Aodh Documentation

Release 14.1.1.dev3

OpenStack Foundation

Jan 25, 2024

CONTENTS

The Alarming service (aodh) project provides a service that enables the ability to trigger actions based on defined rules against metric or event data collected by Ceilometer or Gnocchi.

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Telemetry Alarming service overview

The Telemetry Alarming services trigger alarms when the collected metering or event data break the defined rules.

The Telemetry Alarming service consists of the following components:

An API server (`aodh-api`) Runs on one or more central management servers to provide access to the alarm information stored in the data store.

An alarm evaluator (`aodh-evaluator`) Runs on one or more central management servers to determine when alarms fire due to the associated statistic trend crossing a threshold over a sliding time window.

A notification listener (`aodh-listener`) Runs on a central management server and determines when to fire alarms. The alarms are generated based on defined rules against events, which are captured by the Telemetry Data Collection service's notification agents.

An alarm notifier (`aodh-notifier`) Runs on one or more central management servers to allow alarms to be set based on the threshold evaluation for a collection of samples.

These services communicate by using the OpenStack messaging bus.

1.2 Install and configure for openSUSE and SUSE Linux Enterprise

This section describes how to install and configure the Telemetry Alarming service, code-named aodh, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Compute, Image Service, Identity.

1.2.1 Prerequisites

Before you install and configure the Telemetry service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `aodh` database:

     ```
     CREATE DATABASE aodh;
     ```

   - Grant proper access to the `aodh` database:

     ```
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'localhost' \
       IDENTIFIED BY 'AODH_DBPASS';
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'%' \
       IDENTIFIED BY 'AODH_DBPASS';
     ```

     Replace `AODH_DBPASS` with a suitable password.

   - Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   - Create the `aodh` user:

     ```
     $ openstack user create --domain default \
       --password-prompt aodh
     User Password:
     Repeat User Password:
     +---------------------+----------------------------------+
     | Field               | Value                            |
     +---------------------+----------------------------------+
     | domain_id           | default                          |
     | enabled             | True                             |
     | id                  | b7657c9ea07a4556aef5d34cf70713a3 |
     | name                | aodh                             |
     | options             | {}                               |
     | password_expires_at | None                             |
     +---------------------+----------------------------------+
     ```

   - Add the `admin` role to the `aodh` user:

     ```
     $ openstack role add --project service --user aodh admin
     ```

Note: This command provides no output.

- Create the aodh service entity:

```
$ openstack service create --name aodh \
  --description "Telemetry" alarming
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Telemetry                        |
| enabled     | True                             |
| id          | 3405453b14da441ebb258edfeba96d83 |
| name        | aodh                             |
| type        | alarming                         |
+-------------+----------------------------------+
```

4. Create the Alarming service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  alarming public http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | public                           |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  alarming internal http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | internal                         |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+
```

(continues on next page)

(continued from previous page)

```
$ openstack endpoint create --region RegionOne \
  alarming admin http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | admin                            |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+
```

1.2.2 Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# zypper install openstack-aodh-api \
  openstack-aodh-evaluator openstack-aodh-notifier \
  openstack-aodh-listener openstack-aodh-expirer \
  python-aodhclient
```

2. Edit the `/etc/aodh/aodh.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

   ```
   [database]
   ...
   connection = mysql+pymysql://aodh:AODH_DBPASS@controller/aodh
   ```

   Replace `AODH_DBPASS` with the password you chose for the Telemetry Alarming module database. You must escape special characters such as `:`, `/`, `+`, and `@` in the connection string in accordance with RFC2396.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

   ```
   [DEFAULT]
   ...
   transport_url = rabbit://openstack:RABBIT_PASS@controller
   ```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

  ```
  [DEFAULT]
  ...
  auth_strategy = keystone

  [keystone_authtoken]
  ...
  www_authenticate_uri = http://controller:5000
  auth_url = http://controller:5000
  memcached_servers = controller:11211
  auth_type = password
  project_domain_id = default
  user_domain_id = default
  project_name = service
  username = aodh
  password = AODH_PASS
  ```

  Replace AODH_PASS with the password you chose for the aodh user in the Identity service.

- In the [service_credentials] section, configure service credentials:

  ```
  [service_credentials]
  ...
  auth_type = password
  auth_url = http://controller:5000/v3
  project_domain_id = default
  user_domain_id = default
  project_name = service
  username = aodh
  password = AODH_PASS
  interface = internalURL
  region_name = RegionOne
  ```

  Replace AODH_PASS with the password you chose for the aodh user in the Identity service.

3. In order to initialize the database please run the aodh-dbsync script.

1.2.3 Finalize installation

1. Start the Telemetry Alarming services and configure them to start when the system boots:

```
# systemctl enable openstack-aodh-api.service \
  openstack-aodh-evaluator.service \
  openstack-aodh-notifier.service \
  openstack-aodh-listener.service
# systemctl start openstack-aodh-api.service \
  openstack-aodh-evaluator.service \
```

(continues on next page)

(continued from previous page)

```
openstack-aodh-notifier.service \
openstack-aodh-listener.service
```

1.3 Install and configure for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure the Telemetry Alarming service, code-named aodh, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Compute, Image Service, Identity.

1.3.1 Prerequisites

Before you install and configure the Telemetry service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `aodh` database:

     ```
     CREATE DATABASE aodh;
     ```

   - Grant proper access to the `aodh` database:

     ```
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'localhost' \
       IDENTIFIED BY 'AODH_DBPASS';
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'%' \
       IDENTIFIED BY 'AODH_DBPASS';
     ```

     Replace `AODH_DBPASS` with a suitable password.

   - Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   - Create the `aodh` user:

     ```
     $ openstack user create --domain default \
       --password-prompt aodh
     User Password:
     Repeat User Password:
     +---------------------+----------------------------------+
     | Field               | Value                            |
     ```

(continues on next page)

(continued from previous page)

```
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | b7657c9ea07a4556aef5d34cf70713a3 |
| name                | aodh                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Add the admin role to the aodh user:

```
$ openstack role add --project service --user aodh admin
```

Note: This command provides no output.

- Create the aodh service entity:

```
$ openstack service create --name aodh \
  --description "Telemetry" alarming
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Telemetry                        |
| enabled     | True                             |
| id          | 3405453b14da441ebb258edfeba96d83 |
| name        | aodh                             |
| type        | alarming                         |
+-------------+----------------------------------+
```

4. Create the Alarming service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  alarming public http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | public                           |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  alarming internal http://controller:8042
```

(continues on next page)

(continued from previous page)

```
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | internal                         |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  alarming admin http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | admin                            |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+
```

1.3.2 Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# yum install openstack-aodh-api \
  openstack-aodh-evaluator openstack-aodh-notifier \
  openstack-aodh-listener openstack-aodh-expirer \
  python-aodhclient
```

2. Edit the `/etc/aodh/aodh.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
...
connection = mysql+pymysql://aodh:AODH_DBPASS@controller/aodh
```

Replace AODH_DBPASS with the password you chose for the Telemetry Alarming module database. You must escape special characters such as :, /, +, and @ in the connection string in accordance with RFC2396.

- In the [DEFAULT] section, configure RabbitMQ message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_id = default
user_domain_id = default
project_name = service
username = aodh
password = AODH_PASS
```

Replace AODH_PASS with the password you chose for the aodh user in the Identity service.

- In the [service_credentials] section, configure service credentials:

```
[service_credentials]
...
auth_type = password
auth_url = http://controller:5000/v3
project_domain_id = default
user_domain_id = default
project_name = service
username = aodh
password = AODH_PASS
interface = internalURL
region_name = RegionOne
```

Replace AODH_PASS with the password you chose for the aodh user in the Identity service.

3. In order to initialize the database please run the aodh-dbsync script.

1.3.3 Finalize installation

1. Start the Telemetry Alarming services and configure them to start when the system boots:

```
# systemctl enable openstack-aodh-api.service \
  openstack-aodh-evaluator.service \
  openstack-aodh-notifier.service \
  openstack-aodh-listener.service
# systemctl start openstack-aodh-api.service \
  openstack-aodh-evaluator.service \
  openstack-aodh-notifier.service \
  openstack-aodh-listener.service
```

1.4 Install and configure for Ubuntu

This section describes how to install and configure the Telemetry Alarming service, code-named aodh, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Compute, Image Service, Identity.

1.4.1 Prerequisites

Before you install and configure the Telemetry service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `aodh` database:

     ```
     CREATE DATABASE aodh;
     ```

   - Grant proper access to the `aodh` database:

     ```
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'localhost' \
       IDENTIFIED BY 'AODH_DBPASS';
     GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'%' \
       IDENTIFIED BY 'AODH_DBPASS';
     ```

     Replace `AODH_DBPASS` with a suitable password.

   - Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

- Create the aodh user:

```
$ openstack user create --domain default \
  --password-prompt aodh
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | b7657c9ea07a4556aef5d34cf70713a3 |
| name                | aodh                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Add the `admin` role to the `aodh` user:

```
$ openstack role add --project service --user aodh admin
```

Note: This command provides no output.

