Cue Specs

Release

OpenStack Cue Team

August 26, 2015

Contents

Cue provides Message Broker provisioning services for OpenStack:

- API management for message brokers - provides properly configured message brokers on demand
- Support for RabbitMQ today, Kafka POC coming soon
- Plug-in model for open source and proprietary brokers
- Focused on lifecycle actions such as rolling patches/upgrades, scale out/in, monitoring, clustering, and self-healing

CHAPTER 1

What is Cue?

Cue is a message broker provisioning service for Openstack. Its goal is to simplify and automate the complex tasks of provisioning, management, and administration of message brokers. Cue is designed to provide API, CLI, and UI based management of open source brokers such as RabbitMQ and Kafka. Because it's designed to be pluggable, proprietary message brokers can also be delivered as a service through Cue. The service will provide resource isolation at the VM level, and does not attempt to implement multi-tenancy at the message broker layer.

There are two personas to keep in mind when thinking about Cue. One is the Cloud Operator. The Cloud Operator installs and operates Cue, alongside the rest of OpenStack. This person cares about Cue installation and upgrades, along with broker level availability and versioning capabilities. The second persona is the application developer. This person provisions message broker clusters from the Cue API, CLI, or Horizon console. Alternatively, the application developer consumes Cue through a Platform as a Service product such as Cloud Foundry. In this scenario there is already a Cue provisioned cluster and the developer simply specifies that a queue of a certain name is needed in the application's manifest file. At deploy time, the queue is created directly on the message broker itself using a Cloud Foundry service broker.

CHAPTER 2

Why Cue?

Messaging is a common development pattern for building loosely coupled distributed systems. Messaging systems act as glue between independent applications. Dozens of off-the-shelf products exist that implement messaging and queuing semantics, many of which implement open protocols such as AMQP 0.9 and 1.0.

There is a significant operational burden associated with the lifecycle management of message brokers in an enterprise. Not to mention the time associated with ensuring the broker is deployed in a cloud native pattern, assuming failure of underlying infrastructure. Cue aims to simplify the provisioning and management of messaging systems, providing high availability and auto-healing capabilities for both the cloud operator and end user, while providing secure tenant-level isolation.

The main goal of this service is to simplify the end user application development lifecycle for both legacy and "cloud native" applications, allowing the developer to focus on their application, instead of the underlying middleware services.

CHAPTER 3

Getting Started

3.1 Development Environment with Vagrant

Cue is comprised of three main components *cue-api*, *cue-worker* and *cue-manage*. The Cue project includes a Vagrant configuration which deploys the Cue service and related scenario tests as part of a DevStack installation. This configuration allows new developers to get up and running quickly and efficiently.

This guide will walk you through setting up a Vagrant VM with devstack and Cue installed.

3.1.1 Development Environment

The Vagrant configuration allows the deployment of Cue service into DevStack. This environment provides a developer with a quick and easy way to run Cue with latest changes locally, run integration and scenario tests.

Deploying Cue Through Vagrant

1. Clone the Cue repo from GitHub

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

2. Startup Vagrant VM

```
$ cd cue/contrib/vagrant
$ vagrant up ubuntu
```

3. SSH into Vagrant VM

```
$ vagrant ssh ubuntu
```

4. Install Devstack

```
$ cd devstack
$ ./stack.sh
```

You are now in the Vagrant VM with DevStack installed/configured with Cue API, and Cue Worker.

Unit and Functional Testing

Unit are located in: cue/cue/tests/unit

Functional tests are located in: cue/cue/tests/functional

To run all unit and function tests, execute 'tox' from Cue project folder:

```
$ cd cue
$ tox
```

Integration Tests

Integration tests verify Cue through calling the REST API directly. These tests make use of the Tempest framework and are located in: cue/tests/integration

To run all integration tests, ssh into the Vagrant VM with DevStack/Cue installation (above) and run the following script:

```
$ ./cue/tests/integration/run_tests.sh
```

Scenario Tests

Scenario tests verify Cue through the Python Cue Client. These tests make use of Rally Benchmark framework and are located in: cue/rally-jobs

To run all scenario tests, ssh into the Vagrant VM with DevStack/Cue installation (above) and run the following script:

```
$ rally task start --task ~/cue/rally-jobs/rabbitmq-scenarios.yaml
```

CHAPTER 4

Getting Involved

4.1 #openstack-cue

There is an active IRC channel at irc://freenode.net/#openstack-cue, where many of the cue contributors can be found, as well as users from various organisations.

