almanach Documentation

Release

OpenStack Foundation

Aug 14, 2017

Contents

Almanach stores the utilization of OpenStack resources (instances and volumes) for each tenant.

CHAPTER 1

What is Almanach?

The main purpose of this software is to record the usage of the cloud resources of each tenants.

Almanach is composed of two parts:

- **Collector**: Listen for OpenStack events and store the relevant information in the database.
- **REST API**: Expose the information collected to external systems.

CHAPTER 2

Requirements

- OpenStack infrastructure installed (Nova, Cinder...)
- MongoDB
- Python 2.7, 3.4 or 3.5

CHAPTER 3

Generate config file with default values

```
tox -e genconfig
```

CHAPTER 4

Command line usage

Start the API daemon:

```
almanach-api --config-file /etc/almanach/almanach.conf
```

Start the collector:

```
almanach-collector --config-file /etc/almanach/almanach.conf
```

CHAPTER 5

Signal Handling

- `SIGINT`: force instantaneous termination
- `SIGTERM`: graceful termination of the service
- `SIGHUP`: reload service

CHAPTER 6

Authentication

Protocol

The authentication mechanism use the HTTP header `X-Auth-Token` to send a token. This token is validated through Keystone or with the config file (private secret key).

```
GET /volume_types HTTP/1.1
X-Auth-Token: secret
Content-Type: application/json

{}
```

If the token is not valid, you will receive a `401 Not Authorized` response.

Private Key Authentication

The private secret key authentication is the default method. In your config file, you have to define your private key in the field `auth_token`:

```
[auth]
strategy = private_key
private_key = secret
```

Keystone Authentication

The token will be validated with Keystone. To use this authentication backend you have to define the authentication strategy to `keystone`.

```
[auth]
strategy = keystone

[keystone_authtoken]

# Keystone service username (string value)
username = almanach

# Keystone service password (string value)
password = secret

# Keystone service user domain ID (string value)
user_domain_id = default

# Keystone service user domain name (string value)
user_domain_name = Default

# Keystone service project domain name (string value)
project_domain_name = Default

# Keystone service project name (string value)
project_name = service

# Keystone API V3 admin endpoint (string value)
auth_url = http://127.0.0.1:35357/v3
```

CHAPTER 7

RabbitMQ configuration

Each OpenStack services (Nova, Cinder, Neutron) need to be configured to send notifications to the Almanach queue.

For example with Nova, add the topic "almanach" in the config file `/etc/nova.conf`:

```
notification_topics=almanach
```

CHAPTER 8

MongoDB configuration

Almanach requires a specific user to connect to the database. To create a new user, open a new MongoDB shell:

```
m = new Mongo()
m.getDB("almanach").createUser({user: "almanach", pwd: "almanach", roles: [{role:
↪"readWrite", db: "almanach"}]})
```

CHAPTER 9

Devstack configuration

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD

enable_plugin almanach https://git.openstack.org/openstack/almanach
```

CHAPTER 10

Database entities

Each entity have at least these properties:

- `entity_id`: Unique id for the entity (UUID)
- `entity_type`: “instance” or “volume”
- `project_id`: Tenant unique ID (UUID)
- `start`: Start date of the resource usage
- `end`: End date of the resource usage or `null` if the resource still in use by the tenant
- `name`: Resource name

Compute Object

```
{
    "entity_id": "UUID",
    "entity_type": "instance",
    "project_id": "UUID",
    "start": "2014-01-01T06:00:00.000Z",
    "end": null,
    "last_event": "2014-01-01T06:00:00.000Z",
    "flavor": "MyFlavor1",
    "os": {
        "distro": "ubuntu",
        "version": "14.04"
    },
    "name": "my-virtual-machine.domain.tld"
}
```

Block Storage Object

```
{
    "entity_id": "UUID",
    "entity_type": "volume",
    "project_id": "UUID",
    "start": "2014-01-01T06:00:00.000Z",
    "end": null,
    "last_event": "2014-01-01T06:00:00.000Z",
    "volume_type": "MyVolumeType",
    "size": 50,
    "name": "my-virtual-machine.domain.tld-volume",
    "attached_to": "UUID"
}
```

CHAPTER 11

List of events handled

Almanach will process those events:

- `compute.instance.create.end`
- `compute.instance.delete.end`
- `compute.instance.resize.confirm.end`
- `compute.instance.rebuild.end`
- `volume.create.end`
- `volume.delete.end`
- `volume.resize.end`
- `volume.attach.end`
- `volume.detach.end`
- `volume.update.end`
- `volume.exists`
- `volume_type.create`

CHAPTER 12

API v1 Documentation

`GET /v1/volume_types`

List volume types.

Status Codes:

- **200 OK** Volume types exist

Example output:

```
[
  {
    "volume_type_id": "8b2944c2-9268-4fca-a5df-b4f23a7af1ba",
    "volume_type_name": "my_volume_type1"
  },
  {
    "volume_type_id": "a1c73195-d54e-4aea-8c3e-3df017b7a44a",
    "volume_type_name": "my_volume_type2"
  }
]
```

`GET /v1/volume_type/<volume_type_id>`

Get a volume type.

Status Codes:

- **200 OK** Volume type exist
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the volume type does not exist

Request:

Name	In	Type	Description
volume_type_id	path	uuid	The Volume Type Uuid

Example output:

```
{
  "volume_type_id": "8b2944c2-9268-4fca-a5df-b4f23a7af1ba",
  "volume_type_name": "my_volume_type1"
}
```

`POST /v1/volume_type`

Create a volume type.

Status Codes:

- **201 Created** Volume type successfully created
- **400 Bad Request** If request data has an invalid or missing field

Request:

Name	In	Type	Description
type_id	body	uuid	The Volume Type Uuid
type_name	body	string	The Volume Type Name

Example input:

```
{
  "type_id": "ae23091d-caf5-44f9-ae7d-2be3623c5e3a",
  "type_name": "my_volume_type3"
}
```

`DELETE /v1/volume_type/<volume_type_id>`

Delete a volume type.

Status Codes:

- **202 Accepted** Volume type successfully deleted
- **404 Not Found** If the volume type does not exist

Request:

Name	In	Type	Description
volume_type_id	path	uuid	The Volume Type Uuid

`GET /v1/info`

Display information about the current version and entity counts.

Status Codes:

- **200 OK** Service is available

Example output:

```
{
  "info": {
    "version": "3.2.0"
  },
  "database": {
    "all_entities": 999,
    "active_entities": 997
  }
}
```

`POST /v1/project/<project_id>/instance`

Create an instance.

Status Codes:

- **201 Created** Instance successfully created
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If tenant does not exist

Request:

Name	In	Type	Description
project_id	path	uuid	The Tenant Uuid
id	body	uuid	The instance Uuid
created_at	body	datetime	Y-m-d H:M:S.f
flavor	body	uuid	The flavor Uuid
os_type	body	string	The OS type
os_distro	body	string	The OS distro
os_version	body	string	The OS version
name	body	string	The instance name

Example input:

```
{
  "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
  "id": "460bb2b6-28d6-42c0-9da4-4288dc3025cc",
  "created_at": "2016-11-24 15:15:05+00:00",
  "flavor": "946d7b8f-b20a-4a1d-bf43-567fa27db614",
  "os_type": "linux",
  "os_version": "7",
  "os_distro": "centos",
  "name": "created_instance1"
}
```

`DELETE /v1/instance/<instance_id>`

Delete an instance.

Status Codes:

- **202 Accepted** Instance successfully deleted
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the instance does not exist

Request:

Name	In	Type	Description
instance_id	path	uuid	The instance Uuid
date	body	datetime	Y-m-d H:M:S.f

Example input:

```
{
  "date": "2016-11-24 17:55:05+00:00"
}
```

`PUT /v1/instance/<instance_id>/resize`

Re-size an instance.

Status Codes:

- **202 Accepted** Instance successfully re-sized
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the instance does not exist

Request:

Name	In	Type	Description
instance_id	path	uuid	The instance Uuid
date	body	datetime	Y-m-d H:M:S.f
flavor	body	uuid	The flavor Uuid

Example input:

```
{
  "flavor": "173a54a0-fd55-423e-8084-1ef66d1b428a",
  "date": "2016-11-24 17:25:05+00:00"
}
```

`PUT /v1/instance/<instance_id>/rebuild`

Rebuild an instance.

Status Codes:

- **202 Accepted** Instance successfully rebuilt
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the instance does not exist

Request:

Name	In	Type	Description
instance_id	path	uuid	The instance Uuid
rebuild_date	body	datetime	Y-m-d H:M:S.f
os_type	body	string	The OS type
os_distro	body	string	The OS distro
os_version	body	string	The OS version

Example input:

```
{
  "instance_id": "37870255-a0f0-447c-b602-7e29f32cc88c",
  "rebuild_date": "2016-11-24 15:15:05+00:00",
  "os_type": "linux",
  "version": "14.04",
  "distro": "ubuntu"
}
```

`GET /v1/project/<project_id>/instances`

List instances for a tenant.

Status Codes:

- **200 OK** Instances exist
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the tenant does not exist

Request:

Name	In	Type	Description
project_id	path	uuid	The Tenant Uuid
start	path	datetime	Y-m-d H:M:S.f
end	path	datetime	Y-m-d H:M:S.f

Example output:


```
[
  {
    "entity_id": "7f8284db-c955-4383-b253-d54cbc8c4364",
    "end": null,
    "name": "host1.com",
    "last_event": "2016-11-24 15:14:08+00:00",
    "entity_type": "instance",
    "start": "2016-11-24 15:14:08+00:00",
    "flavor": "173a54a0-fd55-423e-8084-1ef66d1b428a",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "os": {
      "os_type": "linux",
      "version": "7",
      "distro": "centos"
    },
    "metadata": {}
  }
]
```


`POST /v1/project/<project_id>/volume`

Create a volume.

Status Codes:

- **201 Created** Volume successfully created
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If tenant does not exist

Request:

Name	In	Type	Description
project_id	path	uuid	The Tenant Uuid
volume_id	body	uuid	The volume Uuid
start	body	datetime	Y-m-d H:M:S.f
volume_type	body	uuid	The volume type Uuid
size	body	string	The volume size
volume_name	body	string	The volume name
attached_to	body	uuid	The instance uuid the volume is attached to

Example input:


```
{
  "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
  "volume_id": "a1c95ee7-3317-4597-b176-131209368d27",
  "start": "2016-11-24 17:15:05+00:00",
  "volume_type": "8b2944c2-9268-4fca-a5df-b4f23a7af1ba",
  "size": 20,
  "volume_name": "created_volume2",
  "attached_to": ""
}
```

`DELETE /v1/volume/<volume_id>`

Delete a volume.

Status Codes:

- **202 Accepted** Volume successfully deleted
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the volume does not exist

Request:

Name	In	Type	Description
volume_id	path	uuid	The volume Uuid
date	body	datetime	Y-m-d H:M:S.f

Example input:

```
{
  "date": "2016-11-24 17:55:05+00:00"
}
```

`PUT /v1/volume/<volume_id>/resize`

Re-size a volume.

Status Codes:

- **202 Accepted** Volume successfully re-sized
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the volume does not exist

Request:

Name	In	Type	Description
volume_id	path	uuid	The volume Uuid
date	body	datetime	Y-m-d H:M:S.f
size	body	string	The volume size

Example input:

```
{
  "size": 22,
  "date": "2016-11-24 17:25:05+00:00"
}
```

`PUT /v1/volume/<volume_id>/attach`

Update the attachments for a volume.

Status Codes:

- **202 Accepted** Volume successfully attached
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the volume does not exist

Request:

Name	In	Type	Description
volume_id	path	uuid	The volume Uuid
date	body	datetime	Y-m-d H:M:S.f
attachments	body	dict	The volume attachments

Example input:

```
{
  "date": "2016-11-24 17:15:05+00:00",
  "attachments": ["460bb2b6-28d6-42c0-9da4-4288dc3025cc"]
}
```

`PUT /v1/volume/<volume_id>/detach`

Detach a volume.

Status Codes:

- **202 Accepted** Volume successfully detached
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the volume does not exist

Request:

Name	In	Type	Description
volume_id	path	uuid	The volume Uuid
date	body	datetime	Y-m-d H:M:S.f
attachments	body	dict	The volume attachments

Example input:

```
{
  "date": "2016-11-24 17:25:05+00:00",
  "attachments": ["460bb2b6-28d6-42c0-9da4-4288dc3025cc"]
}
```

`GET /v1/project/<project_id>/volumes`

List volumes for a tenant.

Status Codes:

- **200 OK** Volumes exist
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the tenant does not exist

Request:

Name	In	Type	Description
project_id	path	uuid	The Tenant Uuid
start	path	datetime	Y-m-d H:M:S.f
end	path	datetime	Y-m-d H:M:S.f

Example output:

```
[
  {
    "entity_id": "020f3636-6a8a-4a37-beb0-0735074175a9",
    "attached_to": ["b5a4b119-7444-4993-afda-89b8f8f70147"],
    "end": null,
```

