Mục lục

[CHƯƠNG 1: TỔNG QUAN 2](#_Toc9062641)

[1.1 Bài toán giám sát hệ thống 2](#_Toc9062642)

[1.2. Các sản phẩm phục vụ giám sát hệ thống 2](#_Toc9062643)

[1.3. Mục tiêu 2](#_Toc9062644)

[Chức năng 2](#_Toc9062645)

[Hiệu năng 3](#_Toc9062646)

[1.4. Nội dung đồ án 3](#_Toc9062647)

[CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ 4](#_Toc9062648)

[2.1. Ceilometer 4](#_Toc9062649)

[2.2. Gnocchi 14](#_Toc9062650)

[2.3. Aodh 16](#_Toc9062651)

[2.4. Grafana 17](#_Toc9062652)

[2.5. TICK stack 19](#_Toc9062653)

[2.6. Vitrage 20](#_Toc9062654)

[2.7. Heat 23](#_Toc9062655)

[2.8. Mistral 24](#_Toc9062656)

[CHƯƠNG 3: PHÁT TRIỂN HỆ THỐNG GIÁM SÁT 26](#_Toc9062657)

[3.1. Phân tích hệ thống 26](#_Toc9062658)

[3.1. Các đối tượng hệ thống 26](#_Toc9062659)

[3.2. Mô tả uscase 26](#_Toc9062660)

[3.1. Thiết kế hệ thống 30](#_Toc9062661)

[3.1.1.Bố trí các thành phần monitor. 30](#_Toc9062662)

[3.1.2 Mô tả luồng hoạt động 30](#_Toc9062663)

[3.3. Xây dựng hệ thống. 36](#_Toc9062664)

[3.3.1. Các yêu cầu cho hệ thống sẵn có 36](#_Toc9062665)

[3.3.2. Chi tiết về việc xây dựng hệ thống. 37](#_Toc9062666)

[CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ 38](#_Toc9062667)

[4.1.Tự động evacuate instance khi host down 38](#_Toc9062668)

[4.2.Tự động live migrate instance từ host cao tải sang host ít tải hơn. 43](#_Toc9062669)

[4.3.Tự động scale group-instance khi có cảnh báo về tài nguyên 43](#_Toc9062670)

[CHƯƠNG 5: KẾT LUẬN 46](#_Toc9062671)

[TÀI LIỆU THAM KHẢO 46](#_Toc9062672)

# CHƯƠNG 1: TỔNG QUAN

## 1.1 Bài toán giám sát hệ thống

Xây dựng hệ thống từ lâu đã là một bài toán cơ bản và quan trọng nhất của người quản trị hệ thống ( system-administrator) cần giải quyết. Một hệ thống giám sát giúp ta giải quyết các vấn đề

* Cho biết được trạng thái hoạt động của hệ thống gồm tài nguyên (ram, cpu, network, ổ cứng) sử dụng và các dịch vụ cần quan tâm. Giúp điều chỉnh lượng tài nguyên sử dụng cho phù hợp với nhu cầu tại từng thời điểm
* Cảnh báo những sự cố có thể xảy ra, giúp ích được cho quá trình troubleshot lỗi . Tiến tới có thể giúp người quản trị có những tác động vào hệ thống khi gặp phải các sự cố đó.

Một hệ thống giám sát tốt sẽ giúp quản lý hiệu quả tài nguyên và gia tăng chất lượng dịch vụ cung cấp cho khách hàng.

Trong môi trường cloud, bài toán này lại trở nên rất đặc thù: người sử dụng hệ thống cloud bản thân cũng là một người quản trị. Hệ thống giám sát phải có thể hỗ trợ được cả 2 góc nhìn: từ phía nhà cung cấp cloud, và phía người sử dụng hệ thống cloud. Phía nhà cung cấp cloud quan tâm chủ yếu là các tài nguyên vật lý và các tiến trình chạy trên đó . Phía người sử dụng hệ thống cloud cần quan tâm tới tài nguyên ảo hóa mà mình được nhận.

## 1.2. Các sản phẩm phục vụ giám sát hệ thống

Hiện nay các sản phẩm sử dụng giám sát hệ thống đã rất phong phú, các sản phẩm opensource phải kể đến như: Zabbix, Nagios, Graphite, Prometheus, TICK,… Giữa các sản phẩm tuy có những khác biệt về kiến trúc, mô hình dữ liệu,… nhưng các usecase cho việc monitor thì cũng đều đầy đủ và không nhiều khác biệt nên không sản phẩm nào thực sự ưu việt hơn tất cả. Em sẽ lựa chọn TICK-stack làm giải pháp monitor cho các thành phần vật lý, ý kiến chủ quan là do.

* Zabbix, nagios sử dụng thành phần lưu trữ là SQL database. Dữ liệu monitor của server -các datapoint tại từng thời điểm có đặc tính là kích thước1 bản ghi rất nhỏ, đổ về liên tục từng milliseconds. Về lâu dài sẽ làm quá tải trên database
* Graphite, Prometheus, TICK: sử dụng time series database- loại database thiết kế riêng cho việc monitor. TICK bản thân em thấy sử dụng dễ dàng và có nhiều hỗ trợ từ cộng đồng.