4.2 Weekly meetings

There is a weekly irc meet. The agenda, date/time and other details are listed at Cue meetings wiki page.

4.3 Contributing

We welcome fixes, extensions, documentation, pretty much anything that helps improve Cue, contributing is easy & follows the standard OpenStack Gerrit workflow, if you're looking for something to do, you could always checkout the blueprint & bug lists.

Assuming you've already got a working *Development Environment*, here's a quick summary:

Install the git-review package to make life easier

```
pip install git-review
```

Branch, work, & submit:

```
# cut a new branch, tracking master
git checkout --track -b bug/id origin/master
# work work work
git add stuff
git commit
# rebase/squash to a single commit before submitting
git rebase -i
# submit
git-review
```

4.4 Coding Standards

Cue uses the OpenStack flake8 coding standards guidelines. These are stricter than pep8, and are run by gerrit on every commit.

You can use tox to check your code locally by running

```
# For just flake8 tests
tox -e flake8
# For tests + flake8
tox
```

CHAPTER 5

Installation Guide

5.1 Installing Cue on Ubuntu

This install guide provides details on how to install on a Ubuntu based image with the necessary dependencies and required configuration.

5.1.1 Architecture

Installation of Cue requires four (2) primary Cue components, two (2) components that are external dependencies for Cue, and three (3) optional components.

Required Cue Components

- Cue API
- Cue Worker

Required Dependencies

- MySQL
- Zookeeper

Optional Components

- Cue Command-line Client (python-cueclient)
- Cue Dashboard (cue-dashboard)
- Openstack Client (python-openstackclient)

Additional Setup

Additional setup beyond installing Cue services is required in Keystone to allow authentication of user credentials against Keystone. Also the Keystone service catalog must be updated to include Cue's service endpoints.

5.1.2 Prerequisites

Install

```
$ sudo apt-get install zookeeper zookeeperd python-mysqldb
```

MySQL

Note: The following commands should be done using the mysql command line or similar.

Create the MySQL user

```
$ GRANT ALL ON cue.* TO 'cue'@'localhost' IDENTIFIED BY 'cue'
```

Create the database

```
$ CREATE DATABASE cue
```

5.1.3 Installing using Source (Git)

1. Install pre-requisites:

```
$ sudo apt-get install git python-dev python-pip
$ sudo apt-get build-dep python-lxml
```

2. Clone the repository:

```
$ git clone git://github.com/openstack/cue cue
```

3. Change directory to the newly cloned repository

```
$ cd cue
```

4. Install all dependencies using pip

```
$ sudo pip install -r requirements.txt
$ sudo pip install MySQL-python
```

5. Install Cue:

```
$ sudo python setup.py develop
```

6. Copy over configuration files

```
$ sudo cp -R etc/cue /etc/
$ ls /etc/cue/*.sample | while read f; do sudo cp $f $(echo $f | sed "s/.sample$//g"); done
```

Create directories

Since we are not running packages some directories are not created for us.

```
$ sudo mkdir /var/lib/cue /var/log/cue
# Needed if you are running cue as a non root user.
$ sudo chown cue_user /var/lib/cue /var/log/cue
```

5.1.4 Configuring

Register Cue with Keystone

1. Create new user for cue service

```
keystone user-create --name cue --tenant <tenant_uuid> --pass <password
```

2. Add admin role for cue_admin user

```
keystone user-role-add --user cue_admin --tenant cue_admin_service --role=admin
```

Add Cue Service Endpoint to Keystone

1. Add cue service to keystone

```
keystone service-create --type message_broker --name cue --description "Message Broker provisioning s
```

2. Create a new endpoint for the cue service in keystone

```
keystone endpoint-create --region RegionOne --service <cue_service_uuid> --publicurl http://<cue_api_
```

Create Message Broker Image

The Cue service makes use of custom Ubuntu images with required messaging broker (e.g. RabbitMQ) installed. Building images uses the Tripleo *diskimage-builder* tools. Image elements for the RabbitMQ image are found in cue/contrib/image-elements/, you will also find the 'image-build-rabbitmq.sh' script which will build a custom image compatible with Cue.