```
    "name": "my.host.name.com-volume",
    "last_event": "2016-11-24 21:16:47.106000+00:00",
    "entity_type": "volume",
    "volume_type": "my_volume_type",
    "start": "2016-11-24 21:16:47.106000+00:00",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "size": 20
  },
  {
    "entity_id": "020f3636-6a8a-4a37-beb0-0735074175a9",
    "attached_to": [],
    "end": "2016-11-24 21:16:47.106000+00:00",
    "name": "",
    "last_event": "2016-11-24 21:16:47.106000+00:00",
    "entity_type": "volume",
    "volume_type": "my_volume_type2",
    "start": "2016-11-24 21:15:38+00:00",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "size": 20
  }
]
```


`GET /v1/project/<project_id>/entities`

List entities for a tenant.

Status Codes:

- **200 OK** Entities exist
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the tenant does not exist

Request:

Name	In	Type	Description
project_id	path	uuid	The Tenant Uuid
start	path	datetime	Y-m-d H:M:S.f
end	path	datetime	Y-m-d H:M:S.f

Example output:


```
[
  {
    "entity_id": "b5a4b119-7444-4993-afda-89b8f8f70147",
    "end": null,
    "name": "my.host.name.com",
    "last_event": "2016-11-24 21:15:35+00:00",
    "entity_type": "instance",
    "start": "2016-11-24 21:15:35+00:00",
    "flavor": "my_flavor_name",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "os": {
      "os_type": "linux",
      "version": "7",
      "distro": "centos"
    },
    "metadata": {}
  },
```

```
  {
    "entity_id": "020f3636-6a8a-4a37-beb0-0735074175a9",
    "attached_to": ["b5a4b119-7444-4993-afda-89b8f8f70147"],
    "end": null,
    "name": "my.host.name.com-volume",
    "last_event": "2016-11-24 21:16:47.106000+00:00",
    "entity_type": "volume",
    "volume_type": "my_volume_type1",
    "start": "2016-11-24 21:16:47.106000+00:00",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "size": 20
  }
]
```


`PUT /v1/entity/instance/<instance_id>`

Update an instance.

Status Codes:

- **202 Accepted** Instance successfully updated
- **400 Bad Request** If request data has an invalid or missing field
- **404 Not Found** If the instance does not exist

Request:

Name	In	Type	Description
instance_id	path	uuid	The instance Uuid
start	body	datetime	Y-m-d H:M:S.f
end	body	datetime	Y-m-d H:M:S.f

Example input:

```
{
  "start": "2016-11-24T17:25:05.00Z",
  "end": "2016-11-24T17:35:05.00Z"
}
```

Example output:


```
{
  "entity_id": "460bb2b6-28d6-42c0-9da4-4288dc3025cc",
  "end": "2016-11-24 17:35:05+00:00",
  "name": "my_instance_name",
  "last_event": "2016-11-24 17:25:05+00:00",
  "entity_type": "instance",
  "start": "2016-11-24 17:25:05+00:00",
  "flavor": "my_flavor_name",
  "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
  "os": {
    "os_type": "linux",
    "version": "7",
    "distro": "centos"
  },
  "metadata": {}
}
```


`HEAD /v1/entity/<entity_id>`

Verify that an entity exists.

Status Codes:

- **200 OK** Entity exists
- **404 Not Found** If the entity does not exist

Request:

Name	In	Type	Description
entity_id	path	uuid	The Entity Uuid

Example output:

```
[
  {
    "entity_id": "7f8284db-c955-4383-b253-d54cbc8c4364",
    "end": null,
    "name": "host1.ccom",
    "last_event": "2016-11-24 15:14:08+00:00",
    "entity_type": "instance",
    "start": "2016-11-24 15:14:08+00:00",
    "flavor": "my_flavor_name",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "os": {
      "os_type": "linux",
      "version": "7",
      "distro": "centos"
    },
    "metadata": {}
  }
]
```

`GET /v1/entity/<entity_id>`

Get an entity.

Status Codes:

- **200 OK** If the entity exists
- **404 Not Found** If the entity does not exist

Request:

Name	In	Type	Description
entity_id	path	uuid	The Entity Uuid

Example output:

```
[
  {
    "entity_id": "7f8284db-c955-4383-b253-d54cbc8c4364",
    "end": null,
    "name": "host1.ccom",
    "last_event": "2016-11-24 15:14:08+00:00",
    "entity_type": "instance",
    "start": "2016-11-24 15:14:08+00:00",
    "flavor": "my_flavor_name",
    "project_id": "ce2d9f6bde52447a831887aac8b7ec98",
    "os": {
      "os_type": "linux",
```

```
      "version": "7",
      "distro": "centos"
    },
    "metadata": {}
  }
]
```