-003",
                    "created_at": "2014-09-02 23:13:53",
                    "updated_at": "2014-09-02 23:14:27",
                    "instance_id": "25ee1e5e-1839-4919-be85-70733bf0238b",
                    "management_ip": "172.18.168.225",
                    "volumes": [],
                    "internal_ip": "10.50.0.54",
                    "id": "1bdbc0bc-bd15-481f-9f64-a6c79449afe4"
                }
            ],
            "volume_mount_prefix": "/volumes/disk",
            "created_at": "2014-09-02 23:13:50",
            "updated_at": "2014-09-02 23:13:53",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "image_id": null,
            "volumes_size": 0,
            "node_configs": {
                "HDFS": {
                    "DataNode Heap Size": 1024
                },
                "YARN": {
                    "NodeManager Heap Size": 2048
                }
            },
            "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
            "volumes_per_node": 0,
            "node_processes": [
                "datanode",
                "nodemanager"
```

```
                    ],
                    "security_groups": null,
                    "flavor_id": "3"
                }
            ],
            "management_public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDcKdaU6FpUV0qyDkOazP
            "status_description": "",
            "hadoop_version": "2.4.1",
            "id": "fa57eed8-ee5e-4f9a-b365-7ca92e389ba0",
            "trust_id": null,
            "name": "doc-cluster"
        }
    ]
}
```

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/fa57eed8-ee5e-4f9a-b365-7ca92e389ba0
```

response

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

{
    "cluster": {
        "status": "Active",
        "info": {
            "HDFS": {
                "NameNode": "hdfs://doc-cluster-master-001:9000",
                "Web UI": "http://172.18.168.227:50070"
            },
            "JobFlow": {
                "Oozie": "http://172.18.168.227:11000"
            },
            "MapReduce JobHistory Server": {
                "Web UI": "http://172.18.168.227:19888"
            },
            "YARN": {
                "Web UI": "http://172.18.168.227:8088"
            }
        },
        "cluster_template_id": "1beae95b-fd20-47c0-a745-5125dccbd560",
        "is_transient": false,
        "description": null,
        "cluster_configs": {},
        "created_at": "2014-09-02 23:13:50",
```

```
"default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
"user_keypair_id": "doc-keypair",
"updated_at": "2014-09-02 23:17:22",
"plugin_name": "vanilla",
"neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
"anti_affinity": [],
"tenant_id": "af8996ec973444048f159f2bf2e3c24e",
"node_groups": [
    {
        "count": 1,
        "name": "master",
        "auto_security_group": null,
        "instances": [
            {
                "instance_name": "doc-cluster-master-001",
                "created_at": "2014-09-02 23:13:53",
                "updated_at": "2014-09-02 23:14:27",
                "instance_id": "59dd622c-787d-4bb8-98a2-33887dfc5b41",
                "management_ip": "172.18.168.227",
                "volumes": [],
                "internal_ip": "10.50.0.55",
                "id": "a01cd5a1-5c4e-419e-9718-c8e839995150"
            }
        ],
        "volume_mount_prefix": "/volumes/disk",
        "created_at": "2014-09-02 23:13:50",
        "updated_at": "2014-09-02 23:13:53",
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "image_id": null,
        "volumes_size": 0,
        "node_configs": {},
        "node_group_template_id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
        "volumes_per_node": 0,
        "node_processes": [
            "namenode",
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "security_groups": null,
        "flavor_id": "3"
    },
    {
        "count": 3,
        "name": "worker",
        "auto_security_group": null,
        "instances": [
            {
                "instance_name": "doc-cluster-worker-001",
                "created_at": "2014-09-02 23:13:52",
                "updated_at": "2014-09-02 23:14:27",
                "instance_id": "be59bf7b-5b63-4e63-ba13-fbfd94078885",
                "management_ip": "172.18.168.226",
                "volumes": [],
                "internal_ip": "10.50.0.53",
                "id": "b4e9d4ad-e421-4bf1-8b4d-756154f7396a"
            },
            {
```

```
                            "instance_name": "doc-cluster-worker-002",
                            "created_at": "2014-09-02 23:13:52",
                            "updated_at": "2014-09-02 23:14:28",
                            "instance_id": "19c55dea-2a03-41c6-adba-2512a38bc708",
                            "management_ip": "172.18.168.228",
                            "volumes": [],
                            "internal_ip": "10.50.0.56",
                            "id": "e1cb99d6-bce5-4df6-8725-522224154119"
                        },
                        {
                            "instance_name": "doc-cluster-worker-003",
                            "created_at": "2014-09-02 23:13:53",
                            "updated_at": "2014-09-02 23:14:27",
                            "instance_id": "25ee1e5e-1839-4919-be85-70733bf0238b",
                            "management_ip": "172.18.168.225",
                            "volumes": [],
                            "internal_ip": "10.50.0.54",
                            "id": "1bdbc0bc-bd15-481f-9f64-a6c79449afe4"
                        }
                    ],
                    "volume_mount_prefix": "/volumes/disk",
                    "created_at": "2014-09-02 23:13:50",
                    "updated_at": "2014-09-02 23:13:53",
                    "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                    "image_id": null,
                    "volumes_size": 0,
                    "node_configs": {
                        "HDFS": {
                            "DataNode Heap Size": 1024
                        },
                        "YARN": {
                            "NodeManager Heap Size": 2048
                        }
                    },
                    "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
                    "volumes_per_node": 0,
                    "node_processes": [
                        "datanode",
                        "nodemanager"
                    ],
                    "security_groups": null,
                    "flavor_id": "3"
                }
            ],
            "management_public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDcKdaU6FpUV0qyDkOazP6ffX
            "status_description": "",
            "hadoop_version": "2.4.1",
            "id": "fa57eed8-ee5e-4f9a-b365-7ca92e389ba0",
            "trust_id": null,
            "name": "doc-cluster"
        }
    }
```


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": "2.4.1",
    "cluster_template_id": "1beae95b-fd20-47c0-a745-5125dccbd560",
    "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
    "user_keypair_id": "doc-keypair",
    "name": "doc-cluster",
    "cluster_configs": {}
}
```