- Create the `aodh` service entity:

```
$ openstack service create --name aodh \
  --description "Telemetry" alarming
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Telemetry                        |
| enabled     | True                             |
| id          | 3405453b14da441ebb258edfeba96d83 |
| name        | aodh                             |
| type        | alarming                         |
+-------------+----------------------------------+
```

4. Create the Alarming service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  alarming public http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | public                           |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
```

(continues on next page)

(continued from previous page)

```
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  alarming internal http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | internal                         |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  alarming admin http://controller:8042
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 340be3625e9b4239a6415d034e98aace |
  | interface    | admin                            |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | 8c2c7f1b9b5049ea9e63757b5533e6d2 |
  | service_name | aodh                             |
  | service_type | alarming                         |
  | url          | http://controller:8042           |
  +--------------+----------------------------------+
```

1.4.2 Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# apt-get install aodh-api aodh-evaluator aodh-notifier \
  aodh-listener aodh-expirer python-aodhclient
```

2. Edit the `/etc/aodh/aodh.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

     ```
     [database]
     ...
     connection = mysql+pymysql://aodh:AODH_DBPASS@controller/aodh
     ```

     Replace `AODH_DBPASS` with the password you chose for the Telemetry Alarming module database. You must escape special characters such as `:`, `/`, `+`, and `@` in the connection string in accordance with RFC2396.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

     ```
     [DEFAULT]
     ...
     transport_url = rabbit://openstack:RABBIT_PASS@controller
     ```

     Replace `RABBIT_PASS` with the password you chose for the `openstack` account in `RabbitMQ`.

   - In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

     ```
     [DEFAULT]
     ...
     auth_strategy = keystone

     [keystone_authtoken]
     ...
     www_authenticate_uri = http://controller:5000
     auth_url = http://controller:5000
     memcached_servers = controller:11211
     auth_type = password
     project_domain_id = default
     user_domain_id = default
     project_name = service
     username = aodh
     password = AODH_PASS
     ```

     Replace `AODH_PASS` with the password you chose for the `aodh` user in the Identity service.

   - In the `[service_credentials]` section, configure service credentials:

     ```
     [service_credentials]
     ...
     auth_type = password
     auth_url = http://controller:5000/v3
     project_domain_id = default
     user_domain_id = default
     project_name = service
     username = aodh
     password = AODH_PASS
     interface = internalURL
     region_name = RegionOne
     ```

Replace `AODH_PASS` with the password you chose for the `aodh` user in the Identity service.

3. In order to initialize the database please run the `aodh-dbsync` script.

1.4.3 Finalize installation

1. Restart the Alarming services:

```
# service aodh-api restart
# service aodh-evaluator restart
# service aodh-notifier restart
# service aodh-listener restart
```

1.5 Next steps

Your OpenStack environment now includes the aodh service.

To add additional services, see the OpenStack Installation Tutorials and Guides

This chapter assumes a working setup of OpenStack following the OpenStack Installation Tutorials and Guides.

CHAPTER

TWO

CONTRIBUTOR GUIDE

In the Contributions Guide, you will find documented policies for developing with Aodh. This includes the processes we use for bugs, contributor onboarding, core reviewer memberships, and other procedural items.

2.1 Overview

2.1.1 System Architecture

High-Level Architecture

Each of Aodh's services are designed to scale horizontally. Additional workers and nodes can be added depending on the expected load. It provides daemons to evaluate and notify based on defined alarming rules.

Evaluating the data

Alarming Service

The alarming component of Aodh, first delivered in Ceilometer service during Havana development cycle then split out to this independent project in Liberty development cycle, allows you to set alarms based on threshold evaluation for a collection of samples or a dedicate event. An alarm can be set on a single meter, or on a combination. For example, you may want to trigger an alarm when the memory consumption reaches 70% on a given instance if the instance has been up for more than 10 min. To setup an alarm, you will call *Aodh's API server* specifying the alarm conditions and an action to take.

Of course, if you are not administrator of the cloud itself, you can only set alarms on meters for your own components.

There can be multiple form of actions, but only several actions have been implemented so far:

1. *HTTP callback*: you provide a URL to be called whenever the alarm has been set off. The payload of the request contains all the details of why the alarm was triggered.
2. *log*: mostly useful for debugging, stores alarms in a log file.
3. *zaqar*: Send notification to messaging service via Zaqar API.

Alarm Rules

composite

Composite alarm rule.

A simple dict type to preset composite rule.

event

Alarm Event Rule.

Describe when to trigger the alarm based on an event

gnocchi_aggregation_by_metrics_threshold

Base class Alarm Rule extension and wsme.types.

gnocchi_aggregation_by_resources_threshold

Base class Alarm Rule extension and wsme.types.

gnocchi_resources_threshold

Base class Alarm Rule extension and wsme.types.

loadbalancer_member_health

Base class Alarm Rule extension and wsme.types.

Alarm Evaluators

composite

Base class for alarm rule evaluator plugins.

gnocchi_aggregation_by_metrics_threshold

Base class for alarm rule evaluator plugins.

gnocchi_aggregation_by_resources_threshold

Base class for alarm rule evaluator plugins.

gnocchi_resources_threshold

Base class for alarm rule evaluator plugins.

loadbalancer_member_health

Base class for alarm rule evaluator plugins.

Alarm Notifiers

http

Rest alarm notifier.

https

Rest alarm notifier.

log

Log alarm notifier.

test

Test alarm notifier.

trust+heat

Heat autohealing notifier.

The auto-healing notifier works together with loadbalancer_member_health evaluator.

Presumably, the end user defines a Heat template which contains an autoscaling group and all the members in the group are joined in an Octavia load balancer in order to expose service to the outside, so that when the stack scales up or scales down, Heat makes sure the new members are joining the load balancer automatically and the old members are removed.

However, this notifier deals with the situation that when some member fails, the stack could be recovered by marking the given autoscaling group member unhealthy, then update Heat stack in place. In order to do that, the notifier needs to know:

- Heat top/root stack ID.

- Heat autoscaling group ID.
- The failed Octavia pool members.

trust+http

Notifier supporting keystone trust authentication.

This alarm notifier is intended to be used to call an endpoint using keystone authentication. It uses the aodh service user to authenticate using the trust ID provided.

The URL must be in the form `trust+http://host/action`.

trust+https

Notifier supporting keystone trust authentication.

This alarm notifier is intended to be used to call an endpoint using keystone authentication. It uses the aodh service user to authenticate using the trust ID provided.

The URL must be in the form `trust+http://host/action`.

trust+zaqar

Zaqar notifier using a Keystone trust to post to user-defined queues.

The URL must be in the form `trust+zaqar://?queue_name=example`.

zaqar

Zaqar notifier.

This notifier posts alarm notifications either to a Zaqar subscription or to an existing Zaqar queue with a pre-signed URL.

To create a new subscription in the service project, use a notification URL of the form:

```
zaqar://?topic=example&subscriber=mailto%3A//test%40example.com&ttl=3600
```

Multiple subscribers are allowed. `ttl` is the time to live of the subscription. The queue will be created automatically, in the service project, with a name based on the topic and the alarm ID.

To use a pre-signed URL for an existing queue, use a notification URL with the scheme `zaqar://` and the pre-signing data from Zaqar in the query string:

```
zaqar://?queue_name=example&project_id=foo&
        paths=/messages&methods=POST&expires=1970-01-01T00:00Z&
        signature=abcdefg
```

Alarm Storage

log

Log the data.

mysql

Put the data into a SQLAlchemy database.

mysql+pymysql

Put the data into a SQLAlchemy database.

postgresql

Put the data into a SQLAlchemy database.

sqlite

Put the data into a SQLAlchemy database.

2.1.2 Web API

V2 Web API

Capabilities

The Capabilities API allows you to directly discover which functions from the V2 API functionality, including the selectable aggregate functions, are supported by the currently configured storage driver. A capabilities query returns a flattened dictionary of properties with associated boolean values - a 'False' or absent value means that the corresponding feature is not available in the backend.

`GET /v2/capabilities`

Returns a flattened dictionary of API capabilities.

Capabilities supported by the currently configured storage driver.

Return type *`Capabilities`*

`type Capabilities`

A representation of the API and storage capabilities.

Usually constrained by restrictions imposed by the storage driver.

Data samples:

Json

```
{
    "alarm_storage": {
        "storage:production_ready": true
    },
    "api": {
        "alarms:history:query:complex": true,
        "alarms:history:query:simple": true,
        "alarms:query:complex": true,
        "alarms:query:simple": true
    }
}
```

XML

```
b'<value>\n  <api>\n    <item>\n      <key>
↪alarms:history:query:complex</key>\n      <value>true</value>\n
↪</item>\n    <item>\n      <key>alarms:history:query:simple</key>\
↪n      <value>true</value>\n    </item>\n    <item>\n      <key>
↪alarms:query:complex</key>\n      <value>true</value>\n    </item>\
↪n    <item>\n      <key>alarms:query:simple</key>\n      <value>
↪true</value>\n    </item>\n  </api>\n  <alarm_storage>\n    <item>\
↪n      <key>storage:production_ready</key>\n      <value>true</
↪value>\n    </item>\n  </alarm_storage>\n</value>'
```

`alarm_storage`

Type dict(str: bool)

A flattened dictionary of alarm storage capabilities

`api`

Type dict(str: bool)

A flattened dictionary of API capabilities

Alarms

`GET /v2/alarms`

Return all alarms, based on the query provided.

Parameters

- **q** (list(`Query`)) -- Filter rules for the alarms to be returned.
- **sort** (list(`str`)) -- A list of pairs of sort key and sort dir.
- **limit** (`int`) -- The maximum number of items to be return.
- **marker** (`str`) -- The pagination query marker.

Return type list(*Alarm*)

`POST /v2/alarms`
Create a new alarm.

Parameters

- `data` (*`Alarm`*) -- an alarm within the request body.

Return type *`Alarm`*

`GET /v2/alarms/`(*alarm_id*)
Return this alarm.

Return type *`Alarm`*

`PUT /v2/alarms/`(*alarm_id*)
Modify this alarm.

Parameters

- `data` (*`Alarm`*) -- an alarm within the request body.

Return type *`Alarm`*

`DELETE /v2/alarms/`(*alarm_id*)
Delete this alarm.

`GET /v2/alarms/`(*alarm_id*)`/history`
Assembles the alarm history requested.

Parameters

- `q` (list(`Query`)) -- Filter rules for the changes to be described.
- `sort` (list(`str`)) -- A list of pairs of sort key and sort dir.
- `limit` (`int`) -- The maximum number of items to be return.
- `marker` (`str`) -- The pagination query marker.

