Document Properties

Document Title	Blazar Architecture
Task number	4.2.3
Responsible	François Rossigneux
Author(s) / contributor(s)	Jean-Patrick Gelas Laurent Lefèvre François Rossigneux
Document Status	Final
Version	Revision n°1

Summary

The main task was to design a resource reservation service with a scheduling algorithm taking into account the efficiency of each host and managing their standby modes.

Blazar, a resource reservation service, allows the user to book physical machines in order to guarantee a mono-tenancy environment with homogeneous performances suitable for HPC applications. The compute nodes are allocated in an efficient way, taking into account their efficiency, and turned off when they are unused.

The efficiency metric is a mix of the power consumption, collected with Kwapi, and the performance index given by an Unix bench. Each new machine added in the cloud is benchmarked, as for identical CPU, the power consumption may vary up to 20% depending of the manufacturing hazards.

Keywords

reservation, ranking, standby, efficiency

Resource Reservation Architecture

Sommaire

1 Introduction

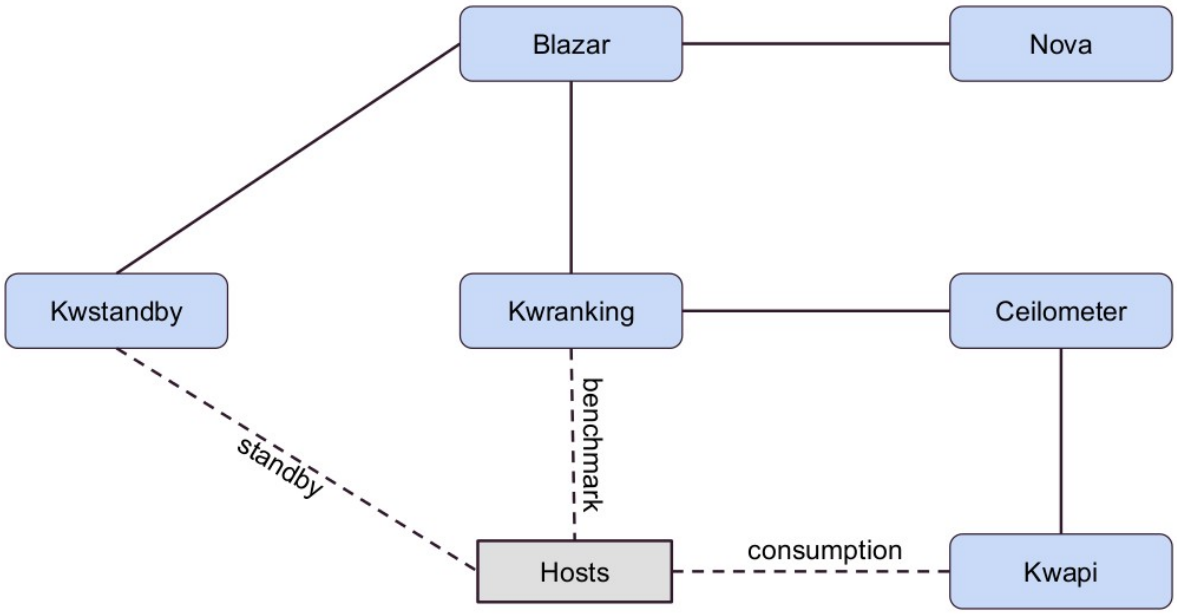
1.1 Purpose

This software design specification provides an overview of design and architecture of the proposed resource management framework for XLcloud project.

1.2 Scope

OpenStack virtual machines scheduling doesn't take into account the energy efficiency criteria. We created a resource reservation service with a scheduling algorithm taking into account the efficiency of each host and managing their standby modes.

2 Architecture Overview



Global Architecture

The proposed architecture includes Blazar, Kwranking and Kwstandby. It depends on Kwapi, Ceilometer and Nova, which have been described in the previous specification.