Quản lý các tài nguyên ảo hóa em sẽ sử dụng là Telemetry, bộ công cụ dùng để monitor cho hệ thống cloud của openstack. Bao gồm các service: ceilometer, gnocchi, aodh. Em cũng kết hợp các service khác từ openstack như heat, vitrage, mistral cho việc tự động phản ứng với những sự cố.

## 1.3. Mục tiêu

Đề tài **“Xây dựng hệ thống giám sát trong triển khai openstack cloud”** của em sẽ giải quyết các bài toán monitor về:

### Chức năng

Xây dựng hệ thống đáp ứng được những yêu cầu về chức năng cho các user:

**Cloud provider:**

* Thống kê số lượng node và trạng thái (status/healthy) của các host vật lý
* Monitor tài nguyên: CPU, Memory, Disk, Network bandwith/throughput của host
* Monitor service openstack và các service cần thiết cho hoạt động openstack, như mysql, rabbitmq,...
* Thiết lập cảnh báo khi có tài nguyên bất thường: vd host mất kết nối, tài nguyên host vượt ngưỡng
* Hỗ trợ phân tích nguyên nhân của alam, tự động hóa giải quyết được những lỗi cơ bản. (auto healing)

*Facebook :hệ thống “auto remediation” được facebook thông báo là chỉ cần 2 người chịu trách nhiệm vận hành 1 datacenter mà bình thường dùng đến 200 người. Tất cả các lỗi đều được giải quyết một cách tự động. [1]*

**Cloud user:**

* Monitor tài nguyên: CPU, Memory, Disk, Network bandwith/throughput của instance
* Thiết lập cảnh báo và có chính sách áp dụng. Ví dụ: tăng số lượng instance khi các máy cao tải (auto scaling)
* Cloud provider không có quyền cài bất kỳ thành phần monitor nào trực tiếp trên máy ảo người dùng.

*Mô hình monitor Cloudwatch của Amazone đã thực hiện được hoàn chỉnh các uscase này.*

### Hiệu năng

Tìm hiểu yêu cầu hiệu năng của hệ thống monitor:

Đề xuất RAM, CPU cần dùng cho monitor, dung lượng disk để lưu dữ liệu giám sát. Mô hình để đảm bảo high availability.

## 1.4. Nội dung đồ án

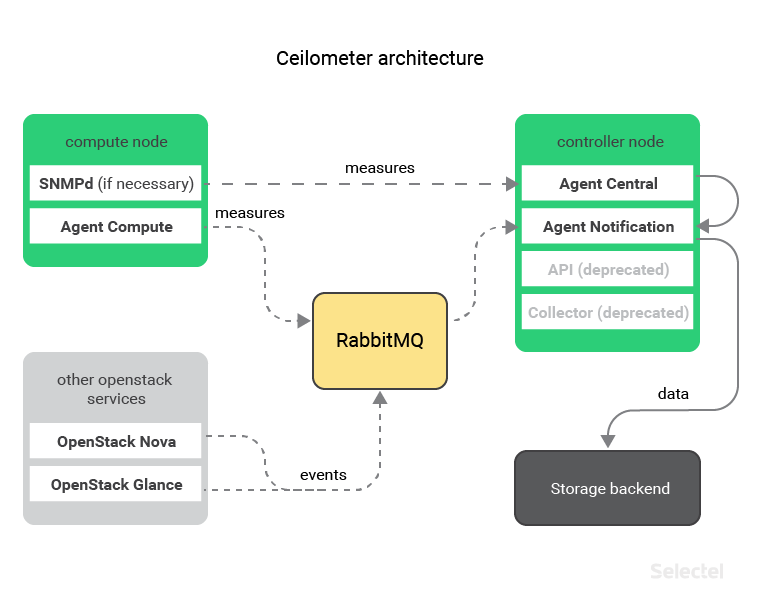
Đồ án có các phần:

* Chương 2 nền tảng công nghệ: Giới thiệu những công nghệ được sử dụng: chức năng, kiến trúc, vai trò của tích hợp công nghệ này trong hệ thống
* Chương 3 phát triển hệ thống giám sát:
  + Phân tích các đối tượng, nhu cầu, ca sử dụng để thiết kế các thành phần hệ thống
  + Mô tả cài đặt chi tiết cho các thành phần công nghệ
* Chương 4 Thử nghiệm: các kịch bản người dùng có thể sử dụng với hệ thống
* Chương 5 Kết luận và đánh giá

# CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ

## 2.1. Ceilometer

Ceilometer là một dịch vụ của openstack. Sử dụng thu thập metric của các tài nguyên ảo hệ thống, nhờ vào khả năng vào thông tin lấy từ các service khác openstack cộng với việc truy vấn libvirt api của thành phần ảo hóa.