response

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


```
{
    "cluster": {
        "status": "Validating",
        "info": {},
        "cluster_template_id": "1beae95b-fd20-47c0-a745-5125dccbd560",
        "is_transient": false,
        "description": null,
        "cluster_configs": {},
        "created_at": "2014-09-02 23:40:36",
        "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
        "user_keypair_id": "doc-keypair",
        "updated_at": "2014-09-02 23:40:36.265920",
        "plugin_name": "vanilla",
        "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
        "anti_affinity": [],
        "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
        "node_groups": [
            {
                "count": 1,
                "name": "master",
                "instances": [],
                "volume_mount_prefix": "/volumes/disk",
                "created_at": "2014-09-02 23:40:36",
                "updated_at": null,
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "image_id": null,
                "volumes_size": 0,
                "node_configs": {},
                "node_group_template_id": "b900b4dc-d3ee-4341-99c3-ac078301f9d8",
                "volumes_per_node": 0,
                "node_processes": [
                    "namenode",
                    "resourcemanager",
                    "oozie",
                    "historyserver"
```

```
                ],
                "security_groups": null,
                "auto_security_group": null,
                "flavor_id": "3"
            },
            {
                "count": 3,
                "name": "worker",
                "instances": [],
                "volume_mount_prefix": "/volumes/disk",
                "auto_security_group": null,
                "created_at": "2014-09-02 23:40:36",
                "updated_at": null,
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "image_id": null,
                "volumes_size": 0,
                "node_configs": {
                    "HDFS": {
                        "DataNode Heap Size": 1024
                    },
                    "YARN": {
                        "NodeManager Heap Size": 2048
                    }
                },
                "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "nodemanager"
                ],
                "security_groups": null,
                "flavor_id": "3"
            }
        ],
        "management_public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDBgbVVyS6gQZA28TooEqzUKv
        "status_description": "",
        "hadoop_version": "2.4.1",
        "id": "c8c3fee5-075a-4969-875b-9a00bb9c7c6c",
        "trust_id": null,
        "name": "doc-cluster"
    }
}
```


Example Cluster creation from Node Groups and with configurations: request

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


```
{
    "plugin_name": "vanilla",
    "hadoop_version": "2.4.1",
    "name": "doc-cluster",
    "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
    "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
    "cluster_configs": {
        "general": {
          "Enable Swift": true,
          "Enable MySQL": true
        },
```

```
        "HDFS": {
            "dfs.replication": 3
        }
    },
    "node_groups": [
        {
            "count": 3,
            "name": "worker",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "node_processes": [
                "datanode",
                "nodemanager"
            ],
            "node_configs": {
                "HDFS": {
                    "DataNode Heap Size": 1024
                },
                "YARN": {
                    "NodeManager Heap Size": 2048
                }
            },
            "flavor_id": "3"
        },
        {
            "count": 1,
            "name": "master",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "node_processes": [
                "namenode",
                "resourcemanager",
                "oozie",
                "historyserver"
            ],
            "flavor_id": "3"
        }
    ]
}
```

response

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

{
    "cluster": {
        "status": "Waiting",
        "info": {},
        "cluster_template_id": null,
        "is_transient": false,
        "description": null,
        "cluster_configs": {
            "HDFS": {
                "dfs.replication": 3
            },
            "general": {
                "Enable MySQL": true,
                "Enable Swift": true
            }
```