Roles of the modules:

- Kwapi: takes the power measurements
- Ceilometer: stores the power measurements
- Blazar: manages the resources
- Kwranking: benchmarks the machines
- Kwstandby: wakes up and turns of the machines

3 Blazar

3.1 Introduction

Blazar allows the reservation of physical and virtual resources (physical hosts, instances, network and volumes).

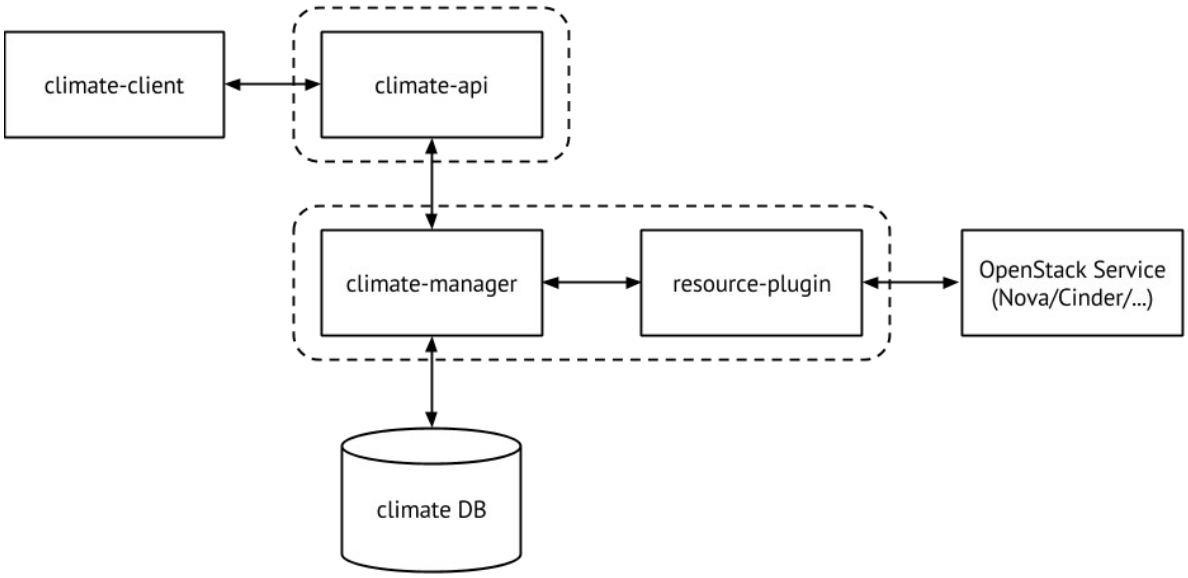
In terms of benefits added, it will:

- improve visibility of cloud resources consumption (current and planned for future);
- enable cloud resource planning based on current and future demand from end users;
- automate the processes of resource allocation and reclaiming;
- provide energy efficiency for physical hosts (both compute and storage ones);
- potentially provide leases as billable items for which customers can be charged a flat fee or a premium price depending on amount of reserved cloud resources and their usage

3.2 Architecture

Is is composed of the following modules:

- **Climate-client:** provides the opportunity to communicate with Blazar via REST API (climate-api service).
- **Climate-api:** waits for the REST calls from the outside world to redirect them to the manager. climate-api communicates with climate-manager via RPC. Runs as a separated process.
- **Climate-manager:** implements all logic and operations with leases, reservations and events. Communicates with Blazar DB and stores there data structure of connected leases, reservations (both physical and virtual) and events. climate-manager service is responsible for running events created for lease and process all actions that should be done this moment. Manager uses resource-plugins to work with concrete resources (instances, volumes, compute hosts). climate-manager uses Keystone trusts to commit actions on behalf of user who has created lease before.
- **Resource-plugin:** responsible for exact actions to do with reserved resources (VMs, volumes, etc.) When working knows only about resource ID and token to use. All resource plugins work in the same process as climate-manager.