The images are based on the latest base Ubuntu cloud image with the following elements:

- os-apply-config
- os-refresh-config
- ntp
- hosts
- cue-rabbitmq-base (cue/contrib/image-elements/cue-rabbit-base)
- ifmetric (cue/contrib/image-elements/ifmetric)

Note: building images will require a machine with more than 4GB of memory.

Once the image is built, it must be uploaded to Glance (disk format is qcow2) and message broker details added to Cue database through cue-management.

1. Create new Message Broker and set it as active broker

```
$ cue-manage --config-file etc/cue/cue.conf broker add <name> true
```

2. Add metadata indicating image id (created above) for new Message Broker

```
$ cue-manage --config-file etc/cue/cue.conf broker add_metadata <broker-uuid> --image <image-uuid>
```

Cue Config

```
$ sudo editor /etc/cue/cue.conf
```

Copy or mirror the configuration from this sample file here:

```
[DEFAULT]
management_network_id = <uuid>
os_security_group = <uuid>
policy_file = /etc/cue/policy.json
# Show more verbose log output (sets INFO log level output)
verbose = True
# Show debugging output in logs (sets DEBUG log level output)
debug = True
# Log levels for Zookeeper client and Stevedore
default_log_levels = kazoo.client=INFO,stevedore=INFO

[database]
# Database connection string - to configure options for a given implementation
connection = mysql://<user>:<password>@<ip-address>/cue

[api]
# Cue REST API specific configuration
api_port = 8795
api_host = <rest-api-ip-address>
auth_strategy = keystone

[taskflow]
# Zookeeper host node
zk_hosts=<zookeeper-ip-address>

[openstack]
# Credentials used by Cue to access OpenStack services
os_password = <password>
os_username = <username>
os_auth_url = http://192.168.131.199:35357/v3
os_auth_version = 3
os_project_name = <project-name>
os_project_domain_name = <project-domain-name>
os_user_domain_name = <user-domain-name>

[keystone_authtoken]
# Credentials used by Cue for KeyStone authentication
auth_url = http://<keystone-ip-address>:35357
auth_plugin = <auth-password>
project_name = service
password = <password>
username = <username>
```

More details on configuration values:

Configuration Options

[DEFAULT]

Parameter	Default	Note
management_network_id	None	The id representing the management network
os_security_group	None	The default Security to access clustered VMs
default_broker_name	rabbitmq	The name of the default broker image
rabbit_port	5672	RabbitMQ AMQP port on clustered VMs
policy_file	/etc/cue/policy.json	JSON file representing policy
verbose	false	Print more verbose output
debug	false	Print debugging output
state-path	/var/lib/cue	Top-level directory for maintaining cue's state

[database]

Parameter	Default	Note
connection	None	The SQLAlchemy connection string to database

[api]

Parameter	Default	Note
host_ip	0.0.0.0	The listen IP for the Cue API server
port	8795	The port for the Cue API server
max_limit	1000	The maximum number of items returned in a single response
max_cluster_size	10	Maximum number of nodes in a cluster
auth_strategy	keystone	Method to use for authentication: noauth or keystone
pecan_debug	False	Pecan HTML Debug Interface

[taskflow]

Parameter	Default	Note
persistence_connection	None	Persistence connection
zk_hosts	localhost	Zookeeper jobboard hosts
zk_path	/cue/taskflow	Zookeeper path for jobs
zk_timeout	10	Zookeeper operations timeout
jobboard_name	'cue'	Board name
engine_type	'serial'	Engine type
cluster_node_check_timeout	10	Number of seconds between node status checks
cluster_node_check_max_count	30	Number of times to check a node for status

[openstack]

Parameter	Default	Note
os_region_name	None	Region name
os_username	None	Openstack Username
os_password	None	Openstack Password
os_auth_url	None	Openstack Authentication (Identity) URL
os_auth_version	None	Openstack Authentication (Identity) Version
os_project_name	None	Openstack Project Name
os_project_domain_name	None	Openstack Project Domain Name
os_user_domain_name	None	Openstack User Domain Name
os_key_name	None	SSH key to be provisioned to cue VMs
os_availability_zone	None	Default availability zone to provision cue VMs

[keystone_authtoken]

Parameter	Default	Note
auth_url	None	The URL to Keystone Identity Service
auth_plugin	None	Name of the plugin to load
project_name	None	Project name accessing Keystone (usually 'service')
username	None	Username for accessing Keystone
password	None	password for accessing keystone

Sync Database schemas

Initialize database schema for Cue

```
$ cue-manage --config-file /etc/cue/cue.conf database upgrade
```

Notes:

- magement_network_id must be different than provided user network id through API.