Ceilometer bao gồm hai loại agent:

* **Notification agent:** lấy message từ notification bus (rabbitmq) và chuyển đổi thành ceilometer samples hoặc events. Agent thuộc loại này: **ceilometer-agent-notification**.
* **Polling agent:** thực hiện poll OpenStack APIs hoặc một số công cụ khác (ipmid, snmpq) để thu thập thông tin theo chu kỳ đặt trước. Các agents thuộc loại này: **ceilometer-agent-compute, ceilometer-agent-ipmi, ceilometer-agent-central.**

Chi tiết các agent được trình bày sau đây.

* **Ceilometer-agent-compute:** chạy trên các Compute Node thu thập metrics của các VMs bằng cách poll libvirt-api, dữ liệu này chuyển tới ceilometer-agent-notification qua message queues.
* Danh sách chi tiết các metrics thu thập bởi ceilometer-agent-compute:

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| memory | Gauge | MB | instance ID | Notification | Volume of RAM allocated to the instance |
| memory.usage | Gauge | MB | instance ID | Pollster | Volume of RAM used by the instance from the amount of its allocated memory |
| memory.resident | Gauge | MB | instance ID | Pollster | Volume of RAM used by the instance on the physical machine |
| cpu | Cumulative | ns | instance ID | Pollster | CPU time used |
| cpu.delta | Delta | ns | instance ID | Pollster | CPU time used since previous datapoint |
| cpu\_util | Gauge | % | instance ID | Pollster | Average CPU utilization |
| vcpus | Gauge | vcpu | instance ID | Notification | Number of virtual CPUs allocated to the instance |
| disk.read.requests | Cumulative | request | instance ID | Pollster | Number of read requests |
| disk.read.requests.rate | Gauge | request/s | instance ID | Pollster | Average rate of read requests |
| disk.write.requests | Cumulative | request | instance ID | Pollster | Number of write requests |
| disk.write.requests.rate | Gauge | request/s | instance ID | Pollster | Average rate of write requests |
| disk.read.bytes | Cumulative | B | instance ID | Pollster | Volume of reads |
| disk.read.bytes.rate | Gauge | B/s | instance ID | Pollster | Average rate of reads |
| disk.write.bytes | Cumulative | B | instance ID | Pollster | Volume of writes |
| disk.write.bytes.rate | Gauge | B/s | instance ID | Pollster | Average rate of writes |
| disk.device.read.requests | Cumulative | request | disk ID | Pollster | Number of read requests |
| disk.device.read.requests.rate | Gauge | request/s | disk ID | Pollster | Average rate of read requests |
| disk.device.write.requests | Cumulative | request | disk ID | Pollster | Number of write requests |
| disk.device.write.requests.rate | Gauge | request/s | disk ID | Pollster | Average rate of write requests |
| disk.device.read.bytes | Cumulative | B | disk ID | Pollster | Volume of reads |
| disk.device.read.bytes .rate | Gauge | B/s | disk ID | Pollster | Average rate of reads |
| disk.device.write.bytes | Cumulative | B | disk ID | Pollster | Volume of writes |
| disk.device.write.bytes .rate | Gauge | B/s | disk ID | Pollster | Average rate of writes |
| disk.root.size | Gauge | GB | instance ID | Notification | Size of root disk |
| disk.ephemeral.size | Gauge | GB | instance ID | Notification | Size of ephemeral disk |
| disk.latency | Gauge | ms | instance ID | Pollster | Average disk latency |
| disk.iops | Gauge | count/s | instance ID | Pollster | Average disk iops |
| disk.device.latency | Gauge | ms | disk ID | Pollster | Average disk latency per device |
| disk.device.iops | Gauge | count/s | disk ID | Pollster | Average disk iops per device |
| disk.capacity | Gauge | B | instance ID | Pollster | The amount of disk that the instance can see |
| disk.allocation | Gauge | B | instance ID | Pollster | The amount of disk occupied by the instance on the host machine |
| disk.usage | Gauge | B | instance ID | Pollster | The physical size in bytes of the image container on the host |
| disk.device.capacity | Gauge | B | disk ID | Pollster | The amount of disk per device that the instance can see |
| disk.device.allocation | Gauge | B | disk ID | Pollster | The amount of disk per device occupied by the instance on the host machine |
| disk.device.usage | Gauge | B | disk ID | Pollster | The physical size in bytes of the image container on the host per device |
| network.incoming.bytes | Cumulative | B | interface ID | Pollster | Number of incoming bytes |
| network.incoming.bytes.rate | Gauge | B/s | interface ID | Pollster | Average rate of incoming bytes |
| network.outgoing.bytes | Cumulative | B | interface ID | Pollster | Number of outgoing bytes |
| network.outgoing.bytes.rate | Gauge | B/s | interface ID | Pollster | Average rate of outgoing bytes |
| network.incoming.packets | Cumulative | packet | interface ID | Pollster | Number of incoming packets |
| network.incoming.packets.rate | Gauge | packet/s | interface ID | Pollster | Average rate of incoming packets |
| network.outgoing.packets | Cumulative | packet | interface ID | Pollster | Number of outgoing packets |
| network.outgoing.packets.rate | Gauge | packet/s | interface ID | Pollster | Average rate of outgoing packets |
| cpu\_l3\_cache | Gauge | B | instance ID | Pollster | L3 cache used by the instance |
| memory.bandwidth.total | Gauge | B/s | instance ID | Pollster | Total system bandwidth from one level of cache |
| memory.bandwidth.local | Gauge | B/s | instance ID | Pollster | Bandwidth of memory traffic for a memory controller |
| perf.cpu.cycles | Gauge | cycle | instance ID | Pollster | the number of cpu cycles one instruction needs |
| perf.instructions | Gauge | instruction | instance ID | Pollster | the count of instructions |
| perf.cache.references | Gauge | count | instance ID | Pollster | the count of cache hits |
| perf.cache.misses | Gauge | count | instance ID | Pollster | the count of cache misses |