Blazar Architecture

3.3 Compute host reservation

Compute hosts reserving contains two steps:

- Admin marks hosts from common pool as possible to be reserved. That is implemented by moving these hosts to special aggregate named freepool.
- User asks for reserving of host with specified characteristics like:
  - the region
  - the availability zone
  - the host capabilities extra specs (scoped and non-scoped format should be accepted)
  - the number of CPU cores
  - the amount of free RAM
  - the amount of free disk space
  - the number of hosts

Resource here will be new aggregate containing reserved hosts. The time lease starts, user may use reserved compute capacity to run his/her instances on it passing special scheduler hint to Nova. When host is reserved, it's not used for usual instance running, it might be used only when lease starts and only by passing reservation ID to Nova.

Blazar supports immediate reservation and in advance reservation.

The eligible hosts are sorted by efficiency, by contacting Kwranking. The reservations are regularly optimized to use always the most efficient hosts and save energy without performance loss. The unused hosts are put in standby modes if there is no new leases on them in the next minutes.

3.4 Installation

Clone the Blazar git repository to the management server:

```
git clone https://github.com/stackforge/blazar
```

As a user with `root` permissions or `sudo` privileges, run the Kwranking installer and copy the configuration files:

```
$ pip install blazar
$ cp -r blazar/etc/ blazar /etc/
```

Start the Blazar services:

```
$ blazar-api
$ blazar-manager
```

3.5 Configuration file

The configuration file is located in /etc/blazar/blazar.conf.

The *manager* section defines the supported reservation plugins:

Option	Default value	Note
plugins	physical.host.plugin	Enable physical host reservation

The *physical:host* section defines the specific options for physical host reservation:

Option	Default value	Note
on_start	on_start	Method to trigger when a lease starts
on_end	on_end	Method to trigger when a lease ends
climate_username	climate	Used for contacting Kwranking and Kwstandby

climate_password	password	Used for contacting Kwranking and Kwstandby
climate_tenant_name	service	Used for contacting Kwranking and Kwstandby

3.6 Specific API

Host management:

Verb	URL	Parameters	Expected result
GET	/v1/os-hosts/		Returns all hosts details
GET	/v1/os-hosts/<host>	host	Returns the host details
PUT	/v1/os-hosts/<host>	host {"values": { "foo": "bar"}}	Update the host values

POST	/v1/os-hosts	{
		"name": "compute",
		"values": {
		"foo": "bar"
		}
		}
			Add an host existing in Nova

DELETE	/v1/os-hosts/<host>	host	Delete an host

Reservation management:

Verb	URL	Parameters	Expected result
GET	/v1/leases		Returns all leases details
GET	/v1/leases/<id>	Lease id	Returns the lease details

POST	/v1/leases	{
		"name": a,
		"start_date": b,
		"end_date": c,
		"reservations": [
		{
		"min": d,
		"max": e,
		"resource_type": "physical:host",
		"resource_properties": f,
		"hypervisor_properties": ""
		}
		]
		}
		
		a = lease name
		b = start date "AAAA-MM-JJ HH:MM"
		c = end date "AAAA-MM-JJ HH:MM"
		d = min host requested
		e = max host requested
		f = resource properties using the Nova JsonFilter syntax
			Creates a physical lease

POST	/v1/leases	{
		"name": a
		}
		
		a = lease name
			Update the lease name

DELETE	/v1/leases/<id>	Lease id	Delete the lease if it is not yet started

3.7 Hardware requirements syntax

The hardware requirements syntax is similar to the Nova JsonFilter syntax.

It provides the opportunity to write complicated queries for the hosts capabilities filtering, based on simple JSON-like syntax. There can be used the following operations for the host states properties: `=`, `<`, `>`, `in`, `<=`, `>=`, that can be combined with logical operations. For example, there is a query:

```
['and',
    ['=', '$vcpus', 4],
    ['>=', '$memory_mb', 4096]
]
```

The keys can be found by doing a request to /v1/os-hosts/.

3.8 Documentation

Documentation can be found at:

https://wiki.openstack.org/wiki/Blazar

3.9 Code repositories

https://github.com/stackforge/blazar

https://github.com/stackforge/python-blazarclient

4 Kwranking

Kwranking provides information about host efficiency.