Return type list(*`AlarmChange`*)

`PUT /v2/alarms/`(*alarm_id*)`/state`
Set the state of this alarm.

Parameters

- `state` (`Enum(ok, alarm, insufficient data)`) -- an alarm state within the request body.

Return type `Enum(ok, alarm, insufficient data)`

`GET /v2/alarms/`(*alarm_id*)`/state`
Get the state of this alarm.

Return type `Enum(ok, alarm, insufficient data)`

`type Alarm`
Representation of an alarm.

Data samples:

Json

```
{
    "alarm_actions": [
        "http://site:8000/alarm"
    ],
    "alarm_id": null,
    "description": "An alarm",
    "enabled": true,
    "insufficient_data_actions": [
        "http://site:8000/nodata"
    ],
    "name": "SwiftObjectAlarm",
    "ok_actions": [
        "http://site:8000/ok"
    ],
    "project_id": "c96c887c216949acbdfbd8b494863567",
    "repeat_actions": false,
    "severity": "moderate",
    "state": "ok",
    "state_reason": "threshold over 90%",
    "state_timestamp": "2015-01-01T12:00:00",
    "time_constraints": [
        {
            "description": "nightly build every night at 23h for 3␣
↪hours",
            "duration": 10800,
            "name": "SampleConstraint",
            "start": "0 23 * * *",
            "timezone": "Europe/Ljubljana"
        }
    ],
    "timestamp": "2015-01-01T12:00:00",
    "type": "gnocchi_aggregation_by_metrics_threshold",
    "user_id": "c96c887c216949acbdfbd8b494863567"
}
```

XML

```
b'<value>\n  <alarm_id nil="true" />\n  <name>SwiftObjectAlarm</name>
↪\n  <description>An alarm</description>\n  <enabled>true</enabled>\
↪n  <ok_actions>\n    <item>http://site:8000/ok</item>\n  </ok_
↪actions>\n  <alarm_actions>\n    <item>http://site:8000/alarm</
↪item>\n  </alarm_actions>\n  <insufficient_data_actions>\n    
↪<item>http://site:8000/nodata</item>\n  </insufficient_data_
↪actions>\n  <repeat_actions>false</repeat_actions>\n  <type>
↪gnocchi_aggregation_by_metrics_threshold</type>\n  <time_
↪constraints>\n    <item>\n      <name>SampleConstraint</name>\n    ␣
↪  <description>nightly build every night at 23h for 3 hours</
↪description>\n      <start>0 23 * * *</start>\n      <duration>
↪10800</duration>\n      <timezone>Europe/Ljubljana</timezone>\n    
↪</item>\n  </time_constraints>\n  <project_id>
↪c96c887c216949acbdfbd8b494863567</project_id>\n  <user_id>
↪c96c887c216949acbdfbd8b494863567</user_id>\n  <timestamp>2015-01-
↪01T12:00:00</timestamp>\n  <state>ok</state>\n  <state_timestamp>
↪2015-01-01T12:00:00</state_timestamp>\n  <state_reason>threshold␣
↪over 90%</state_reason>\n  <severity>moderate</severity>\n</value>'
```

(continues on next page)

(continued from previous page)

alarm_actions

Type list(str)

The actions to do when alarm state change to alarm

alarm_id

Type str

The UUID of the alarm

property description

The description of the alarm

enabled

Type bool

This alarm is enabled?

evaluate_timestamp

Type datetime

The latest alarm evaluation time

insufficient_data_actions

Type list(str)

The actions to do when alarm state change to insufficient data

name

Type str

The name for the alarm

ok_actions

Type list(str)

The actions to do when alarm state change to ok

project_id

Type str

The ID of the project or tenant that owns the alarm

repeat_actions

Type bool

The actions should be re-triggered on each evaluation cycle

`property severity`
The severity of the alarm

`property state`
The state offset the alarm

`state_reason`

Type str

The reason of the current state

`state_timestamp`

Type datetime

The date of the last alarm state changed

`time_constraints`

Type list(AlarmTimeConstraint)

Describe time constraints for the alarm

`timestamp`

Type datetime

The date of the last alarm definition update

`property type`
Explicit type specifier to select which rule to follow below.

`user_id`

Type str

The ID of the user who created the alarm

`type MetricOfResourceRule`
Data samples:

Json

```
{
    "comparison_operator": "eq",
    "evaluation_periods": 1,
    "granularity": 60
}
```

XML

```
b'<value>\n  <comparison_operator>eq</comparison_operator>\n
↪<evaluation_periods>1</evaluation_periods>\n  <granularity>60</
↪granularity>\n</value>'
```

metric

Type str

The name of the metric

resource_id

Type str

The id of a resource

resource_type

Type str

The resource type

type AggregationMetricByResourcesLookupRule

Data samples:

Json

```
{
    "comparison_operator": "eq",
    "evaluation_periods": 1,
    "granularity": 60
}
```

XML

```
b'<value>\n  <comparison_operator>eq</comparison_operator>\n
↪<evaluation_periods>1</evaluation_periods>\n  <granularity>60</
↪granularity>\n</value>'
```

metric

Type str

The name of the metric

query

Type str

The query to filter the metric, Don't forget to filter out deleted resources (example: {"and": [{"=": {"ended_at": null}}, ...]}), Otherwise Gnocchi will try to create the aggregate against obsolete resources

resource_type

Type str

The resource type

type AggregationMetricsByIdLookupRule

Data samples:

Json

```
{
    "comparison_operator": "eq",
    "evaluation_periods": 1,
    "granularity": 60
}
```

XML

```
b'<value>\n  <comparison_operator>eq</comparison_operator>\n
↪<evaluation_periods>1</evaluation_periods>\n  <granularity>60</
↪granularity>\n</value>'
```

metrics

Type list(str)

A list of metric Ids

type AlarmTimeConstraint

Representation of a time constraint on an alarm.

Data samples:

Json

```
{
    "description": "nightly build every night at 23h for 3 hours",
    "duration": 10800,
    "name": "SampleConstraint",
    "start": "0 23 * * *",
    "timezone": "Europe/Ljubljana"
}
```

XML

```
b'<value>\n  <name>SampleConstraint</name>\n  <description>nightly␣
↪build every night at 23h for 3 hours</description>\n  <start>0 23␣
↪* * *</start>\n  <duration>10800</duration>\n  <timezone>Europe/
↪Ljubljana</timezone>\n</value>'
```

property description

The description of the constraint

duration

Type integer

How long the constraint should last, in seconds

name

Type str

The name of the constraint

start

Type cron

Start point of the time constraint, in cron format

timezone

Type str

Timezone of the constraint

type AlarmChange

Representation of an event in an alarm's history.

Data samples:

Json

```
{
    "alarm_id": "e8ff32f772a44a478182c3fe1f7cad6a",
    "detail": "{\"threshold\": 42.0, \"evaluation_periods\": 4}",
    "on_behalf_of": "92159030020611e3b26dde429e99ee8c",
    "project_id": "b6f16144010811e387e4de429e99ee8c",
    "timestamp": "2015-01-01T12:00:00",
    "type": "rule change",
    "user_id": "3e5d11fda79448ac99ccefb20be187ca"
}
```

XML

```
b'<value>\n  <alarm_id>e8ff32f772a44a478182c3fe1f7cad6a</alarm_id>\n␣
↪ <type>rule change</type>\n  <detail>{"threshold": 42.0,
↪"evaluation_periods": 4}</detail>\n  <project_id>
↪b6f16144010811e387e4de429e99ee8c</project_id>\n  <user_id>
↪3e5d11fda79448ac99ccefb20be187ca</user_id>\n  <on_behalf_of>
↪92159030020611e3b26dde429e99ee8c</on_behalf_of>\n  <timestamp>2015-
↪01-01T12:00:00</timestamp>\n</value>'
```

alarm_id

Type str

The UUID of the alarm

detail

Type str

JSON fragment describing change

`event_id`

Type str

The UUID of the change event

`on_behalf_of`

Type str

The tenant on behalf of which the change is being made

`project_id`

Type str

The project ID of the initiating identity

`timestamp`

Type datetime

The time/date of the alarm change

`type`

Type Enum(creation, rule change, state transition, deletion)

The type of change

`user_id`

Type str

The user ID of the initiating identity

Filtering Queries

The filter expressions of the query feature operate on the fields of *Alarm* and *AlarmChange*. The following comparison operators are supported: =, *!*=, <, <=, >, >= and *in*; and the following logical operators can be used: *and or* and *not*. The field names are validated against the database models.

Complex Query supports defining the list of orderby expressions in the form of [{"field_name": "asc"}, {"field_name2": "desc"}, ...].

The number of the returned items can be bounded using the *limit* option.

The *filter*, *orderby* and *limit* are all optional fields in a query.

`POST /v2/query/alarms`
Define query for retrieving Alarm data.

Parameters

- **body** (*ComplexQuery*) -- Query rules for the alarms to be returned.

Return type list(*Alarm*)

POST /v2/query/alarms/history

Define query for retrieving AlarmChange data.

Parameters

- **body** (*ComplexQuery*) -- Query rules for the alarm history to be returned.

Return type list(*AlarmChange*)

type ComplexQuery

Holds a sample query encoded in json.