```
},
"created_at": "2014-09-02 23:34:09",
"default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
"user_keypair_id": null,
"updated_at": "2014-09-02 23:34:13",
"plugin_name": "vanilla",
"neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
"anti_affinity": [],
"tenant_id": "af8996ec973444048f159f2bf2e3c24e",
"node_groups": [
    {
        "count": 1,
        "name": "master",
        "auto_security_group": null,
        "instances": [
            {
                "instance_name": "cluster-template-3-master-001",
                "created_at": "2014-09-02 23:34:13",
                "updated_at": "2014-09-02 23:34:56",
                "instance_id": "c7d17c4f-56fc-46a4-bcd1-76ec3d459d82",
                "management_ip": "172.18.168.233",
                "volumes": [],
                "internal_ip": "10.50.0.59",
                "id": "47aac1fc-11e2-4f89-b699-69ede345379b"
            }
        ],
        "volume_mount_prefix": "/volumes/disk",
        "created_at": "2014-09-02 23:34:09",
        "updated_at": "2014-09-02 23:34:13",
        "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
        "image_id": null,
        "volumes_size": 0,
        "node_configs": {},
        "node_group_template_id": null,
        "volumes_per_node": 0,
        "node_processes": [
            "namenode",
            "resourcemanager",
            "oozie",
            "historyserver"
        ],
        "security_groups": null,
        "flavor_id": "3"
    },
    {
        "count": 3,
        "name": "worker",
        "auto_security_group": null,
        "instances": [
            {
                "instance_name": "cluster-template-3-worker-001",
                "created_at": "2014-09-02 23:34:11",
                "updated_at": "2014-09-02 23:34:55",
                "instance_id": "3e2a0cc1-fd25-42c0-885d-efffb11f56e3",
                "management_ip": "172.18.168.232",
                "volumes": [],
                "internal_ip": "10.50.0.57",
                "id": "e6b41b36-dfa8-49f6-ab19-a3796d510014"
```

```
                    },
                    {
                        "instance_name": "cluster-template-3-worker-002",
                        "created_at": "2014-09-02 23:34:12",
                        "updated_at": "2014-09-02 23:34:55",
                        "instance_id": "9e4d5f63-1424-4a8c-b830-b953fb674854",
                        "management_ip": "172.18.168.231",
                        "volumes": [],
                        "internal_ip": "10.50.0.60",
                        "id": "41d8808d-00f1-4887-8791-6ee990307095"
                    },
                    {
                        "instance_name": "cluster-template-3-worker-003",
                        "created_at": "2014-09-02 23:34:12",
                        "updated_at": "2014-09-02 23:34:56",
                        "instance_id": "4e7ecea4-1d2d-46ff-983f-ad3134601662",
                        "management_ip": "172.18.168.234",
                        "volumes": [],
                        "internal_ip": "10.50.0.58",
                        "id": "16d59d71-25fa-42eb-9a7a-f050224dd653"
                    }
                ],
                "volume_mount_prefix": "/volumes/disk",
                "created_at": "2014-09-02 23:34:09",
                "updated_at": "2014-09-02 23:34:12",
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "image_id": null,
                "volumes_size": 0,
                "node_configs": {
                    "HDFS": {
                        "DataNode Heap Size": 1024
                    },
                    "YARN": {
                        "NodeManager Heap Size": 2048
                    }
                },
                "node_group_template_id": null,
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "nodemanager"
                ],
                "security_groups": null,
                "flavor_id": "3"
            }
        ],
        "management_public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCwXKFhoOhyyKF3xtFcWv/TYw
        "status_description": "",
        "hadoop_version": "2.4.1",
        "id": "b77e8def-a66d-4df8-bc9a-10a9a216fd60",
        "trust_id": null,
        "name": "doc-cluster"
    }
}
```

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": 4,
            "name": "worker"
        }
    ],

    "add_node_groups": [
        {
            "count": 2,
            "name": "big-worker",
            "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6"
        }
    ]
}
```

response

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

{
    "cluster": {
        "status": "Configuring",
        "info": {
            "HDFS": {
                "NameNode": "hdfs://cluster-template-3-master-001:9000",
                "Web UI": "http://172.18.168.233:50070"
            },
            "JobFlow": {
                "Oozie": "http://172.18.168.233:11000"
            },
            "MapReduce JobHistory Server": {
                "Web UI": "http://172.18.168.233:19888"
            },
            "YARN": {
                "Web UI": "http://172.18.168.233:8088"
            }
        },
        "cluster_template_id": null,
        "is_transient": false,
        "description": null,
        "cluster_configs": {
```

```
        "HDFS": {
            "dfs.replication": 3
        },
        "general": {
            "Enable MySQL": true,
            "Enable Swift": true
        }
    },
    "created_at": "2014-09-02 23:34:09",
    "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7",
    "user_keypair_id": null,
    "updated_at": "2014-09-02 23:47:28",
    "plugin_name": "vanilla",
    "neutron_management_network": "8b826011-27af-4068-a36a-9488d6d0d1c5",
    "anti_affinity": [],
    "tenant_id": "af8996ec973444048f159f2bf2e3c24e",
    "node_groups": [
        {
            "auto_security_group": null,
            "instances": [
                {
                    "instance_name": "cluster-template-3-big-worker-001",
                    "created_at": "2014-09-02 23:46:38",
                    "updated_at": "2014-09-02 23:47:02",
                    "instance_id": "3bba57b4-737f-4d84-a441-f1ef456ab0fe",
                    "management_ip": "172.18.168.235",
                    "volumes": [],
                    "internal_ip": "10.50.0.64",
                    "id": "281ea99f-2ef7-42c9-a192-4988d6b5d15b"
                },
                {
                    "instance_name": "cluster-template-3-big-worker-002",
                    "created_at": "2014-09-02 23:46:39",
                    "updated_at": "2014-09-02 23:47:03",
                    "instance_id": "edfa4f85-fd6d-41c7-9318-98ab19c53191",
                    "management_ip": "172.18.168.244",
                    "volumes": [],
                    "internal_ip": "10.50.0.65",
                    "id": "54521553-8806-4e64-bd56-4ace7a62566b"
                }
            ],
            "volume_mount_prefix": "/volumes/disk",
            "created_at": "2014-09-02 23:46:36",
            "updated_at": "2014-09-02 23:46:39",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "image_id": null,
            "volumes_size": 0,
            "node_configs": {
                "HDFS": {
                    "DataNode Heap Size": 1024
                },
                "YARN": {
                    "NodeManager Heap Size": 2048
                }
            },
            "node_group_template_id": "734551b4-0542-4bc1-b9bf-85f77d85c6f6",
            "volumes_per_node": 0,
            "node_processes": [
```