* Ví dụ: truy vấn thông tin một resource thuộc loại **“instance”**.

|  |
| --- |
| root@controller02:/# openstack metric resource show f6cbfb95-ff58-4692-a1be-d8e2aa1f9b4c  +-----------------------+------------------------------------------------------------------  | Field | Value |  +-----------------------+------------------------------------------------------------------  | created\_by\_project\_id | 2498895c3f484bf89eed68a1f4b6cfbc |  | created\_by\_user\_id | e8b821cce5104ee494a304b1ba39d50e |  | creator | e8b821cce5104ee494a304b1ba39d50e:2498895c3f484bf89eed68a1f4b6cfbc |  | ended\_at | None |  | id | f6cbfb95-ff58-4692-a1be-d8e2aa1f9b4c |  | metrics | compute.instance.booting.time: a4599e5d-11e5-4172-a835-b1d545aa75ab |  | | cpu.delta: 8ef57d26-93da-47e5-87fa-55de3eb72485 |  | | cpu: 094ed0c0-f500-4a69-bfed-33571684c641 |  | | cpu\_l3\_cache: 1741fcef-10d0-453e-9eb1-a5b63318675b |  | | cpu\_util: 96e17829-adaa-4b8d-bbff-e1ee5f242e92 |  | | disk.allocation: e6677d6d-dd0e-40d6-8170-af1c75285af9 |  | | disk.capacity: 96bd85bb-19ad-4b6e-9e15-8c9f495c9a72 |  | | disk.ephemeral.size: 2e0cfb9f-993e-4da0-8583-06fc473ad3ab |  | | disk.iops: 6400221e-a0b4-495e-bd62-7e4b13d3b2a5 |  | | disk.latency: f5c02d4b-b9df-4502-b5dd-1022e4e61127 |  | | disk.read.bytes.rate: d49ce524-895b-414c-9c16-5f7d998d2b5e |  | | disk.read.bytes: 7fa5684e-de4d-473a-8627-949de365997e |  | | disk.read.requests.rate: 1b682961-48c8-4da9-893d-445db761d904 |  | | disk.read.requests: 2631334e-3e42-4d01-a0df-1351d41f7f8a |  | | disk.root.size: 000ccdf1-88e6-4860-8c8b-cbbe7be5c224 |  | | disk.usage: 08e30750-fb29-48a6-b1f4-d759335fd132 |  | | disk.write.bytes.rate: 2c2cd070-55f4-44b5-a215-b5bb7e442657 |  | | disk.write.bytes: b57f66cc-b065-41a2-ab17-6875664dc944 |  | | disk.write.requests.rate: 302bbcc8-eca6-4392-a852-34dcfff79dca |  | | disk.write.requests: 845ce60d-16a5-4641-9e17-96873e688d2b |  | | memory.bandwidth.local: febe7c43-811a-441c-aba9-1bfc76fd4cf7 |  | | memory.bandwidth.total: 5656344e-4fe2-48a9-beec-57e0d497dcd4 |  | | memory.resident: bc642726-2cbd-439f-8791-2c698b76fff7 |  | | memory.swap.in: 8ce2f9a2-aff4-4c43-8af6-6b36a726ab22 |  | | memory.swap.out: 1a7bad2c-90da-4e1c-9763-1ee44f73cbb7 |  | | memory.usage: e0903342-464d-4522-9685-4056929d7bcc |  | | memory: 0ae8fbc4-c24c-479d-9c3b-d24556e6923c |  | | perf.cache.misses: fd619c08-0202-43bc-ae2d-a3608dacf777 |  | | perf.cache.references: 04ffa509-0019-4cc7-9934-791133bb9abf |  | | perf.cpu.cycles: 69a04a8b-f94e-4b87-910f-e07ccf01705d |  | | perf.instructions: 051adefa-27d1-4fdf-bcd5-dd2fbd92020e |  | | vcpus: 73c4c88a-ceb3-4343-9623-dce624a3c109 |  | original\_resource\_id | f6cbfb95-ff58-4692-a1be-d8e2aa1f9b4c |  | project\_id | c7bc7ddd815947fba9a0f877b17c07de |  | revision\_end | None |  | revision\_start | 2018-07-31T08:29:53.398970+00:00 |  | started\_at | 2018-07-31T08:29:53.398945+00:00 |  | type | instance |  | user\_id | 940678edf03c4ea59626d6bf9d89c21c |  +-----------------------+-------------------------------------------------------- |