Data samples:

Json

```
{
    "filter": "{\"and\": [{\"and\": [{\"=\": {\"counter_name\": \
↪\"cpu_util\"}}, {\">\": {\"counter_volume\": 0.23}}, {\"<\": {\
↪\"counter_volume\": 0.26}}]}, {\"or\": [{\"and\": [{\">\": {\
↪\"timestamp\": \"2013-12-01T18:00:00\"}}, {\"<\": {\"timestamp\": \
↪\"2013-12-01T18:15:00\"}}]}, {\"and\": [{\">\": {\"timestamp\": \
↪\"2013-12-01T18:30:00\"}}, {\"<\": {\"timestamp\": \"2013-12-
↪01T18:45:00\"}}]}]}]}",
    "limit": 42,
    "orderby": "[{\"counter_volume\": \"ASC\"}, {\"timestamp\": \
↪\"DESC\"}]"
}
```

XML

```
b'<value>\n  <filter>{"and": [{"and": [{"=": {"counter_name": "cpu_
↪util"}}, {"&gt;": {"counter_volume": 0.23}}, {"&lt;": {"counter_
↪volume": 0.26}}]}, {"or": [{"and": [{"&gt;": {"timestamp": "2013-
↪12-01T18:00:00"}}, {"&lt;": {"timestamp": "2013-12-01T18:15:00"}}]}
↪, {"and": [{"&gt;": {"timestamp": "2013-12-01T18:30:00"}}, {"&lt;
↪": {"timestamp": "2013-12-01T18:45:00"}}]}]}]}</filter>\n
↪<orderby>[{"counter_volume": "ASC"}, {"timestamp": "DESC"}]</
↪orderby>\n  <limit>42</limit>\n</value>'
```

filter

Type str

The filter expression encoded in json.

limit

Type int

The maximum number of results to be returned.

orderby

Type str

List of single-element dicts for specifing the ordering of the results.

Composite rule Alarm

The *composite* type alarm allows users to specify a composite rule to define an alarm with multiple triggering conditions, using a combination of *and* and *or* relations. A composite rule is composed of multiple threshold rules or gnocchi rules. A sample composite alarm request form is as follows:

```
{
    "name": "test_composite",
    "type": "composite",
    "composite_rule": {
        "and": [THRESHOLD_RULE1, THRESHOLD_RULE2, {
            'or': [THRESHOLD_RULE3, GNOCCHI_RULE1,
                   GNOCCHI_RULE2, GNOCCHI_RULE3]
        }]
    }
}
```

A sub-rule in composite_rule is same as a threshold_rule in threshold alarm or a gnocchi_rule in gnocchi alarm. Additionally it has a mandatory *type* field to specify the rule type, like in the following sample:

```
{
    "threshold": 0.8,
    "meters": [
        "f6857d3f-bde6-441a-aa1d-e98fa4ea543f",
        "ea1491ca-5309-4b5a-9f05-34409c6e8b6c"
    ],
    "type": "gnocchi_resources_threshold"
}
```

You can get API version list via request to endpoint root path. For example:

```
curl -H "X-AUTH-TOKEN: fa2ec18631f94039a5b9a8b4fe8f56ad" http://127.0.0.1:8042
```

Sample response:

```
{
    "versions": {
        "values": [
            {
                "id": "v2",
                "links": [
                    {
                        "href": "http://127.0.0.1:8042/v2",
                        "rel": "self"
```

(continues on next page)

(continued from previous page)

```
                    },
                    {
                        "href": "https://docs.openstack.org/",
                        "rel": "describedby",
                        "type": "text/html"
                    }
                ],
                "media-types": [
                    {
                        "base": "application/json",
                        "type": "application/vnd.openstack.telemetry-v2+json"
                    },
                    {
                        "base": "application/xml",
                        "type": "application/vnd.openstack.telemetry-v2+xml"
                    }
                ],
                "status": "stable",
                "updated": "2013-02-13T00:00:00Z"
            }
        ]
    }
}
```

2.2 Developer Documentation

2.2.1 Installing Aodh

Installing development sandbox

Configuring devstack

1. Download devstack.
2. Create a `local.conf` file as input to devstack.

   Note: `local.conf` replaces the former configuration file called `localrc`. If you used localrc before, remove it to switch to using the new file. For further information see the devstack configuration.

3. The aodh services are not enabled by default, so they must be enabled in `local.conf` before running `stack.sh`.

   This example `local.conf` file shows all of the settings required for aodh:

   ```
   [[local|localrc]]
   ```

(continues on next page)

(continued from previous page)

```
# Enable the aodh alarming services
enable_plugin aodh https://opendev.org/openstack/aodh master
```

Installing Manually

Installing the API Server

There are two recommended ways to start api server:

1. Starting API server through mod_wsgi;
2. Starting API server through: uwsgi.

Not recommended, for testing purpose, we can also start api server by aodh-api binary:

```
aodh-api --port 8042 -- --config-file /etc/aodh/aodh.conf
```

Database configuration

You can use any SQLAlchemy-supported DB such as *PostgreSQL* or *MySQL*. To use MySQL as the storage backend, change the 'database' section in aodh.conf as follows:

```
[database]
connection = mysql+pymysql://username:password@host/aodh?charset=utf8
```

Installing the API behind mod_wsgi

Aodh comes with a WSGI application file named *aodh/api/app.wsgi* for configuring the API service to run behind Apache with `mod_wsgi`. This file is installed with the rest of the Aodh application code, and should not need to be modified.

You can then configure Apache with something like this:

```
Listen 8042

<VirtualHost *:8042>
    WSGIDaemonProcess aodh-api processes=2 threads=10 user=SOMEUSER display-
↪name=%{GROUP}
    WSGIProcessGroup aodh-api
    WSGIScriptAlias / /usr/lib/python2.7/dist-packages/aodh/api/app
    WSGIApplicationGroup %{GLOBAL}
    <IfVersion >= 2.4>
        ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    ErrorLog /var/log/httpd/aodh_error.log
    CustomLog /var/log/httpd/aodh_access.log combined
</VirtualHost>
```

(continues on next page)

(continued from previous page)

```
WSGISocketPrefix /var/run/httpd
```

Modify the WSGIDaemonProcess directive to set the user and group values to an appropriate user on your server. In many installations aodh will be correct.

Installing the API with uwsgi

Aodh comes with a few example files for configuring the API service to run behind Apache with mod_wsgi.

app.wsgi

The file aodh/api/app.wsgi sets up the V2 API WSGI application. The file is installed with the rest of the Aodh application code, and should not need to be modified.

Example of uwsgi configuration file

Create aodh-uwsgi.ini file:

```
[uwsgi]
http = 0.0.0.0:8041
wsgi-file = <path_to_aodh>/aodh/api/app.wsgi
plugins = python
# This is running standalone
master = true
# Set die-on-term & exit-on-reload so that uwsgi shuts down
exit-on-reload = true
die-on-term = true
# uwsgi recommends this to prevent thundering herd on accept.
thunder-lock = true
# Override the default size for headers from the 4k default. (mainly for␣
↪keystone token)
buffer-size = 65535
enable-threads = true
# Set the number of threads usually with the returns of command nproc
threads = 8
# Make sure the client doesn't try to re-use the connection.
add-header = Connection: close
# Set uid and gip to an appropriate user on your server. In many
# installations ``aodh`` will be correct.
uid = aodh
gid = aodh
```

Then start the uwsgi server:

```
uwsgi ./aodh-uwsgi.ini
```

Or start in background with:

```
uwsgi -d ./aodh-uwsgi.ini
```

Configuring with uwsgi-plugin-python on Debian/Ubuntu

Install the Python plugin for uwsgi:

apt-get install uwsgi-plugin-python

Run the server:

uwsgi_python --master --die-on-term --logto /var/log/aodh/aodh-api.log --http-socket :8042 --wsgi-file /usr/share/aodh-common/app.wsgi

2.2.2 Running the Tests

Aodh includes an extensive set of automated unit tests which are run through tox.

1. Install `tox`:

   ```
   $ sudo pip install tox
   ```

2. On Ubuntu install `libmysqlclient-dev` packages:

   ```
   $ sudo apt-get install libmysqlclient-dev
   ```

   For Fedora20 there is no `libmysqlclient-dev` package, so youll need to install `mariadb-devel.x86-64` (or `mariadb-devel.i386`) instead:

   ```
   $ sudo yum install mariadb-devel.x86_64
   ```

3. Run the unit and code-style tests:

   ```
   $ cd /opt/stack/aodh
   $ tox -e py27,pep8
   ```

   As tox is a wrapper around testr, it also accepts the same flags as testr. See the testr documentation for details about these additional flags.

   Use a double hyphen to pass options to testr. For example, to run only tests under tests/functional/api/v2:

   ```
   $ tox -e py27 -- functional.api.v2
   ```

   To debug tests (ie. break into pdb debugger), you can use "debug" tox environment. Here's an example, passing the name of a test since you'll normally only want to run the test that hits your breakpoint:

   ```
   $ tox -e debug aodh.tests.unit.test_bin
   ```

   For reference, the **debug** tox environment implements the instructions here: https://wiki.openstack.org/wiki/Testr#Debugging_.28pdb.29_Tests

4. There is a growing suite of tests which use a tool called gabbi to test and validate the behavior of the Aodh API. These tests are run when using the usual `functional` tox target but if desired they can be run by themselves:

   ```
   $ tox -e gabbi
   ```

   The YAML files used to drive the gabbi tests can be found in `aodh/tests/functional/gabbi/gabbits`. If you are adding to or adjusting the API you should consider adding tests here.

See also:

- tox

2.2.3 Contributing to Aodh

Aodh follows the same workflow as other OpenStack projects. To start contributing to Aodh, please follow the workflow found here.

Project Hosting Details

Bug tracker https://bugs.launchpad.net/aodh

Mailing list http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-dev (prefix subjects with `[Aodh]` for faster responses)

Code Hosting https://opendev.org/openstack/aodh/

Code Review https://review.opendev.org/#/q/status:open+project:openstack/aodh,n,z

2.2.4 Event alarm

Aodh allows users to define alarms which can be evaluated based on events passed from other OpenStack services. The events can be emitted when the resources from other OpenStack services have been updated, created or deleted, such as 'compute.instance.reboot.end', 'scheduler.select_destinations.end'. When creating an alarm with type of "event", an event_type can be specified to identify the type of event that will trigger the alarm. The event_type field support fuzzy matching with wildcard. Additionally, users can also specify query conditions to filter specific events used to trigger the alarm.