```
                "datanode",
                "nodemanager"
            ],
            "auto_security_group": null,
            "instances": [
                {
                    "instance_name": "cluster-template-3-master-001",
                    "created_at": "2014-09-02 23:34:13",
                    "updated_at": "2014-09-02 23:34:56",
                    "instance_id": "c7d17c4f-56fc-46a4-bcd1-76ec3d459d82",
                    "management_ip": "172.18.168.233",
                    "volumes": [],
                    "internal_ip": "10.50.0.59",
                    "id": "47aac1fc-11e2-4f89-b699-69ede345379b"
                }
            ],
            "volume_mount_prefix": "/volumes/disk",
            "created_at": "2014-09-02 23:34:09",
            "updated_at": "2014-09-02 23:34:13",
            "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
            "image_id": null,
            "volumes_size": 0,
            "node_configs": {},
            "node_group_template_id": null,
            "volumes_per_node": 0,
            "node_processes": [
                "namenode",
                "resourcemanager",
                "oozie",
                "historyserver"
            ],
            "auto_security_group": null,
            "security_groups": null,
            "flavor_id": "3"
        },
        {
            "count": 4,
            "name": "worker",
            "instances": [
                {
                    "instance_name": "cluster-template-3-worker-001",
                    "created_at": "2014-09-02 23:34:11",
                    "updated_at": "2014-09-02 23:34:55",
                    "instance_id": "3e2a0cc1-fd25-42c0-885d-efffb11f56e3",
                    "management_ip": "172.18.168.232",
                    "volumes": [],
                    "internal_ip": "10.50.0.57",
                    "id": "e6b41b36-dfa8-49f6-ab19-a3796d510014"
                },
                {
                    "instance_name": "cluster-template-3-worker-002",
                    "created_at": "2014-09-02 23:34:12",
                    "updated_at": "2014-09-02 23:34:55",
                    "instance_id": "9e4d5f63-1424-4a8c-b830-b953fb674854",
                    "management_ip": "172.18.168.231",
                    "volumes": [],
                    "internal_ip": "10.50.0.60",
                    "id": "41d8808d-00f1-4887-8791-6ee990307095"
```