* **Ceilometer-agent-central:**
* ***Poll snmp daemon để thu thập hardware resources metrics (cpu load, cpu util, memory usage, memory swap, network throughput,….). Để thu thập metric này các host cần mở giao thức snmp, kém an toàn trong môi trường thực tế nên không sử dụng***
* ***Poll public REST APIs của các OpenStack services để thu thập metrics của các resources do các OpenStack services này quản lý.***

Các OpenStack services được hỗ trợ:

* **OpenStack Object Storage**

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| storage.objects | Gauge | object | storage ID | Pollster | Number of objects |
| storage.objects.size | Gauge | B | storage ID | Pollster | Total size of stored objects |
| storage.objects.containers | Gauge | container | storage ID | Pollster | Number of containers |
| storage.containers.objects | Gauge | object | storage ID/container | Pollster | Number of objects in container |

* **Ceilometer-agent-notification:**

Thu thập dữ liệu qua message bus. Trong đó bao gồm:

* ***Các events về các sự kiện, các thao tác xử lý của các OpenStack services khác***
* ***Các metrics sample từ ceilometer-agent-compute & ceilometer-agent-central.***

Ceilometer-agent-notification thực hiện xử lý các events, samples nhận được và đẩy ra các publisher tương ứng. Trong đó:

* Các events được oslo publisher đẩy sang khối lưu trữ và xử lý event
* Các metrics được gnocchi publisher đẩy sang khối metrics storage (Gnocchi).

Các notification thu thập bởi ceilometer-agent-notification:

* ***OpenStack Networking***

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| bandwidth | Delta | B | label ID | Notification | Bytes through this l3 metering label |
| **Meters removed as of Ocata release** | | | | | |
| network | Gauge | network | network ID | Notification | Existence of network |
| network.create | Delta | network | network ID | Notification | Creation requests for this network |
| network.update | Delta | network | network ID | Notification | Update requests for this network |
| subnet | Gauge | subnet | subnet ID | Notification | Existence of subnet |
| subnet.create | Delta | subnet | subnet ID | Notification | Creation requests for this subnet |
| subnet.update | Delta | subnet | subnet ID | Notification | Update requests for this subnet |
| port | Gauge | port | port ID | Notification | Existence of port |
| port.create | Delta | port | port ID | Notification | Creation requests for this port |
| port.update | Delta | port | port ID | Notification | Update requests for this port |
| router | Gauge | router | router ID | Notification | Existence of router |
| router.create | Delta | router | router ID | Notification | Creation requests for this router |
| router.update | Delta | router | router ID | Notification | Update requests for this router |
| ip.floating | Gauge | ip | ip ID | Notification | Existence of IP |
| ip.floating.create | Delta | ip | ip ID | Notification | Creation requests for this IP |
| ip.floating.update | Delta | ip | ip ID | Notification | Update requests for this IP |

* ***OpenStack Object Storage***

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| storage.objects.incoming.bytes | Delta | B | storage ID | Notification | Number of incoming bytes |
| storage.objects.outgoing.bytes | Delta | B | storage ID | Notification | Number of outgoing bytes |
| storage.api.request | Delta | request | storage ID | Notification | Number of API requests against OpenStack Object Storage |

* ***OpenStack Block Storage***

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| volume.size | Gauge | GB | volume ID | Notification | Size of the volume |
| snapshot.size | Gauge | GB | snapshot ID | Notification | Size of the snapshot |
| **Meters removed as of Ocata release** | | | | | |
| volume | Gauge | volume | volume ID | Notification | Existence of the volume |
| snapshot | Gauge | snapshot | snapshot ID | Notification | Existence of the snapshot |
| volume.create.(start|end) | Delta | volume | volume ID | Notification | Creation of the volume |
| volume.delete.(start|end) | Delta | volume | volume ID | Notification | Deletion of the volume |
| volume.update.(start|end) | Delta | volume | volume ID | Notification | Update the name or description of the volume |
| volume.resize.(start|end) | Delta | volume | volume ID | Notification | Update the size of the volume |
| volume.attach.(start|end) | Delta | volume | volume ID | Notification | Attaching the volume to an instance |
| volume.detach.(start|end) | Delta | volume | volume ID | Notification | Detaching the volume from an instance |
| snapshot.create.(start|end) | Delta | snapshot | snapshot ID | Notification | Creation of the snapshot |
| snapshot.delete.(start|end) | Delta | snapshot | snapshot ID | Notification | Deletion of the snapshot |
| volume.backup.create.(start|end) | Delta | volume | backup ID | Notification | Creation of the volume backup |
| volume.backup.delete.(start|end) | Delta | volume | backup ID | Notification | Deletion of the volume backup |
| volume.backup.restore.(start|end) | Delta | volume | backup ID | Notification | Restoration of the volume backup |