This feature was implemented with proposal event-alarm.

Usage

When creating an alarm of "event" type, the "event_rule" need to be specified, which includes an "event_type" field and a "query" field, the "event_type" allow users to specify a specific event type used to match the incoming events when evaluating alarm, and the "query" field includes a list of query conditions used to filter specific events when evaluating the alarm.

The following is an example of event alarm rule:

```
"event_rule": {
    "event_type": "compute.instance.update",
    "query" : [
```

(continues on next page)

(continued from previous page)

```
        {
            "field" : "traits.instance_id",
            "type" : "string",
            "value" : "153462d0-a9b8-4b5b-8175-9e4b05e9b856",
            "op" : "eq",
        },
        {
            "field" : "traits.state",
            "type" : "string",
            "value" : "error",
            "op" : "eq",
        },
    ]
}
```

Configuration

To enable this functionality, config the Ceilometer to be able to publish events to the queue the aodh-listener service listen on. The *event_alarm_topic* config option of Aodh identify which messaging topic the aodh-listener on, the default value is "alarm.all". In Ceilometer side, a publisher of notifier type need to be configured in the event pipeline config file(`event_pipeline.yaml` as default), the notifier should be with a messaging topic same as the *event_alarm_topic* option defined. For an example:

```
---
sources:
    - name: event_source
      events:
          - "*"
      sinks:
          - event_sink
sinks:
    - name: event_sink
      transformers:
      publishers:
          - notifier://
          - notifier://?topic=alarm.all
```

2.2.5 Guru Meditation Reports

Aodh contains a mechanism whereby developers and system administrators can generate a report about the state of a running Aodh executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR1* signal to any Aodh process with support (see below). The *GMR* will then be outputted standard error for that particular process.

For example, suppose that `aodh-listener` has process id `8675`, and was run with `2>/var/log/aodh/aodh-listener.log`. Then, `kill -USR1 8675` will trigger the Guru Meditation report to be printed to `/var/log/aodh/aodh-listener.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package Shows information about the package to which this process belongs, including version information

Threads Shows stack traces and thread ids for each of the threads within this process

Green Threads Shows stack traces for each of the green threads within this process (green threads don't have thread ids)

Configuration Lists all the configuration options currently accessible via the CONF object for the current process

Adding Support for GMRs to New Executables

Adding support for a *GMR* to a given executable is fairly easy.

First import the module (currently residing in oslo-incubator), as well as the Aodh version module:

```
from oslo_reports import guru_meditation_report as gmr
from aodh import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the "main loop" of the executable (usually `service.server(server)` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the inline documentation about oslo.reports: oslo.reports

2.3 Appendix

2.3.1 Release Notes

- Liberty

Since Mitaka development cycle, we start to host release notes on: Aodh Release Notes

2.4 Indices and tables

- genindex
- modindex
- search

CHAPTER

THREE

ADMINISTRATION GUIDE

This guide contains information that will help you understand how to deploy, operate, and upgrade Aodh

3.1 Alarms

Alarms provide user-oriented Monitoring-as-a-Service for resources running on OpenStack. This type of monitoring ensures you can automatically scale in or out a group of instances through the Orchestration service, but you can also use alarms for general-purpose awareness of your cloud resources' health.

These alarms follow a tri-state model:

ok The rule governing the alarm has been evaluated as `False`.

alarm The rule governing the alarm has been evaluated as `True`.

insufficient data There are not enough datapoints available in the evaluation periods to meaningfully determine the alarm state.

3.1.1 Alarm definitions

The definition of an alarm provides the rules that govern when a state transition should occur, and the actions to be taken thereon. The nature of these rules depend on the alarm type.

Threshold rule alarms

For conventional threshold-oriented alarms, state transitions are governed by:

- A static threshold value with a comparison operator such as greater than or less than.
- A statistic selection to aggregate the data.
- A sliding time window to indicate how far back into the recent past you want to look.

Both Ceilometer and Gnocchi are supported as data source of the threshold rule alarm. Valid threshold alarms are:

- threshold
- gnocchi_resources_threshold
- gnocchi_aggregation_by_metrics_threshold
- gnocchi_aggregation_by_resources_threshold

Composite rule alarms

Composite alarms enable users to define an alarm with multiple triggering conditions, using a combination of `and` and `or` relations.

3.1.2 Alarm dimensioning

A key associated concept is the notion of *dimensioning* which defines the set of matching meters that feed into an alarm evaluation. Recall that meters are per-resource-instance, so in the simplest case an alarm might be defined over a particular meter applied to all resources visible to a particular user. More useful however would be the option to explicitly select which specific resources you are interested in alarming on.

At one extreme you might have narrowly dimensioned alarms where this selection would have only a single target (identified by resource ID). At the other extreme, you could have widely dimensioned alarms where this selection identifies many resources over which the statistic is aggregated. For example all instances booted from a particular image or all instances with matching user metadata (the latter is how the Orchestration service identifies autoscaling groups).

3.1.3 Alarm evaluation

Alarms are evaluated by the `alarm-evaluator` service on a periodic basis, defaulting to once every minute.

Alarm actions

Any state transition of individual alarm (to `ok`, `alarm`, or `insufficient data`) may have one or more actions associated with it. These actions effectively send a signal to a consumer that the state transition has occurred, and provide some additional context. This includes the new and previous states, with some reason data describing the disposition with respect to the threshold, the number of datapoints involved and most recent of these. State transitions are detected by the `alarm-evaluator`, whereas the `alarm-notifier` effects the actual notification action.

HTTP/HTTPS action These are the *de facto* notification type used by Telemetry alarming and simply involve an HTTP(S) POST request being sent to an endpoint, with a request body containing a description of the state transition encoded as a JSON fragment.

OpenStack Services The user is able to define an alarm that simply trigger some OpenStack service by directly specifying the service URL, e.g. `trust+http://127.0.0.1:7070/v1/webhooks/ab91ef39-3e4a-4750-a8b8-0271518cd481/invoke`. `aodh-notifier` will prepare `X-Auth-Token` header and send HTTP(S) POST request to that URL, containing the alarm information in the request body.

Heat Autoscaling This notifier works together with `loadbalancer_member_health` evaluator. Presumably, the end user defines a Heat template which contains an autoscaling group and all the members in the group are joined in an Octavia load balancer in order to expose highly available service to the outside, so that when the stack scales up or scales down, Heat makes sure the new members are joining the load balancer automatically and the old members are removed. However, this notifier deals with the situation that when some member fails, the Heat stack could be recovered automatically. More information here

Log actions These are a lightweight alternative to webhooks, whereby the state transition is simply logged by the `alarm-notifier`, and are intended primarily for testing purposes by admin users.

If none of the above actions satisfy your requirement, you can implement your own alarm actions according to the current suppported actions in `aodh/notifier` folder.

3.1.4 Using alarms

Alarm creation

Threshold based alarm

An example of creating a Gnocchi threshold-oriented alarm, based on an upper bound on the CPU utilization for a particular instance:

```
$ aodh alarm create \
  --name cpu_hi \
  --type gnocchi_resources_threshold \
  --description 'instance running hot' \
  --metric cpu_util \
  --threshold 70.0 \
  --comparison-operator gt \
  --aggregation-method mean \
  --granularity 600 \
  --evaluation-periods 3 \
  --alarm-action 'log://' \
  --resource-id INSTANCE_ID \
  --resource-type instance
```

This creates an alarm that will fire when the average CPU utilization for an individual instance exceeds 70% for three consecutive 10 minute periods. The notification in this case is simply a log message, though it could alternatively be a webhook URL.

Note: Alarm names must be unique for the alarms associated with an individual project. Administrator can limit the maximum resulting actions for three different states, and the ability for a normal user to create `log://` and `test://` notifiers is disabled. This prevents unintentional consumption of disk and memory resources by the Telemetry service.

The sliding time window over which the alarm is evaluated is 30 minutes in this example. This window is not clamped to wall-clock time boundaries, rather it's anchored on the current time for each evaluation cycle, and continually creeps forward as each evaluation cycle rolls around (by default, this occurs every minute).

Note: The alarm granularity must match the granularities of the metric configured in Gnocchi.

Otherwise the alarm will tend to flit in and out of the `insufficient data` state due to the mismatch between the actual frequency of datapoints in the metering store and the statistics queries used to compare against the alarm threshold. If a shorter alarm period is needed, then the corresponding interval should be adjusted in the `pipeline.yaml` file.

Other notable alarm attributes that may be set on creation, or via a subsequent update, include:

state The initial alarm state (defaults to `insufficient data`).

description A free-text description of the alarm (defaults to a synopsis of the alarm rule).

enabled True if evaluation and actioning is to be enabled for this alarm (defaults to `True`).

repeat-actions True if actions should be repeatedly notified while the alarm remains in the target state (defaults to `False`).

ok-action An action to invoke when the alarm state transitions to `ok`.

insufficient-data-action An action to invoke when the alarm state transitions to `insufficient data`.

time-constraint Used to restrict evaluation of the alarm to certain times of the day or days of the week (expressed as `cron` expression with an optional timezone).

Composite alarm

An example of creating a composite alarm, based on the composite of two basic rules:

```
$ aodh alarm create \
  --name meta \
  --type composite \
  --composite-rule '{"or": [{"threshold": 0.8, "metric": "cpu_util", \
    "type": "gnocchi_resources_threshold", "resource_id": INSTANCE_ID1, \
    "resource_type": "instance", "aggregation_method": "last"}, \
    {"threshold": 0.8, "metric": "cpu_util", \
    "type": "gnocchi_resources_threshold", "resource_id": INSTANCE_ID2, \
    "resource_type": "instance", "aggregation_method": "last"}]}' \
  --alarm-action 'http://example.org/notify'
```

This creates an alarm that will fire when either of two basic rules meets the condition. The notification in this case is a webhook call. Any number of basic rules can be composed into a composite rule this way, using either `and` or `or`. Additionally, composite rules can contain nested conditions:

Note: Observe the *underscore in* `resource_id` & `resource_type` in composite rule as opposed to `--resource-id` & `--resource-type` CLI arguments.