```
                    },
                    {
                        "instance_name": "cluster-template-3-worker-003",
                        "created_at": "2014-09-02 23:34:12",
                        "updated_at": "2014-09-02 23:34:56",
                        "instance_id": "4e7ecea4-1d2d-46ff-983f-ad3134601662",
                        "management_ip": "172.18.168.234",
                        "volumes": [],
                        "internal_ip": "10.50.0.58",
                        "id": "16d59d71-25fa-42eb-9a7a-f050224dd653"
                    },
                    {
                        "instance_name": "cluster-template-3-worker-004",
                        "created_at": "2014-09-02 23:46:39",
                        "updated_at": "2014-09-02 23:47:03",
                        "instance_id": "6bdd8744-8591-453b-8ad8-27593a97825a",
                        "management_ip": "172.18.168.245",
                        "volumes": [],
                        "internal_ip": "10.50.0.66",
                        "id": "5a930e18-1dbf-4958-8686-32cd4a741048"
                    }
                ],
                "volume_mount_prefix": "/volumes/disk",
                "created_at": "2014-09-02 23:34:09",
                "updated_at": "2014-09-02 23:46:40",
                "floating_ip_pool": "77e2c46d-9585-46a2-95f9-8721c302b257",
                "image_id": null,
                "volumes_size": 0,
                "node_configs": {
                    "HDFS": {
                        "DataNode Heap Size": 1024
                    },
                    "YARN": {
                        "NodeManager Heap Size": 2048
                    }
                },
                "node_group_template_id": null,
                "volumes_per_node": 0,
                "node_processes": [
                    "datanode",
                    "nodemanager"
                ],
                "security_groups": null,
                "flavor_id": "3"
            }
        ],
        "management_public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCwXKFhoOhyyKF3xtFcWv/TYw
        "status_description": "",
        "hadoop_version": "2.4.1",
        "id": "b77e8def-a66d-4df8-bc9a-10a9a216fd60",
        "trust_id": null,
        "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
```

2.16.2 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:

This example creates a Swift data source.

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-76fbc3b8c6
        "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

2.17 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.

2.17.1 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

2.17.2 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.

2.17.3 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.

2.18 Swift Integration

Hadoop and Swift integration are the essential continuation of the Hadoop/OpenStack marriage. The key component to making this marriage work is the Hadoop Swift filesystem implementation. Although this implementation has been merged into the upstream Hadoop project, Sahara maintains a version with the most current features enabled.

- The original Hadoop patch can be found at https://issues.apache.org/jira/browse/HADOOP-8545
- The most current Sahara maintained version of this patch can be found in the Sahara Extra repository https://github.com/openstack/sahara-extra
- The latest compiled version of the jar for this component can be downloaded from http://sahara-files.mirantis.com/hadoop-swift/hadoop-swift-latest.jar

2.18.1 Hadoop patching

You may build the jar file yourself by choosing the latest patch from the Sahara Extra repository and using Maven to build with the pom.xml file provided. Or you may get the latest jar pre-built from the CDN at http://sahara-files.mirantis.com/hadoop-swift/hadoop-swift-latest.jar

You will need to put this file into the hadoop libraries (e.g. /usr/lib/share/hadoop/lib) on each job-tracker and task-tracker node for Hadoop 1.x, or each ResourceManager and NodeManager node for Hadoop 2.x in the cluster.

2.18.2 Hadoop configurations

In general, when Sahara runs a job on a cluster it will handle configuring the Hadoop installation. In cases where a user might require more in-depth configuration all the data is set in the `core-site.xml` file on the cluster instances 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. The 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`
   - `.domain.name` - Domains can be used to specify users who are not in the tenant specified.
   - `.trust.id` - Trusts are optionally used to scope the authentication tokens of the supplied user.
   - `.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`

2.18.3 Example

For this example it is assumed that you have setup a Hadoop instance with a valid configuration and the Swift filesystem component. Furthermore there is assumed to be a Swift container named `integration` holding an object named `temp`, as well as a Keystone user named `admin` with a password of `swordfish`.

The following example illustrates how to copy an object to a new location in the same container. We will use Hadoop's `distcp` command (http://hadoop.apache.org/docs/r0.19.0/distcp.html) to accomplish the copy. Note that the service provider for our Swift access is `sahara`, and that we will not need to specify the project of our Swift container as it will be provided in the Hadoop configuration.

Swift paths are expressed in Hadoop according to the following template: `swift://${container}.${provider}/${object}`. For our example source this will appear as `swift://integration.sahara/temp`.

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 confirm that `temp1` has been created in our `integration` container.

2.18.4 Limitations

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

2.19 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 cloud images for all versions of supported plugins and all operating systems (subset of Ubuntu, Fedora, and CentOS depending on plugin). 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 set your values of parameters.

   Then it will create required cloud images using image elements that install all the necessary packages and configure them. You will find created images in the current directory.

Note: Disk Image Builder will generate QCOW2 images, used with the default OpenStack Qemu/KVM hypervisors. If your OpenStack uses a different hypervisor, the generated image should be converted to an appropriate format.

VMware Nova backend requires VMDK image format. You may use qemu-img utility to convert a QCOW2 image to VMDK.

```
qemu-img convert -O vmdk <original_image>.qcow2 <converted_image>.vmdk
```

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. Create config file from the sample:

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

4. Look through the sahara.conf and modify parameter values as needed. For details see *Sahara Configuration Guide*

5. Create database schema:

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

6. To start Sahara call:

```
$ tox -e venv -- sahara-all --config-file etc/sahara/sahara.conf --debug
```

3.2.3 Setup local OpenStack dashboard with Sahara plugin

Sahara UI Dev Environment Setup

This page describes how to setup Horizon for developing Sahara by either installing it as part of DevStack with Sahara or installing it in an isolated environment and running from the command line.

Install as a part of DevStack

See the DevStack guide for more information on installing and configuring DevStack with Sahara.

After Horizon installation, it will contain a Data Processing tab under Projects tab. Sahara UI source code will be located at `$DEST/horizon/openstack_dashboard/dashboards/project/data_processing` where `$DEST/` is usually `/opt/stack/`.

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:

   ```
   $ git clone https://github.com/openstack/horizon
   ```

   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"
   ```

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

   ```
   SAHARA_AUTO_IP_ALLOCATION_ENABLED = True
   ```

5. If Sahara is not registered in keystone service catalog, then we should modify `openstack_dashboard/api/sahara.py`:

   Add following lines before `def client(request):` Note, that you should replace the ip and port in `SAHARA_URL` with the appropriate values.

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

   def get_sahara_url(request):

       if SAHARA_URL:
           url = SAHARA_URL.rstrip('/')
           if url.split('/')[-1] in ['v1.0', 'v1.1']:
               url = SAHARA_URL + '/' + request.user.tenant_id
           return url
   ```

```
    return base.url_for(request, SAHARA_SERVICE)
```

After that modify sahara_url provided in `def client(request):`