5.1.5 Starting the services

Worker

```
$ cue-worker --config-file /etc/cue/cue.conf
```

API

The API can be started as is (through shell) or can be started behind Apache. Starting the API behind Apache is the recommended method for running the API (section below).

Starting with command shell:

```
$ cue-api --config-file /etc/cue/cue.conf
```

5.1.6 Running Cue API behind Apache2

Note: In this howto we explain how to setup cue-api to run behind a Apache2 instance vs as a process of it's own. We will assume that cue is available under /opt/stack/cue as it is in devstack.

Symlink app.wsgi to /var/www

```
$ sudo mkdir /var/www/cue
$ sudo ln -s /opt/stack/cue/cue/api/app.wsgi /var/www/cue
```

Setup Apache2 config

```
$ sudo cp /opt/cue/etc/apache2/cue.conf /etc/apache2/sites-available
$ sudo a2ensite cue
$ sudo service apache2 reload
```

You should now have cue-api running under Apache2!

5.2 Recommended Deployment

5.2.1 Overview

Cue currently does not support creation of broker VMs in user tenants. This is to ensure the overall security of Cue in a multi-tenant environment as well as to prevent direct modification of Broker VMs by tenants. Cue is a full life cycle management solution for Message Brokers, therefore it requires final control over the Broker VMs.

Cue can be installed in a very flat network with all components and interfaces existing in a single network plane. However, for the sake of security and isolation, it is recommended that separate networks and firewall domains/zones be created. Below is a diagram showing the currently recommended deployment model.

Networks

- API Network - This is the network that the Cue API will expose its API endpoint to. This network will need to be routable/reachable by all Cue users and will likely need to be a "public" network.
- Cue Control Plane Network - This is the network that all Cue Control Plane components (API, MySQL, Zookeeper, Worker) will use to communicate with each other. It is recommended this network be isolated from all other traffic. This network can optionally be connected to a "service" network that provides access to shared services like monitoring.
- Management Network - This is the network that the Cue Worker and all Broker VMs attach to. The Cue Worker connects to Broker VMs through the management network to monitor and control each Broker VM.
- Tenant Network - The tenant network is not a single network, it is any network that the Cue user specifies. Cue creates and attaches a port from the tenant network to the Broker VM in order to provide access to Message Broker services.
- External/Services Network - This is a network that the Cue Control Plane might optionally be attached to. This network, if attached to the Cue Control Plane, should be isolated from Tenant/User traffic in order to prevent possible attacks to the Cue Control Plane.

Security Groups

All security groups should have a default deny policy and allow only the specific ports as specified below:

- API - The API listens on the Cue service port (8795 by default) and must allow incoming requests to this port from the API network.
- Control Plane - The Cue API and Cue Worker access MySQL and Zookeeper for persisting data and coordination. It is highly recommended that MySQL and Zookeeper both be clustered, which requires that respective clustering ports be opened to members of each respective cluster. If the firewall is implemented with Security Groups, a group rule can be used to allow access to all members of the control plane, or explicit rules can be used to limit access to specific ports.
- Message Broker Internal - There exists a unique "Message Broker Internal" security domain/zone for each Message Broker type that is supported by Cue. The Message Broker VM must allow access from Cue Worker in order to allow Cue to manage the Message Broker VM. Depending on the Message Broker, it will also need to allow cluster traffic between the Message Broker VMs to facilitate clustering.
- Message Broker Public - There exists a unique "Message Broker Public" security domain/zone for each Message Broker type that is supported by Cue. The Message Broker VM must allow access to the public endpoints for the Message Broker service it is providing.

Cue Tenant and Users

Cue requires its own Tenant, which is used to create all Broker VMs.

Cue uses two logically separate users to interact with Openstack:

- Cue Service User
- Cue Tenant User

Cue Service User is used by Cue API to communicate with Keystone for authenticating user tokens. The Cue Service User must have an appropriate Keystone role to be able to authenticate Keystone auth tokens.

Cue Tenant User is used by Cue Worker to communicate with Openstack services like nova, neutron, and cinder, to create and manage Broker VMs. The Cue Tenant User must have appropriate access to the Cue Tenant for creating VMs. The Cue Tenant User must also have an appropriate neutron role to be able to create ports on networks not owned by the Cue Tenant.