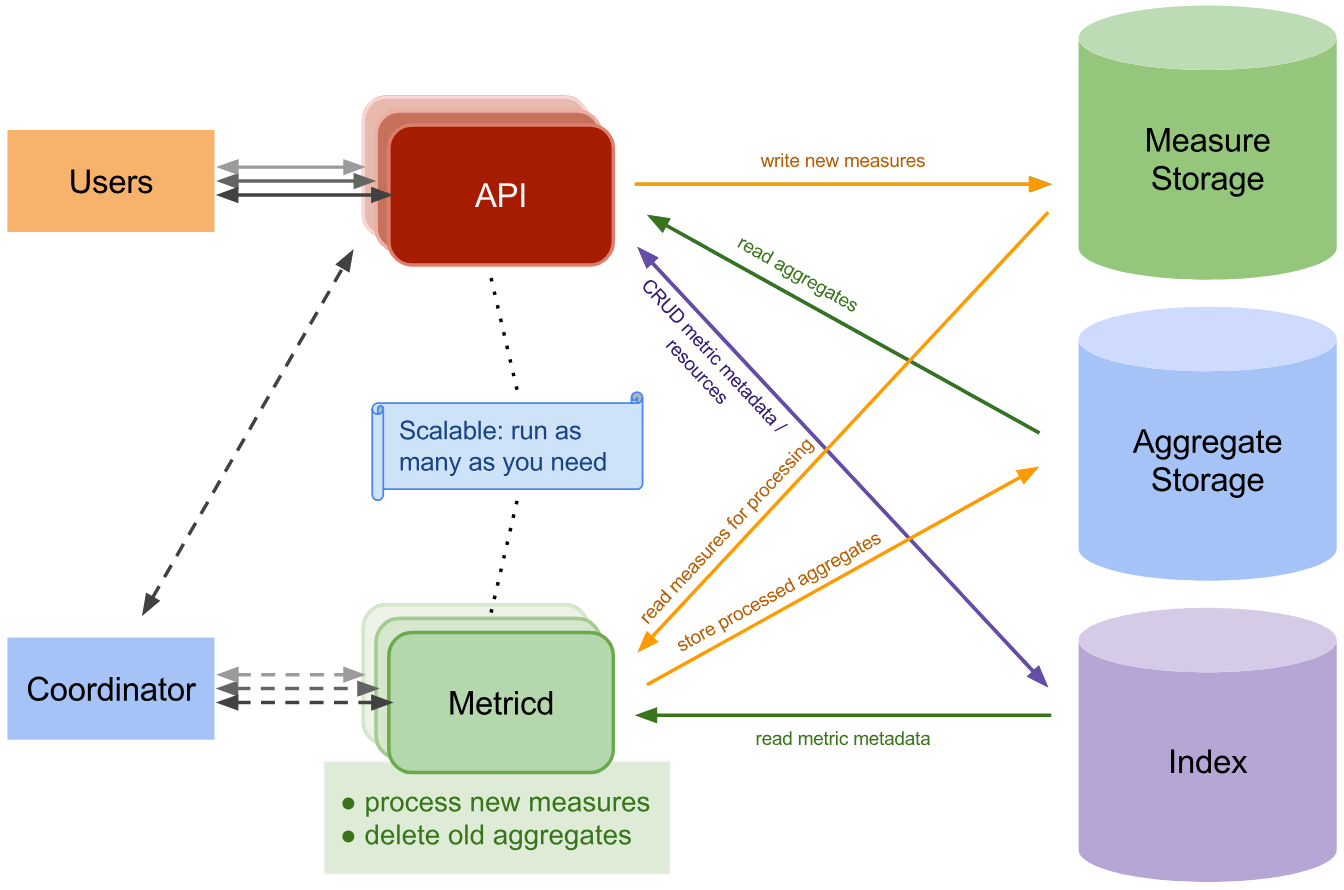
* ***OpenStack Images***

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| image.size | Gauge | image | image ID | Notification | Size of the uploaded image |
| image.update | Delta | image | image ID | Notification | Number of updates on the image |
| image.upload | Delta | image | image ID | Notification | Number of uploads on the image |
| image.delete | Delta | image | image ID | Notification | Number of deletes on the image |
| image.download | Delta | B | image ID | Notification | Image is downloaded |
| image.serve | Delta | B | image ID | Notification | Image is served out |
| **Meters removed as of Ocata release** | | | | | |
| image | Gauge | image | image ID | Notification | Existence of the image |

## 2.2. Gnocchi

Gnocchi được dùng để lưu trữ metric thu thập từ ceilometter , tức lưu trữ metric của tài nguyên ảo hóa. Đây là opensoucre opensoucre time series database, xuất phát cũng là một project của openstack sau tách ra.