```
sahara_url=get_sahara_url(request)
```

6. 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.

7. Applying changes

If you have changed any `*.py` files in `horizon/openstack_dashboard/dashboards/project/data_processin` 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 6.

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 that the `~/.pip/cache` folder should be created manually.

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 Setup DevStack

The DevStack could be installed on Fedora, Ubuntu and CentOS. For supported versions see DevStack documentation

We recommend to install DevStack not into your main system, but run it in a VM instead. That way you may avoid contamination of your system with various stuff. You may find hypervisor and VM requirements in the the next section. If you still want to install DevStack on top of your main system, just skip the next section and read further.

3.3.1 Start VM and set up OS

In order to run DevStack in a local VM, you need to start by installing a guest with Ubuntu 12.04 server. Download an image file from Ubuntu's web site and create a new guest from it. Virtualization solution must support nested virtualization. Without nested virtualization VMs running inside the DevStack will be extremely slow lacking hardware acceleration, i.e. you will run QEMU VMs without KVM.

On Linux QEMU/KVM supports nested virtualization, on Mac OS - VMware Fusion. VMware Fusion requires adjustments to run VM with fixed IP. You may find instructions which can help *below*.

Start a new VM with Ubuntu Server 12.04. Recommended settings:

- Processor - at least 2 cores
- Memory - at least 8GB
- Hard Drive - at least 60GB

When allocating CPUs and RAM to the DevStack, assess how big clusters you want to run. A single Hadoop VM needs at least 1 cpu and 1G of RAM to run. While it is possible for several VMs to share a single cpu core, remember that they can't share the RAM.

After you installed the VM, connect to it via SSH and proceed with the instructions below.

3.3.2 Install DevStack

The instructions assume that you've decided to install DevStack into Ubuntu 12.04 system.

1. Clone DevStack:

```
$ sudo apt-get install git-core
$ git clone https://github.com/openstack-dev/devstack.git
```

2. Create file `local.conf` in devstack directory with the following content:

```
[[local|localrc]]
ADMIN_PASSWORD=nova
MYSQL_PASSWORD=nova
RABBIT_PASSWORD=nova
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=nova

# Enable Swift
enable_service s-proxy s-object s-container s-account

SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1
SWIFT_DATA_DIR=$DEST/data

# Force checkout prerequsites
# FORCE_PREREQ=1

# keystone is now configured by default to use PKI as the token format which produces huge tokens.
# set UUID as keystone token format which is much shorter and easier to work with.
KEYSTONE_TOKEN_FORMAT=UUID
```

```
# Change the FLOATING_RANGE to whatever IPs VM is working in.
# In NAT mode it is subnet VMware Fusion provides, in bridged mode it is your local network.
# But only use the top end of the network by using a /27 and starting at the 224 octet.
FLOATING_RANGE=192.168.55.224/27

# Enable auto assignment of floating IPs. By default Sahara expects this setting to be enabled
EXTRA_OPTS=(auto_assign_floating_ip=True)

# Enable logging
SCREEN_LOGDIR=$DEST/logs/screen

# Set ''OFFLINE'' to ''True'' to configure ''stack.sh'' to run cleanly without
# Internet access. ''stack.sh'' must have been previously run with Internet
# access to install prerequisites and fetch repositories.
# OFFLINE=True

# Enable Sahara
enable_service sahara
```

3. Sahara can send notifications to Ceilometer, if Ceilometer is enabled. If you want to enable Ceilometer add the following lines to `local.conf` file:

```
enable_service ceilometer-acompute ceilometer-acentral ceilometer-anotification ceilometer-collector
enable_service ceilometer-alarm-evaluator,ceilometer-alarm-notifier
enable_service ceilometer-api
```

4. Start DevStack:

```
$ ./stack.sh
```

5. Once previous step is finished Devstack will print Horizon URL. Navigate to this URL and login with login "admin" and password from `local.conf`.
6. Congratulations! You have OpenStack running in your VM and ready to launch VMs inside that VM :)

3.3.3 Managing Sahara in DevStack

If you install DevStack with Sahara included you can rejoin screen with `rejoin-stack.sh` command and switch to `sahara` tab. Here you can manage the Sahara service as other OpenStack services. Sahara source code is located at `$DEST/sahara` which is usually `/opt/stack/sahara`.

3.3.4 Setting fixed IP address for VMware Fusion VM

1. Open file `/Library/Preferences/VMware Fusion/vmnet8/dhcpd.conf`
2. There is a block named "subnet". It might look like this:

```
subnet 192.168.55.0 netmask 255.255.255.0 {
        range 192.168.55.128 192.168.55.254;
```

3. You need to pick an IP address outside of that range. For example - `192.168.55.20`
4. Copy VM MAC address from VM settings->Network->Advanced
5. Append the following block to file `dhcpd.conf` (don't forget to replace `VM_HOSTNAME` and `VM_MAC_ADDRESS` with actual values):

```
host VM_HOSTNAME {
        hardware ethernet VM_MAC_ADDRESS;
        fixed-address 192.168.55.20;
}
```

6. Now quit all the VmWare Fusion applications and restart vmnet:

```
$ sudo /Applications/VMware\ Fusion.app/Contents/Library/vmnet-cli --stop
$ sudo /Applications/VMware\ Fusion.app/Contents/Library/vmnet-cli --start
```

7. Now start your VM, it should have new fixed IP address

3.4 Sahara UI Dev Environment Setup

This page describes how to setup Horizon for developing Sahara by either installing it as part of DevStack with Sahara or installing it in an isolated environment and running from the command line.

3.4.1 Install as a part of DevStack

See the DevStack guide for more information on installing and configuring DevStack with Sahara.

After Horizon installation, it will contain a Data Processing tab under Projects tab. Sahara UI source code will be located at `$DEST/horizon/openstack_dashboard/dashboards/project/data_processing` where `$DEST/` is usually `/opt/stack/`.

3.4.2 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:

   ```
   $ git clone https://github.com/openstack/horizon
   ```

   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"
```

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

```
SAHARA_AUTO_IP_ALLOCATION_ENABLED = True
```

5. If Sahara is not registered in keystone service catalog, then we should modify `openstack_dashboard/api/sahara.py`:

   Add following lines before `def client(request):` Note, that you should replace the ip and port in `SAHARA_URL` with the appropriate values.