It is also recommended that the Cue Tenant User has appropriate quotas set for Nova, Cinder, and Neutron. These settings reflect the capacity available to Cue for provisioning Broker Clusters.

5.3 Dashboard

Cue provides a Horizon Dashboard plugin. The plugin can be found at the cue-dashboard repo with accompanying installation instructions.

CHAPTER 6

Configuration Options

6.1 [DEFAULT]

Parameter	Default	Note
management_network_id	None	The id representing the management network
os_security_group	None	The default Security to access clustered VMs
default_broker_name	rabbitmq	The name of the default broker image
rabbit_port	5672	RabbitMQ AMQP port on clustered VMs
policy_file	/etc/cue/policy.json	JSON file representing policy
verbose	false	Print more verbose output
debug	false	Print debugging output
state-path	/var/lib/cue	Top-level directory for maintaining cue's state

6.2 [database]

Parameter	Default	Note
connection	None	The SQLAlchemy connection string to database

6.3 [api]

Parameter	Default	Note
host_ip	0.0.0.0	The listen IP for the Cue API server
port	8795	The port for the Cue API server
max_limit	1000	The maximum number of items returned in a single response
max_cluster_size	10	Maximum number of nodes in a cluster
auth_strategy	keystone	Method to use for authentication: noauth or keystone
pecan_debug	False	Pecan HTML Debug Interface

6.4 [taskflow]

Parameter	Default	Note
persistence_connection	None	Persistence connection
zk_hosts	localhost	Zookeeper jobboard hosts
zk_path	/cue/taskflow	Zookeeper path for jobs
zk_timeout	10	Zookeeper operations timeout
jobboard_name	'cue'	Board name
engine_type	'serial'	Engine type
cluster_node_check_timeout	10	Number of seconds between node status checks
cluster_node_check_max_count	30	Number of times to check a node for status

6.5 [openstack]

Parameter	Default	Note
os_region_name	None	Region name
os_username	None	Openstack Username
os_password	None	Openstack Password
os_auth_url	None	Openstack Authentication (Identity) URL
os_auth_version	None	Openstack Authentication (Identity) Version
os_project_name	None	Openstack Project Name
os_project_domain_name	None	Openstack Project Domain Name
os_user_domain_name	None	Openstack User Domain Name
os_key_name	None	SSH key to be provisioned to cue VMs
os_availability_zone	None	Default availability zone to provision cue VMs

6.6 [keystone_authtoken]

Parameter	Default	Note
auth_url	None	The URL to Keystone Identity Service
auth_plugin	None	Name of the plugin to load
project_name	None	Project name accessing Keystone (usually 'service')
username	None	Username for accessing Keystone
password	None	password for accessing keystone

CHAPTER 7

Developer Guide

7.1 Cue Dev Guide - Overview

7.1.1 Summary

The developer guide provides details on how to get Cue up and running quickly. This guide is divided into two parts; the first describes how to install DevStack with Cue. The second part describes how to use of the Vagrant configuration found in project Cue.

DevStack with Cue:

- experiment with the Cue service

Vagrant VM with DevStack and Cue:

- used as part of a development environment
- allows user to quickly verify changes
- run integration and scenario tests

7.2 DevStack

Instructions on how to install Cue as part of a DevStack deployment.

7.2.1 Instructions

1. **Get a clean Ubuntu 14.04 VM. DevStack "takes over". Don't use your desktop!** Note: Ensure VM has at least 8GB of Memory.
2. Clone Cue and DevStack inside the VM:

```
$ git clone https://git.openstack.org/openstack-dev/devstack
$ git clone https://github.com/openstack/cue.git
```

3. Install the Cue extension for DevStack:

```
$ cd cue/contrib/devstack
$ ./install.sh
```

4. Copy local.conf and local.sh from cue/contrib/devstack:

```
$ cp local.* ../../../devstack/
```

5. Run DevStack:

```
$ cd ../../../devstack
$ ./stack.sh
```

6. Enter the screen sessions "shell" window:

```
$ ./rejoin-stack.sh
```

   Then press Ctrl+A followed by d to exit

7. Load desired credentials into the shell:

```
$ source openrc admin admin # For the admin user, admin tenant
$ source openrc admin demo  # For the admin user, demo tenant
$ source openrc demo demo   # For the demo user, demo tenant
```