Time series database là kiểu database được tối ưu hóa để xử lý dữ liệu từ các phép đo liên tiếp realtime, có chỉ mục được lập theo thời gian.



Gnocchi gồm hai khối chính:

* **Control plane:**
* **gnocchi-api:** tiếp nhận yêu cầu xử lý metrics
* **gnocchi-metricd:** daemon thực hiện các tác vụ tính toán, thống kê,… các metrics và lưu vào backend storage
* **Data plane:**

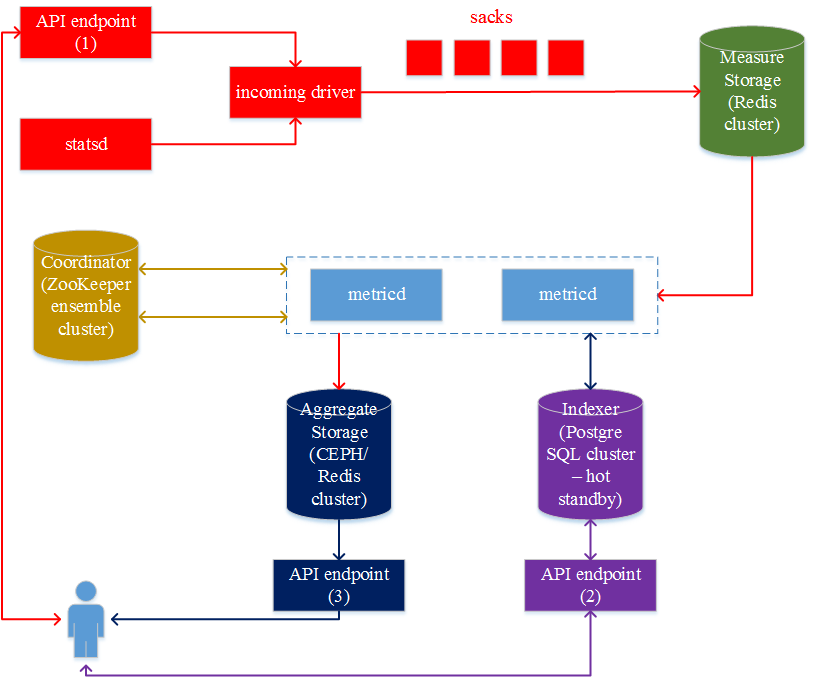
Các backend storage, gnocchi dùng 3 thành phần backend:

* Indexer:  dùng lưu trữ index và metadata : định nghĩa, loại, thuộc tính của các resources, archive policies. Tùy chọn backend: MySQL, PostgreSQL.
* Incoming Storage: lưu trữ các mesures mới gửi tới (một measures là một record bao gồm timestamp và giá trị tương ứng của một metric tại thời điểm nào đó, đây là loại dữ liệu raw chưa xử lý). Tùy chọn backend: Redis, CEPH, filesystem, S3, Swift.
* Aggregate Storage: lưu trữ các aggregate của một metrics. Tức các giá trị có ý nghĩa, sau tính toán, dữ liệu raw theo luật archive policies xác định. Archive policies này định nghĩa việc sử dụng các phép toán: mean, min, max, count,…, Tùy chọn backend: Redis, CEPH, filesystem, S3, Swift.

Một số thành phần khác:

* Thao tác đọc ghi này được phân bố giữa các metricd workers nhờ coordination backend
* Tùy chọn coordination: **redis, zookeeper.**
* Để đảm bảo tính nhất quán giữa các gnocchi-api và gnocchi-metricd workers khi cùng đọc và ghi vào incoming backend, Gnocchi cung cấp carbonara driver hỗ trợ việc khóa timeseries đang được update trên toàn bộ các metricd workers.

**Luồng hoạt động**



Luồng lưu trữ measures:

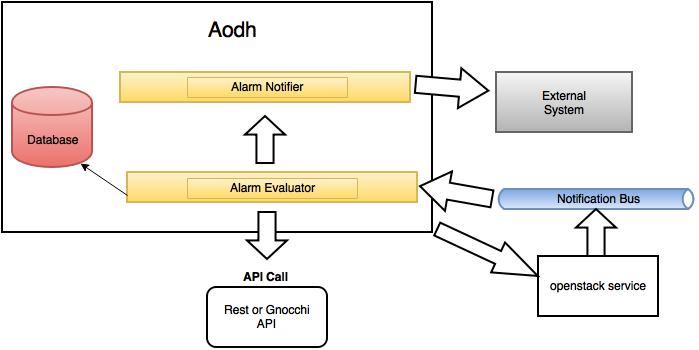
* Các metrics samples (các giá trị measures raw mới thu thập được của các metrics) từ ceilometer gửi tới được xử lý bởi một gnocchi API endpoint hoặc statsd daemon.
* Các entries thu thập được gửi tới incoming driver, tại đây các entries này sẽ được chia nhỏ thành từng phần gọi là sacks rồi lưu vào Incoming Storage (Measure Storage)
* Các metrics và measures mới trong Incoming Storage sẽ được truy vấn bởi metric-processor của gnocchi-metricd với delay là metric\_processing\_delay (config trong gnocchi.conf). Việc truy vấn này nhằm hai mục đích:
* Lưu metadata của metrics, resources và archive policies vào Indexer.
* Tính toán giá trị aggregate của các metrics dựa theo các archive policies đã định nghĩa và lưu trữ vào Aggregate Storage. Workload tính toán aggregate cho các metrics được phân phối và đồng bộ bởi coordinator (ZooKeeper).