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

def get_sahara_url(request):

    if SAHARA_URL:
        url = SAHARA_URL.rstrip('/')
        if url.split('/')[-1] in ['v1.0', 'v1.1']:
            url = SAHARA_URL + '/' + request.user.tenant_id
        return url

    return base.url_for(request, SAHARA_SERVICE)
```

   After that modify sahara_url provided in `def client(request):`

```
sahara_url=get_sahara_url(request)
```

6. 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.

7. Applying changes

   If you have changed any `*.py` files in `horizon/openstack_dashboard/dashboards/project/data_processin` 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 6.

3.5 Quickstart guide

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

3.5.1 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*.

3.5.2 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"
```

3.5.3 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"
```

3.5.4 4. Register image in Image Registry

- Now we will actually start to interact with Sahara.
- Register the image with username `ubuntu`.

```
$ sahara image-register --image-id $IMAGE_ID --username ubuntu
```

- Tag the image:

```
$ sahara image-add-tag --image-id $IMAGE_ID --tag vanilla
$ sahara image-add-tag --image-id $IMAGE_ID --tag 1.2.1
```

- Make sure that image is registered correctly:

```
$ sahara image-list
```

- Output should look like:

```
$ sahara image-list
+----------------+---------------+----------+----------------+-------------+
| name           | id            | username | tags           | description |
+----------------+---------------+----------+----------------+-------------+
| sahara-iceh... | 3f9fc...6919b | ubuntu   | vanilla, 1.2.1 | None        |
+----------------+---------------+----------+----------------+-------------+
```

3.5.5 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"],
    "auto_security_group": True
}
```

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"],
    "auto_security_group": True
}
```

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,
            "security_groups": [],
            "auto_security_group": True
        },
        {
            "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,
            "security_groups": [],
            "auto_security_group": True
        }
    ]
}
```

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`

3.5.6 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`.

3.5.7 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,
                    "security_groups": ["a314895b-d2ee-431d-a26b-7c37b45894c9"],
                    "auto_security_group": True
                },
                {
```

```
                    "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,
                    "security_groups": ["b260407f-a566-43bf-a010-7e8b23953dc6"],
                    "auto_security_group": True
                }
            ],
            "plugin_name": "vanilla",
            "status": "Active",
            "updated": "2013-07-07T19:06:24",
            "user_keypair_id": "stack"
        }
    ]
}
```


3.5.8 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!

3.6 How to Participate

3.6.1 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`

3.6.2 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

3.6.3 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/#/

3.7 How to build Oozie

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

3.7.1 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

3.8 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.

3.8.1 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" --autoger
```

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.

3.8.2 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
```

3.8.3 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'.

3.8.4 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
```

3.9 Sahara Testing

We have a bunch of different tests for Sahara.

3.9.1 Unit Tests

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

3.9.2 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*.

3.9.3 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

3.9.4 Selenium Integration tests

We have a bunch of Selenium-based UI tests for Sahara UI in https://git.openstack.org/cgit/openstack/sahara-dashboard repository. 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

3.10 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.

3.11 Plugin SPI

3.11.1 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: [“1.2.1”, “2.3.0”, “2.4.1”]

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_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

get_edp_engine(cluster, job_type)

Returns an EDP job engine object that supports the specified job_type on the given cluster, or None if there is no support. The EDP job engine object returned must implement the interface described in *Elastic Data Processing (EDP) SPI*. The job_type is a String matching one of the job types listed in *Job Types*.

Returns: an EDP job engine object or 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

3.12 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.

3.12.1 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.

3.12.2 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.

3.12.3 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

3.12.4 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.

3.12.5 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.

3.12.6 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.

3.13 Elastic Data Processing (EDP) SPI

EDP job engine objects provide methods for creating, monitoring, and terminating jobs on Sahara clusters. Provisioning plugins that support EDP must return an EDP job engine object from the *get_edp_engine(cluster, job_type)* method described in *Plugin SPI*.

Sahara provides subclasses of the base job engine interface that support EDP on clusters running Oozie or on Spark standalone clusters. These are described below.

3.13.1 Job Types

Some of the methods below test job type. Sahara supports the following string values for job types:

- Hive
- Java
- Pig
- MapReduce
- MapReduce.Streaming
- Spark

Note, constants for job types are defined in *sahara.utils.edp*

3.13.2 Job Status Values

Several of the methods below return a job status value. A job status value is a dictionary of the form:

{'status': *job_status_value*}

where *job_status_value* is one of the following string values:

- DONEWITHERROR
- FAILED
- KILLED
- PENDING
- RUNNING
- SUCCEEDED

Note, constants for job status are defined in *sahara.utils.edp*

3.13.3 EDP Job Engine Interface

The sahara.service.edp.base_engine.JobEngine class is an abstract class with the following interface:

cancel_job(job_execution)

Stops the running job whose id is stored in the job_execution object.