8. Try out the cue client:

```
$ openstack message-broker cluster create --name cluster_01 --nic d5c35f43-4e8e-4264-9c8a-21c2f0
+------------+------------------------------------------+
| Field      | Value                                    |
+------------+------------------------------------------+
| created_at | 2015-06-02T21:29:15+00:00                |
| endpoints | []                                       |
| flavor     | 8795                                     |
| id         | b7ed9907-2d37-41e6-b70c-22eb1ea44777     |
| name       | cluster_01                               |
| network_id | [u'd5c35f43-4e8e-4264-9c8a-21c2f0a358e8'] |
| size       | 1                                        |
| status     | BUILDING                                 |
+------------+------------------------------------------+

$ openstack message-broker cluster list
+--------------------------------------+------------+----------+------------+
| id                                   | name       | status   | endpoints |
+--------------------------------------+------------+----------+------------+
| b7ed9907-2d37-41e6-b70c-22eb1ea44777 | cluster_01 | BUILDING | []         |
+--------------------------------------+------------+----------+------------+

$ openstack message-broker cluster show b7ed9907-2d37-41e6-b70c-22eb1ea44777
+------------+--------------------------------------------------+
| Field      | Value                                            |
+------------+--------------------------------------------------+
| created_at | 2015-06-02T21:29:15+00:00                        |
| endpoints | [{u'type': u'AMQP', u'uri': u'10.0.0.5:5672'}] |
| flavor     | 8795                                             |
| id         | b7ed9907-2d37-41e6-b70c-22eb1ea44777             |
| name       | cluster_01                                       |
| network_id | [u'd5c35f43-4e8e-4264-9c8a-21c2f0a358e8']        |
| size       | 1                                                |
| status     | ACTIVE                                           |
| updated_at | 2015-06-02T21:29:18+00:00                        |
+------------+--------------------------------------------------+
```

7.3 Development Environment with Vagrant

Cue is comprised of three main components *cue-api*, *cue-worker* and *cue-manage*. The Cue project includes a Vagrant configuration which deploys the Cue service and related scenario tests as part of a DevStack installation. This configuration allows new developers to get up and running quickly and efficiently.

This guide will walk you through setting up a Vagrant VM with devstack and Cue installed.

7.3.1 Development Environment

The Vagrant configuration allows the deployment of Cue service into DevStack. This environment provides a developer with a quick and easy way to run Cue with latest changes locally, run integration and scenario tests.

Deploying Cue Through Vagrant

1. Clone the Cue repo from GitHub

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

2. Startup Vagrant VM

```
$ cd cue/contrib/vagrant
$ vagrant up ubuntu
```

3. SSH into Vagrant VM

```
$ vagrant ssh ubuntu
```

4. Install Devstack

```
$ cd devstack
$ ./stack.sh
```

You are now in the Vagrant VM with DevStack installed/configured with Cue API, and Cue Worker.

Unit and Functional Testing

Unit are located in: cue/cue/tests/unit

Functional tests are located in: cue/cue/tests/functional

To run all unit and function tests, execute 'tox' from Cue project folder:

```
$ cd cue
$ tox
```

Integration Tests

Integration tests verify Cue through calling the REST API directly. These tests make use of the Tempest framework and are located in: cue/tests/integration

To run all integration tests, ssh into the Vagrant VM with DevStack/Cue installation (above) and run the following script:

```
$ ./cue/tests/integration/run_tests.sh
```

Scenario Tests

Scenario tests verify Cue through the Python Cue Client. These tests make use of Rally Benchmark framework and are located in: cue/rally-jobs

To run all scenario tests, ssh into the Vagrant VM with DevStack/Cue installation (above) and run the following script:

```
$ rally task start --task ~/cue/rally-jobs/rabbitmq-scenarios.yaml
```

CHAPTER 8

REST API Documentation

The Cue API has currently one version versions - V1.

8.1 V1 API

8.1.1 Clusters

A Cluster is a representation of a collection of message broker nodes with respective endpoints (e.g. RabbitMQ Cluster). **Create Cluster**

POST /clusters

Create a new Cluster.