```
$ aodh alarm create \
  --name meta \
  --type composite \
  --composite-rule '{"or": [ALARM_1, {"and": [ALARM_2, ALARM_3]}]}' \
  --alarm-action 'http://example.org/notify'
```

Event based alarm

An example of creating a event alarm based on power state of instance:

```
$ aodh alarm create \
  --type event \
  --name instance_off \
  --description 'Instance powered OFF' \
  --event-type "compute.instance.power_off.*" \
  --enable True \
  --query "traits.instance_id=string::INSTANCE_ID" \
  --alarm-action 'log://' \
  --ok-action 'log://' \
  --insufficient-data-action 'log://'
```

Valid list of `event-type` and `traits` can be found in `event_definitions.yaml` file . `--query` may also contain mix of traits for example to create alarm when instance is powered on but went into error state:

```
$ aodh alarm create \
  --type event \
  --name instance_on_but_in_err_state \
  --description 'Instance powered ON but in error state' \
  --event-type "compute.instance.power_on.*" \
  --enable True \
  --query "traits.instance_id=string::INSTANCE_ID;traits.state=string::error"␣
↪\
  --alarm-action 'log://' \
  --ok-action 'log://' \
  --insufficient-data-action 'log://'
```

Sample output of alarm type **event**:

```
+---------------------------+---------------------------------------------------
↪--------------+
| Field                     | Value                                             ␣
↪              |
+---------------------------+---------------------------------------------------
↪--------------+
| alarm_actions             | [u'log://']                                       ␣
↪              |
| alarm_id                  | 15c0da26-524d-40ad-8fba-3e55ee0ddc91              ␣
↪              |
| description               | Instance powered ON but in error state            ␣
↪              |
| enabled                   | True                                              ␣
↪              |
| event_type                | compute.instance.power_on.*                       ␣
↪              |
| insufficient_data_actions | [u'log://']                                       ␣
↪              |
```

(continues on next page)

(continued from previous page)

```
| name                        | instance_on_state_err                                  ␣
↪                 |
| ok_actions                  | [u'log://']                                            ␣
↪                 |
| project_id                  | 9ee200732f4c4d10a6530bac746f1b6e                       ␣
↪                 |
| query                       | traits.instance_id = bb912729-fa51-443b-bac6-
↪bf4c795f081d AND |
|                             | traits.state = error                                   ␣
↪                 |
| repeat_actions              | False                                                  ␣
↪                 |
| severity                    | low                                                    ␣
↪                 |
| state                       | insufficient data                                      ␣
↪                 |
| state_timestamp             | 2017-07-15T02:28:31.114455                             ␣
↪                 |
| time_constraints            | []                                                     ␣
↪                 |
| timestamp                   | 2017-07-15T02:28:31.114455                             ␣
↪                 |
| type                        | event                                                  ␣
↪                 |
| user_id                     | 89b4e48bcbdb4816add7800502bd5122                       ␣
↪                 |
+----------------------------+---------------------------------------------------
↪--------------+
```

Note: To enable event alarms please refer Configuration

Alarm retrieval

You can display all your alarms via (some attributes are omitted for brevity):

```
$ aodh alarm list
+----------+-----------+--------+-------------------+----------+---------+
| alarm_id | type      | name   | state             | severity | enabled |
+----------+-----------+--------+-------------------+----------+---------+
| ALARM_ID | threshold | cpu_hi | insufficient data | low      | True    |
+----------+-----------+--------+-------------------+----------+---------+
```

In this case, the state is reported as `insufficient data` which could indicate that:

- meters have not yet been gathered about this instance over the evaluation window into the recent past (for example a brand-new instance)
- *or*, that the identified instance is not visible to the user/project owning the alarm

- *or*, simply that an alarm evaluation cycle hasn't kicked off since the alarm was created (by default, alarms are evaluated once per minute).

Note: The visibility of alarms depends on the role and project associated with the user issuing the query:

- admin users see *all* alarms, regardless of the owner
- non-admin users see only the alarms associated with their project (as per the normal project segregation in OpenStack)

Alarm update

Once the state of the alarm has settled down, we might decide that we set that bar too low with 70%, in which case the threshold (or most any other alarm attribute) can be updated thusly:

```
$ aodh alarm update ALARM_ID --threshold 75
```

The change will take effect from the next evaluation cycle, which by default occurs every minute.

Most alarm attributes can be changed in this way, but there is also a convenient short-cut for getting and setting the alarm state:

```
$ openstack alarm state get ALARM_ID
$ openstack alarm state set --state ok ALARM_ID
```

Over time the state of the alarm may change often, especially if the threshold is chosen to be close to the trending value of the statistic. You can follow the history of an alarm over its lifecycle via the audit API:

```
$ aodh alarm-history show ALARM_ID
+-----------+------------------+----------------------------------------------
↪-----+----------+
| timestamp | type             | detail                                      ␣
↪     | event_id |
+-----------+------------------+----------------------------------------------
↪-----+----------+
| TIME_3    | rule change      | {"rule": {"evaluation_periods": 3, "metric":␣
↪     | EVENT_ID |
|           |                  | "cpu_util", "resource_id": RESOURCE_ID,     ␣
↪     |          |
|           |                  | "aggregation_method": "mean", "granularity
↪":600,  |          |
|           |                  | "threshold": 75.0, "comparison_operator": "gt
↪"    |          |
|           |                  | "resource_type": "instance"}}               ␣
↪     |          |
| TIME_2    | state transition | {"transition_reason": "Transition to alarm␣
↪due 3  | EVENT_ID |
|           |                  | samples outside threshold, most recent:     ␣
↪     |          |
|           |                  | 81.4108514719", "state": "alarm"}           ␣
↪     |          |
```

(continues on next page)

(continued from previous page)

```
| TIME_1     | state transition | {"transition_reason": "Transition to ok due␣
↪to 1   | EVENT_ID |
|            |                  | samples inside threshold, most recent:      ␣
↪     |          |
|            |                  | 67.952938019089", "state": "ok"}            ␣
↪     |          |
| TIME_0     | creation         | {"alarm_actions": ["log://"], "user_id":␣
↪USER_ID, | EVENT_ID |
|            |                  | "name": "cup_hi", "state": "insufficient data
↪",   |          |
|            |                  | "timestamp": TIME_0, "description":
↪"instance      |          |
|            |                  | running hot", "enabled": true, "state_
↪timestamp": |          |
|            |                  | TIME_0, "rule": {"evaluation_periods": 3,   ␣
↪     |          |
|            |                  | "metric": "cpu_util", "resource_id":␣
↪RESOURCE_ID, |          |
|            |                  | "aggregation_method": "mean", "granularity":␣
↪600, |          |
|            |                  | "resource_type": "instance"}, "alarm_id":   ␣
↪     |          |
|            |                  | ALARM_ID, "time_constraints": [],           ␣
↪     |          |
|            |                  | "insufficient_data_actions": [],            ␣
↪     |          |
|            |                  | "repeat_actions": false, "ok_actions": [],  ␣
↪     |          |
|            |                  | "project_id": PROJECT_ID, "type":           ␣
↪     |          |
|            |                  | "gnocchi_resources_threshold", "severity":
↪"low"} |          |
+------------+------------------+-------------------------------------------
↪-----+----------+
```

Alarm deletion

An alarm that is no longer required can be disabled so that it is no longer actively evaluated:

```
$ aodh alarm update --enabled False ALARM_ID
```

or even deleted permanently (an irreversible step):

```
$ aodh alarm delete ALARM_ID
```

Debug alarms

A good place to start is to add `--debug` flag when creating or updating an alarm. For example:

```
$ aodh --debug alarm create <OTHER_PARAMS>
```

Look for the state to transition when event is triggered in `/var/log/aodh/listener.log` file. For example, the below logs shows the transition state of alarm with id `85a2942f-a2ec-4310-baea-d58f9db98654` triggered by event id `abe437a3-b75b-40b4-a3cb-26022a919f5e`

```
2017-07-15 07:03:20.149 2866 INFO aodh.evaluator [-] alarm 85a2942f-a2ec-4310-
↪baea-d58f9db98654 transitioning to alarm because Event <id=abe437a3-b75b-
↪40b4-a3cb-26022a919f5e,event_type=compute.instance.power_off.start> hits␣
↪the query <query=[{"field": "traits.instance_id", "op": "eq", "type":
↪"string", "value": "bb912729-fa51-443b-bac6-bf4c795f081d"}]>.
```

The below entry in `/var/log/aodh/notifier.log` also confirms that event id `abe437a3-b75b-40b4-a3cb-26022a919f5e` hits the query matching instance id `bb912729-fa51-443b-bac6-bf4c795f081d`

```
2017-07-15 07:03:24.071 2863 INFO aodh.notifier.log [-] Notifying alarm␣
↪instance_off 85a2942f-a2ec-4310-baea-d58f9db98654 of low priority from␣
↪insufficient data to alarm with action log: because Event <id=abe437a3-b75b-
↪40b4-a3cb-26022a919f5e,event_type=compute.instance.power_off.start> hits␣
↪the query <query=[{"field": "traits.instance_id", "op": "eq", "type":
↪"string", "value": "bb912729-fa51-443b-bac6-bf4c795f081d"}]>
```

`aodh alarm-history` as mentioned earlier will also display the transition:

```
$ aodh alarm-history show 85a2942f-a2ec-4310-baea-d58f9db98654
+----------------------------+------------------+-----------------------------
↪-----------------------------------------------------------------------------
↪-----------------+--------------------------------------+
| timestamp                  | type             | detail                     ␣
↪                                                                            ␣
↪                 | event_id                             |
+----------------------------+------------------+-----------------------------
↪-----------------------------------------------------------------------------
↪-----------------+--------------------------------------+
| 2017-07-15T01:33:20.390623 | state transition | {"transition_reason":
↪"Event <id=abe437a3-b75b-40b4-a3cb-26022a919f5e,event_type=compute.instance.
↪power_off.start> hits  | c5ca92ae-584b-4da6-a12c-b7a00dd39fef |
|                            |                  | the query <query=[{\"field\
↪": \"traits.instance_id\", \"op\": \"eq\", \"type\": \"string\", \"value\":␣
↪\"bb912729-fa51   |                                      |
|                            |                  | -443b-bac6-bf4c795f081d\"}]>
↪.", "state": "alarm"}                                                       ␣
↪                 |                                      |
| 2017-07-15T01:31:14.516188 | creation         | {"alarm_actions": ["log://
↪"], "user_id": "89b4e48bcbdb4816add7800502bd5122", "name": "instance_off",
↪"state":            | fb31f4c2-e357-44c3-9b6a-bd2aaaa4ae68 |
```

(continues on next page)

(continued from previous page)

```
|                             |                  | "insufficient data",
↪"timestamp": "2017-07-15T01:31:14.516188", "description": "event_instance_
↪power_off", "enabled":    |                                       |
|                             |                  | true, "state_timestamp":
↪"2017-07-15T01:31:14.516188", "rule": {"query": [{"field": "traits.instance_
↪id", "type":         |                                       |
|                             |                  | "string", "value":
↪"bb912729-fa51-443b-bac6-bf4c795f081d", "op": "eq"}], "event_type":
↪"compute.instance.power_off.*"},  |                                       |
|                             |                  | "alarm_id": "85a2942f-a2ec-
↪4310-baea-d58f9db98654", "time_constraints": [], "insufficient_data_actions
↪": ["log://"],     |                                       |
|                             |                  | "repeat_actions": false,
↪"ok_actions": ["log://"], "project_id": "9ee200732f4c4d10a6530bac746f1b6e",
↪"type": "event",    |                                       |
|                             |                  | "severity": "low"}         ␣
↪                                                                           ␣
↪                 |                                       |
+-----------------------------+------------------+----------------------------
↪---------------------------------------------------------------------------
↪-----------------+---------------------------------------+
```

3.2 Resource Quota Management

The amount of resources(e.g. alarms) that could be created by each OpenStack project is controlled by quota. The default resource quota for each project is set in Aodh config file as follows unless changed by the cloud administrator via Quota API.

```
[api]
user_alarm_quota = -1
project_alarm_quota = -1
alarm_max_actions = -1
```

user_alarm_quota The default alarm quota for an openstack user, default is unlimited. Sometimes the alarm creation request satisfies the project quota but fails the user quota.

project_alarm_quota The default alarm quota for an openstack project, default is unlimited. The cloud administrator can change project quota using Quota API, see examples below.

alarm_max_actions The maximum number of alarm actions could be created per alarm, default is unlimited.

3.2.1 Quota API

Aodh Quota API is aiming for multi-tenancy support. By default, only the admin user is able to change the resource quota for projects as defined by the default policy rule 'telemetry:update_quotas'. User alarm quota and alarm action quota are not supported in Quota API.

An HTTP request example using `httpie` command:

```
cat <<EOF | http post v2/quotas X-Auth-Token:$token
{
  "project_id": "8aecc55977714e898281c24260747d78",
  "quotas": [
    {
      "resource": "alarms",
      "limit": 30
    }
  ]
}
EOF
```

CHAPTER

FOUR

CONFIGURATION GUIDE

4.1 Aodh Sample Configuration File

Configure Aodh by editing /etc/aodh/aodh.conf.

No config file is provided with the source code, it will be created during the installation. In case where no configuration file was installed, one can be easily created by running:

```
aodh-config-generator
```

4.2 Aodh Configuration Options

4.2.1 aodh: DEFAULT

`record_history`

> **Type** boolean
>
> **Default** `True`

Record alarm change events.

`event_alarm_cache_ttl`

> **Type** integer
>
> **Default** `60`

TTL of event alarm caches, in seconds. Set to 0 to disable caching.

`additional_ingestion_lag`

> **Type** integer
>
> **Default** `0`
>
> **Minimum Value** 0

The number of seconds to extend the evaluation windows to compensate the reporting/ingestion lag.

`member_creation_time`

Type integer

Default `120`

The time in seconds to wait for the load balancer member creation.

`rest_notifier_certificate_file`

Type string

Default `''`

SSL Client certificate file for REST notifier.

`rest_notifier_certificate_key`

Type string

Default `''`

SSL Client private key file for REST notifier.

`rest_notifier_ca_bundle_certificate_path`

Type string

Default `<None>`

SSL CA_BUNDLE certificate for REST notifier

`rest_notifier_ssl_verify`

Type boolean

Default `True`

Whether to verify the SSL Server certificate when calling alarm action.

`rest_notifier_max_retries`

Type integer

Default `0`

Number of retries for REST notifier

`notifier_topic`

Type string

Default `alarming`

The topic that aodh uses for alarm notifier messages.

`http_timeout`

Type integer

Default `600`

Timeout seconds for HTTP requests. Set it to None to disable timeout.

`evaluation_interval`

Type integer

Default `60`

Period of evaluation cycle, should be >= than configured pipeline interval for collection of underlying meters.

4.2.2 aodh: api

`paste_config`

Type string

Default `api-paste.ini`

Configuration file for WSGI definition of API.

`auth_mode`

Type string

Default `keystone`

Authentication mode to use. Unset to disable authentication

`gnocchi_external_project_owner`

Type string

Default `service`

Project name of resources creator in Gnocchi. (For example the Ceilometer project name

`gnocchi_external_domain_name`

Type string

Default `Default`

Domain name of resources creator in Gnocchi. (For example, default or service_domain

`user_alarm_quota`

Type integer

Default `-1`

Maximum number of alarms defined for a user.

Table 1: Deprecated Variations

Group	Name
DEFAULT	user_alarm_quota

`project_alarm_quota`

Type integer

Default `-1`

Maximum number of alarms defined for a project.

Table 2: Deprecated Variations

Group	Name
DEFAULT	project_alarm_quota

`alarm_max_actions`

Type integer

Default `-1`

Maximum count of actions for each state of an alarm, non-positive number means no limit.

Table 3: Deprecated Variations

Group	Name
DEFAULT	alarm_max_actions

4.2.3 aodh: coordination

`backend_url`

Type string

Default `<None>`

The backend URL to use for distributed coordination. If left empty, alarm evaluation won't do workload partitioning and will only function correctly if a single instance of the service is running.

`heartbeat_interval`

Type floating point

Default `1.0`

Number of seconds between heartbeats for distributed coordination.

Table 4: Deprecated Variations

Group	Name
coordination	heartbeat

`check_watchers`

Type floating point

Default `10.0`

Number of seconds between checks to see if group membership has changed

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** This parameter is no longer used.

`retry_backoff`

Type integer

Default `1`

Retry backoff factor when retrying to connect with coordination backend

`max_retry_interval`

Type integer

Default `30`

Maximum number of seconds between retry to join partitioning group

4.2.4 aodh: database

`alarm_history_time_to_live`

Type integer

Default `-1`

Number of seconds that alarm histories are kept in the database for (<= 0 means forever).

`alarm_histories_delete_batch_size`

Type integer

Default `0`

Minimum Value 0

Number of alarm histories to be deleted in one iteration from the database (0 means all).

4.2.5 aodh: evaluator

`workers`

Type integer

Default 1

Minimum Value 1

Number of workers for evaluator service. default value is 1.

4.2.6 aodh: listener

`workers`

Type integer

Default 1

Minimum Value 1

Number of workers for listener service. default value is 1.

`event_alarm_topic`

Type string

Default `alarm.all`

The topic that aodh uses for event alarm evaluation.

Table 5: Deprecated Variations

Group	Name
DEFAULT	event_alarm_topic

`batch_size`

Type integer

Default 1

Number of notification messages to wait before dispatching them.

`batch_timeout`

Type integer

Default `<None>`

Number of seconds to wait before dispatching samples when batch_size is not reached (None means indefinitely).

4.2.7 aodh: notifier

`batch_size`

Type integer

Default `1`

Number of notification messages to wait before dispatching them.

`batch_timeout`

Type integer

Default `<None>`

Number of seconds to wait before dispatching samples when batch_size is not reached (None means indefinitely).

`workers`

Type integer

Default `1`

Minimum Value 1

Number of workers for notifier service. default value is 1.

4.2.8 aodh: service_credentials

`region_name`

Type string

Default `<None>`

Region name to use for OpenStack service endpoints.

Table 6: Deprecated Variations

Group	Name
service_credentials	os-region-name
service_credentials	os_region_name

`interface`

Type string

Default `public`

Valid Values public, internal, admin, auth, publicURL, internalURL, adminURL

Type of endpoint in Identity service catalog to use for communication with OpenStack services.

Table 7: Deprecated Variations

Group	Name
service_credentials	os-endpoint-type
service_credentials	os_endpoint_type

4.2.9 aodh: service_types

`zaqar`

Type string

Default `messaging`

Message queue service type.

4.3 Aodh Sample Policy Configuration File

Warning: JSON formatted policy file is deprecated since Aodh 12.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is an overview of all available policies in Aodh. For a sample configuration file, refer to *policy.yaml*.

4.3.1 aodh

`context_is_admin`

Default `role:admin`

(no description provided)

`segregation`

Default `rule:context_is_admin`

(no description provided)

`admin_or_owner`

Default `rule:context_is_admin or project_id:%(project_id)s`

(no description provided)

`default`

Default `rule:context_is_admin or project_id:%(project_id)s`

(no description provided)

`telemetry:get_alarm`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **GET** `/v2/alarms/{alarm_id}`

Scope Types

- **system**
- **project**

Get an alarm.

`telemetry:get_alarms`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **GET** `/v2/alarms`

Scope Types

- **system**
- **project**

Get all alarms, based on the query provided.

`telemetry:get_alarms:all_projects`

Default `role:reader and system_scope:all`

Operations

- **GET** `/v2/alarms`

Scope Types

- **system**
- **project**

Get alarms of all projects.

`telemetry:query_alarm`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **POST** `/v2/query/alarms`

Scope Types

- **system**
- **project**

Get all alarms, based on the query provided.

`telemetry:create_alarm`

Default `(role:admin and system_scope:all) or (role:member and project_id:%(project_id)s)`

Operations

- **POST** `/v2/alarms`

Scope Types

- **system**
- **project**

Create a new alarm.

`telemetry:change_alarm`

Default `(role:admin and system_scope:all) or (role:member and project_id:%(project_id)s)`

Operations

- **PUT** `/v2/alarms/{alarm_id}`

Scope Types

- **system**
- **project**

Modify this alarm.

`telemetry:delete_alarm`

Default `(role:admin and system_scope:all) or (role:member and project_id:%(project_id)s)`

Operations

- **DELETE** `/v2/alarms/{alarm_id}`

Scope Types

- **system**
- **project**

Delete this alarm.

`telemetry:get_alarm_state`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **GET** `/v2/alarms/{alarm_id}/state`

Scope Types

- **system**
- **project**

Get the state of this alarm.

`telemetry:change_alarm_state`

Default `(role:admin and system_scope:all) or (role:member and project_id:%(project_id)s)`

Operations

- **PUT** `/v2/alarms/{alarm_id}/state`

Scope Types

- **system**
- **project**

Set the state of this alarm.

`telemetry:alarm_history`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **GET** `/v2/alarms/{alarm_id}/history`

Scope Types

- **system**
- **project**

Assembles the alarm history requested.

`telemetry:query_alarm_history`

Default `(role:reader and system_scope:all) or (role:reader and project_id:%(project_id)s)`

Operations

- **POST** `/v2/query/alarms/history`

Scope Types

- **system**
- **project**

Define query for retrieving AlarmChange data.

`telemetry:update_quotas`

Default `role:admin and system_scope:all`

Operations

- **POST** `/v2/quotas`

Scope Types

- **system**

Update resources quotas for project.

`telemetry:delete_quotas`

Default `role:admin and system_scope:all`

Operations

- **DELETE** `/v2/quotas/{project_id}`

Scope Types

- **system**

Delete resources quotas for project.

4.4 policy.yaml

Warning: JSON formatted policy file is deprecated since Aodh 12.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

Use the `policy.yaml` file to define additional access controls that will be applied to Aodh:

```
#"context_is_admin": "role:admin"

#"segregation": "rule:context_is_admin"

#"admin_or_owner": "rule:context_is_admin or project_id:%(project_id)s"

#"default": "rule:context_is_admin or project_id:%(project_id)s"

# Get an alarm.
# GET  /v2/alarms/{alarm_id}
# Intended scope(s): system, project
#"telemetry:get_alarm": "(role:reader and system_scope:all) or (role:reader␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:get_alarm":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:get_alarm":"(role:reader and system_scope:all) or
# (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Get all alarms, based on the query provided.
# GET  /v2/alarms
# Intended scope(s): system, project
#"telemetry:get_alarms": "(role:reader and system_scope:all) or (role:reader␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:get_alarms":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
```

(continues on next page)

(continued from previous page)

```
# "telemetry:get_alarms":"(role:reader and system_scope:all) or
# (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Get alarms of all projects.
# GET  /v2/alarms
# Intended scope(s): system, project
#"telemetry:get_alarms:all_projects": "role:reader and system_scope:all"

# DEPRECATED
# "telemetry:get_alarms:all_projects":"rule:context_is_admin" has been
# deprecated since W in favor of
# "telemetry:get_alarms:all_projects":"role:reader and
# system_scope:all".
# The alarm and quota APIs now support system-scope and default roles.

# Get all alarms, based on the query provided.
# POST  /v2/query/alarms
# Intended scope(s): system, project
#"telemetry:query_alarm": "(role:reader and system_scope:all) or (role:reader␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:query_alarm":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:query_alarm":"(role:reader and system_scope:all) or
# (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Create a new alarm.
# POST  /v2/alarms
# Intended scope(s): system, project
#"telemetry:create_alarm": "(role:admin and system_scope:all) or (role:member␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:create_alarm":"" has been deprecated since W in favor of
# "telemetry:create_alarm":"(role:admin and system_scope:all) or
# (role:member and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Modify this alarm.
# PUT  /v2/alarms/{alarm_id}
# Intended scope(s): system, project
#"telemetry:change_alarm": "(role:admin and system_scope:all) or (role:member␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:change_alarm":"rule:context_is_admin or
```

(continues on next page)

(continued from previous page)

```
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:change_alarm":"(role:admin and system_scope:all) or
# (role:member and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Delete this alarm.
# DELETE  /v2/alarms/{alarm_id}
# Intended scope(s): system, project
#"telemetry:delete_alarm": "(role:admin and system_scope:all) or (role:member␣
↪and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:delete_alarm":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:delete_alarm":"(role:admin and system_scope:all) or
# (role:member and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Get the state of this alarm.
# GET  /v2/alarms/{alarm_id}/state
# Intended scope(s): system, project
#"telemetry:get_alarm_state": "(role:reader and system_scope:all) or␣
↪(role:reader and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:get_alarm_state":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:get_alarm_state":"(role:reader and system_scope:all) or
# (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Set the state of this alarm.
# PUT  /v2/alarms/{alarm_id}/state
# Intended scope(s): system, project
#"telemetry:change_alarm_state": "(role:admin and system_scope:all) or␣
↪(role:member and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:change_alarm_state":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:change_alarm_state":"(role:admin and system_scope:all) or
# (role:member and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Assembles the alarm history requested.
# GET  /v2/alarms/{alarm_id}/history
# Intended scope(s): system, project
#"telemetry:alarm_history": "(role:reader and system_scope:all) or␣
↪(role:reader and project_id:%(project_id)s)"
```

(continues on next page)

(continued from previous page)

```
# DEPRECATED
# "telemetry:alarm_history":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:alarm_history":"(role:reader and system_scope:all) or
# (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Define query for retrieving AlarmChange data.
# POST  /v2/query/alarms/history
# Intended scope(s): system, project
#"telemetry:query_alarm_history": "(role:reader and system_scope:all) or␣
↪(role:reader and project_id:%(project_id)s)"

# DEPRECATED
# "telemetry:query_alarm_history":"rule:context_is_admin or
# project_id:%(project_id)s" has been deprecated since W in favor of
# "telemetry:query_alarm_history":"(role:reader and system_scope:all)
# or (role:reader and project_id:%(project_id)s)".
# The alarm and quota APIs now support system-scope and default roles.

# Update resources quotas for project.
# POST  /v2/quotas
# Intended scope(s): system
#"telemetry:update_quotas": "role:admin and system_scope:all"

# DEPRECATED
# "telemetry:update_quotas":"rule:context_is_admin" has been
# deprecated since W in favor of "telemetry:update_quotas":"role:admin
# and system_scope:all".
# The alarm and quota APIs now support system-scope and default roles.

# Delete resources quotas for project.
# DELETE  /v2/quotas/{project_id}
# Intended scope(s): system
#"telemetry:delete_quotas": "role:admin and system_scope:all"

# DEPRECATED
# "telemetry:delete_quotas":"rule:context_is_admin" has been
# deprecated since W in favor of "telemetry:delete_quotas":"role:admin
# and system_scope:all".
# The alarm and quota APIs now support system-scope and default roles.
```

CHAPTER

FIVE

AODH CLI DOCUMENTATION

In this section you will find information on Aodhs command line interface.

5.1 aodh-status

5.1.1 CLI interface for Aodh status commands

Synopsis

```
aodh-status <category> <command> [<args>]
```

Description

aodh-status is a tool that provides routines for checking the status of a Aodh deployment.

Options

The standard pattern for executing a **aodh-status** command is:

```
aodh-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
aodh-status
```

Categories are:

- `upgrade`

Detailed descriptions are below:

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
aodh-status upgrade
```

These sections describe the available categories and arguments for **aodh-status**.

Upgrade

`aodh-status upgrade check` Performs a release-specific readiness check before restarting services with new code. For example, missing or changed configuration options, incompatible object states, or other conditions that could lead to failures while upgrading.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

8.0.0 (Stein)

- Sample check to be filled in with checks as they are added in Stein.

CHAPTER SIX

GLOSSARY

alarm An action triggered whenever a meter reaches a certain threshold.

API server HTTP REST API service for Aodh.

HTTP callback HTTP callback is used for calling a predefined URL, whenever an alarm has been set off. The payload of the request contains all the details of why the alarm was triggered.

log Logging is one of the alarm actions that is useful mostly for debugging, it stores the alarms in a log file.

zaqar According to Zaqar Developer Documentation:

> Zaqar is a multi-tenant cloud messaging and notification service for web and mobile developers.

project The OpenStack tenant or project.

resource The OpenStack entity being metered (e.g. instance, volume, image, etc).

user An OpenStack user.