It deploys UnixBench on remote hosts using SSH with public key authentication and runs arithmetic tests. The returned value is stored in the DB (it never changes over time). During the tests, the max power consumption is reached and stored in Ceilometer, so the flop/w metric is build using the max power value stored in Ceilometer and the flop value returned by the benchmark. The flop/w metric is updated periodically, because the max power consumption may vary over the machine lifetime. An API allows the user to find the most efficient hosts from a list of hosts passed as parameter.

4.1 Installation

Clone the Kwranking git repository to the management server:

```
git clone https://github.com/frossigneux/kwranking
```

As a user with `root` permissions or `sudo` privileges, run the Kwranking installer and copy the configuration files:

```
$ pip install kwranking
$ cp -r kwranking/etc/kwranking /etc/
```

Start the Kwranking API:

```
$ kwranking-api
```

Now add new hosts in kwranking using the API request /v1/hosts/set/. These hosts must be accessible using SSH with public key authenfication.

4.2 Configuration file

The configuration file is located in /etc/kwranking/kwranking.conf.

Option	Default value	Note
api_port	5001	API port
acl_enabled	true	Keystone authentication
policy_file	/etc/kwstandby/policy.json	Access rules
log_file	/var/log/kwranking.log	Log file
refresh_interval	5184000	Interval between two requests to Ceilometer to retrieve the max value
sql_type	mysql	SQL type
sql_server	localhost	SQL server
sql_port	3306	SQL port
sql_user	root	SQL user
sql_password	password	SQL password
sql_database	kwranking	SQL database

4.3 API

Verb	URL	Parameters	Expected result
GET	/v1/hosts/get-id/		Returns all hosts IDs
GET	/v1/hosts/get/		Returns all hosts details
GET	/v1/hosts/get/<host>/	host	Returns the host details

POST	/v1/hosts/get-rank/	{
		"hosts": a,
		"method": b,
		"number": c
		}
		
		a = host list separated by ";"
		b = ranking method (Efficiency or Flop or Wmax or Wmin)
		c = number of hosts to return
			Ranks the hosts passed as parameter

PUT	/v1/hosts/set/	{
		"host": a
		}
		
		a = the host to benchmark and add in the database
			Returns the probe meter value

4.4 Documentation

Documentation can be found at:

http://kwranking.readthedocs.org

4.5 Code repositories

https://github.com/frossigneux/kwranking

https://github.com/frossigneux/python-kwrankingclient

5 Kwstandby

Kwstandby provides a REST API to shutdown and wakeup the hosts using their IPMI cards.

5.1 Installation

Clone the Kwstandby git repository to the management server:

```
git clone https://github.com/frossigneux/kwstandby
```

As a user with `root` permissions or `sudo` privileges, run the Kwranking installer and copy the configuration files:

```
$ pip install kwstandby
$ cp -r kwstandby/etc/kwstandby /etc/
```

Start the Kwranking API:

```
$ kwstandby-api
```

5.2 Configuration file

The configuration file is located in /etc/kwstandby/kwstandby.conf.

Option	Default value	Note
api_port	5002	API port
acl_enabled	true	Keystone authentication
policy_file	/etc/kwstandby/policy.json	Access rules
log_file	/var/log/kwstandby.log	Log file
ipmi_node	{'interface':'lanplus', 'host':'192.168.0.2', 'username':'user1', 'password':'secret1'}	IPMI card information Multiple lines are allowed

Keystone authtoken section :

Option	Default value	Note
auth_node	localhost	Auth node
auth_protocol	http	Protocol
admin_user	kwstandby	User
admin_password	password	Password
admin_tenant_name	service	Tenant

5.3 API

Verb	URL	Parameters	Expected result
GET	/v1/status/<host>	host	Returns the host status
GET	/v1/status/		Returns all the host status

PUT	/v1/status/<host>	host
		{
		"status": “standby” or “wakeup”
		}
			Update the node status

5.4 Documentation

Documentation can be found at:

http://kwstandby.readthedocs.org

5.5 Code repositories

https://github.com/frossigneux/kwstandby

https://github.com/frossigneux/python-kwstandbyclient