Example request:

```
POST /clusters HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json

{
   "name": "Example Cluster",
   "size": 3,
   "flavor": "101",
   "volume_size": "1048576"
   "network_id": [
       "ea540512-4d4a-4c46-9ebd-4fafe4a54a2d"
   ]
   "auth_credential": {
       "type": "PLAIN",
       "token": {
           "username": "rabbit_user"
           "password": "super_secret_password"
       }
   }
}
```

Example response:

```
HTTP/1.1 201 Created
Location: http://127.0.0.1:8795/v1/clusters/2c3c66ba-721b-4443-bc81-55d986848c68
```

```
Content-Type: application/json; charset=UTF-8

{
    "status": "BUILDING",
    "name": "Example Cluster",
    "network_id": [
        "ea540512-4d4a-4c46-9ebd-4fafe4a54a2d"
    ],
    "created_at": "2015-06-03T21:49:49+00:00",
    "volume_size": 1048576,
    "endpoints": [],
    "flavor": "101",
    "id": "2c3c66ba-721b-4443-bc81-55d986848c68",
    "size": 3
}
```

Form Parameters

- **status** – status of the cluster
- **name** – name of the cluster
- **network_id** – a list of UUID of network id's
- **created_at** – create cluster request received timestamp
- **volume_size** – volume size used for each node
- **endpoints** – list of endpoints for each node
- **flavor** – node flavor
- **id** – the UUID of the cluster
- **size** – size of the cluster
- **auth_credential** – Authentication credentials

Status Codes

- 201 Created – Created
- 400 Bad Request – Bad Request
- 401 Unauthorized – Access Denied

Get Cluster

GET /clusters/(**uuid:** *id*)

Get a specific Cluster using the Cluster's uuid id.

Example request:

```
GET /clusters/2c3c66ba-721b-4443-bc81-55d986848c68 HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
```

```
{
    "status": "ACTIVE",
    "name": "Example Cluster",
    "network_id": [
        "c6958944-7ef0-4929-9625-7f924bb2610c"
    ],
    "created_at": "2015-06-03T22:44:17+00:00",
    "updated_at": "2015-06-03T22:47:15+00:00",
    "volume_size": 1048576,
    "endpoints": [
        {
            "type": "AMQP",
            "uri": "10.0.0.9:5672"
        },
        {
            "type": "AMQP",
            "uri": "10.0.0.11:5672"
        },
        {
            "type": "AMQP",
            "uri": "10.0.0.10:5672"
        }
    ],
    "flavor": "8795",
    "id": "2c3c66ba-721b-4443-bc81-55d986848c68",
    "size": 3
}
```

Form Parameters

- **updated_at** – cluster last updated at timestamp

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request

List Clusters

GET /clusters

Lists all clusters

Example request:

```
GET /servers HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

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

{
  "status": "ACTIVE",
  "name": "Example Cluster",
  "network_id": [
```

```
            "c6958944-7ef0-4929-9625-7f924bb2610c"
        ],
        "created_at": "2015-06-03T22:44:17+00:00",
        "updated_at": "2015-06-04T00:31:16+00:00",
        "volume_size": 1048576,
        "endpoints": [
            {
                "type": "AMQP",
                "uri": "10.0.0.9:5672"
            },
            {
                "type": "AMQP",
                "uri": "10.0.0.11:5672"
            },
            {
                "type": "AMQP",
                "uri": "10.0.0.10:5672"
            }
        ],
        "flavor": "8795",
        "id": "2c3c66ba-721b-4443-bc81-55d986848c68",
        "size": 3
    },
    {
        "status": "DELETED",
        "name": "cluster_01",
        "network_id": [
            "ba013641-8b54-40a5-801d-a7839690e272"
        ],
        "created_at": "2015-05-13T21:23:15+00:00",
        "updated_at": "2015-05-13T21:30:15+00:00",
        "endpoints": [
            {
                "type": "AMQP",
                "uri": "10.0.0.7:5672"
            }
        ],
        "flavor": "8795",
        "id": "85a63cac-9bf7-4ef7-962d-dd51bde0b29b",
        "size": 1
    },
```


Status Codes

- 200 OK – Success
- 401 Unauthorized – Access Denied

Delete Cluster

DELETE /clusters/(**uuid:** *id*)

Delete a cluster.

Example request:

```
DELETE /clusters HTTP/1.1
Accept: application/json
```

Example response:

```
HTTP/1.1 202 Accepted
```

Status Codes

- 400 Bad Request – Bad Request
- 204 No Content – Successfully Deleted

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

/clusters

C

D