Returns: None if the operation was unsuccessful or an updated job status value

get_job_status(job_execution)

Returns the current status of the job whose id is stored in the job_execution object.

Returns: a job status value

run_job(job_execution)

Starts the job described by the job_execution object

Returns: a tuple of the form (job_id, job_status_value, job_extra_info)

- *job_id* is required and must be a string that allows the EDP engine to

uniquely identify the job. * *job_status_value* may be None or a job status value * *job_extra_info* may be None or optionally a dictionary that the EDP engine uses to store extra information on the job_execution_object.

validate_job_execution(cluster, job, data)

Checks whether or not the job can run on the cluster with the specified data. Data contains values passed to the */jobs/<job_id>/execute* REST API method during job launch. If the job cannot run for any reason, including job configuration, cluster configuration, or invalid data, this method should raise an exception.

Returns: None

get_possible_job_config(job_type)

Returns hints used by the Sahara UI to prompt users for values when configuring and launching a job. Note that no hints are required.

See *Elastic Data Processing (EDP)* for more information on how configuration values, parameters, and arguments are used by different job types.

Returns: a dictionary of the following form, containing hints for configs, parameters, and arguments for the job type:

{'job_config': {'configs': [], 'params': {}, 'args': []}}

- *args* is a list of strings
- *params* contains simple key/value pairs
- each item in *configs* is a dictionary with entries for 'name' (required), 'value', and 'description'

get_supported_job_types()

This method returns the job types that the engine supports. Not all engines will support all job types.

Returns: a list of job types supported by the engine

3.13.4 Oozie Job Engine Interface

The sahara.service.edp.oozie.engine.OozieJobEngine class is derived from JobEngine. It provides implementations for all of the methods in the base interface but adds a few more abstract methods.

Note, the *validate_job_execution(cluster, job, data)* method does basic checks on the job configuration but probably should be overloaded to include additional checks on the cluster configuration. For example, the job engines for plugins that support Oozie add checks to make sure that the Oozie service is up and running.

get_hdfs_user()

Oozie uses HDFS to distribute job files. This method gives the name of the account that is used on the data nodes to access HDFS (such as 'hadoop' or 'hdfs'). The Oozie job engine expects that HDFS contains a directory for this user under */user/*

Returns: a string giving the username for the account used to access HDFS on the cluster.

create_hdfs_dir(remote, dir_name)

The remote object *remote* references a node in the cluster. This method creates the HDFS directory *dir_name* under the user specified by *get_hdfs_user()* in the HDFS accessible from the specified node. For example, if the HDFS user is 'hadoop' and the dir_name is 'test' this method would create '/user/hadoop/test'.

The reason that this method is broken out in the interface as an abstract method is that different versions of Hadoop treat path creation differently.

Returns: None

get_oozie_server_uri(cluster)

Returns the full URI for the Oozie server, for example *http://my_oozie_host:11000/oozie*. This URI is used by an Oozie client to send commands and queries to the Oozie server.

Returns: a string giving the Oozie server URI.

get_oozie_server(self, cluster)

Returns the node instance for the host in the cluster running the Oozie server

Returns: a node instance

get_name_node_uri(self, cluster)

Returns the full URI for the Hadoop NameNode, for example *http://master_node:8020*.

Returns: a string giving the NameNode URI.

get_resource_manager_uri(self, cluster)

Returns the full URI for the Hadoop JobTracker for Hadoop version 1 or the Hadoop ResourceManager for Hadoop version 2.

Returns: a string giving the JobTrakcer or ResourceManager URI

3.13.5 Spark Job Engine

The sahara.service.edp.spark.engine.SparkJobEngine class provides a full EDP implementation for Spark standalone clusters.

Note, the *validate_job_execution(cluster, job, data)* method does basic checks on the job configuration but probably should be overloaded to include additional checks on the cluster configuration. For example, the job engine returned by the Spark plugin checks that the Spark version is >= 1.0.0 to ensure that *spark-submit* is available.

3.14 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

3.14.1 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.

3.14.2 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.

3.14.3 Deleting an existing Cluster

1. Deleting

The only step, that releases all Cluster's resources and removes it form database.

3.14.4 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

3.15 Project hosting

Launchpad hosts the Sahara project. The Sahara project homepage on Launchpad is http://launchpad.net/sahara.

3.15.1 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*)

3.15.2 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

3.15.3 Bug tracking

Report Sahara bugs at https://bugs.launchpad.net/sahara

3.15.4 Feature requests (Blueprints)

Sahara uses Launchpad Blueprints to track feature requests. Blueprints are at https://blueprints.launchpad.net/sahara.

3.15.5 Technical support

Sahara uses Ask OpenStack to track Sahara technical support questions. Questions related to Sahara should be tagged with 'sahara'.

3.16 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.

3.17 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 the internal Mirantis OpenStack Lab. Jenkins keeps a pool of VMs to run tests in parallel. Even with the pool of VMs integration testing may take a while. Jenkins is controlled for the most part by Zuul which determines what jobs are run when. Zuul status is available by address: Zuul Status. For more information see: SaharaCI.

The integration tests result is +1 or -1 to *Verify* column in a code review from *savanna-ci* user.

/v1.0

/v1.1