Để tính toán aggregate cho các metric, metric processor sẽ đọc từ Indexer để lấy các active metrics ra xử lý.

Với các metrics đã inactive , như khi resoucre bị xóa, gnocchi-metricd sẽ tương tác với Indexer và Aggegate Storage để xóa các inactive metrics và các measures của các metrics này.

## 2.3. Aodh

Aodh là alarming service của openstack, kết nối với gnocchi để cảnh báo khi dữ liệu vượt quá hoặc thấp hơn ngưỡng định trước.



Kiến trúc aodh gồm 4 thành phần:

* **aodh-api:** Cung cấp api truy cập các alarms cho user. Thông tin các alarm lưu trữ trong MySQL database. **aodh-api** chạy dưới HAproxy với mod\_wsgi trên các controller node
* **aodh-evaluator:** Tính toán và đánh giá alarm dựa trên các metric samples match với alarm đó. **aodh-evaluator** truy vấn Gnocchi api để lấy các metrics samples. **aodh-evaluator** có thể chạy trên nhiều node controller, nhưng phải cấu hình coordinator (ZooKeeper) để đồng bộ, tránh xung đột.
* **aodh-listener**: Tính toán và đánh giá alarm dựa trên các events match với alarm đó. **aodh-listener** sử dụng oslo.messaging listeners nhận các messages (chứa events ) từ oslo.messaging publishers của ceilometer-agent-notifications. **aodh-listener** có thể chạy trên một hoặc nhiều node controller mà không cần cấu hình coordinator.
* **aodh-notifier:** Thực hiện các action định nghĩa trong các alarms khi có sự thay đổi trạng thái của mỗi alarm (ok, alarm, insufficient data**). aodh-notifier** có thể chạy trên một hoặc nhiều node controller mà không cần cấu hình coordinator.

Kênh gửi alarm của aodh-notifier:

* log file (của /var/log/aodh-notifier.log). tuy nhiên chỉ user quyền admin mới có thể sử dụng kênh này
* webhook (Telegram, Slack, etc. webhook). Thường thấy nhất là sẽ gửi tin nhắn về cho người dùng telegram
* message bus .Mục đích cho các thành phần khác đọc và xử lý . vd: virtrage, zaquar

## 2.4. Grafana

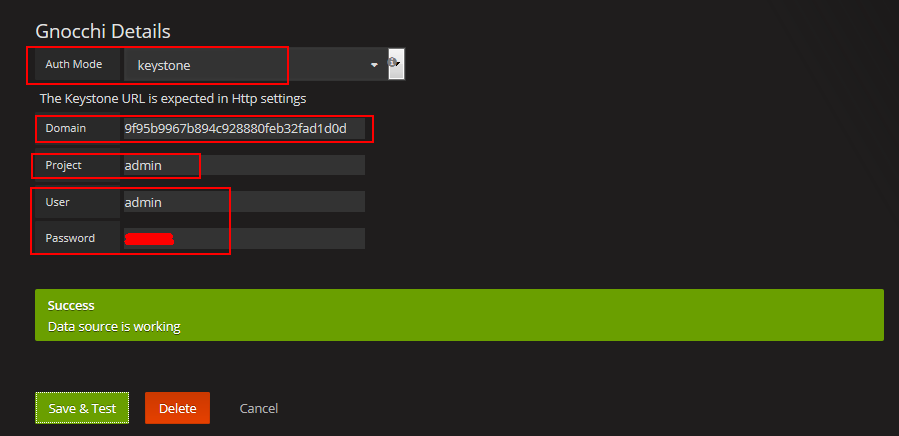
Grafana là một opensource rất nổi tiếng cho việc tạo drashboard và trình diễn dữ liệu thành các bảng biểu.

Các đặc điểm, tính năng chính:

* Grafana là phần mềm mã nguồn mở được phát triển bởi Grafana Labs.
* Grafana cung cấp giao diện ứng dụng web viết bằng ngôn ngữ lập trình JavaScript và Golang có thể tương tác mạnh mẽ với các backend dữ liệu như mysql, InfluxDB, graphite và gnocchi.
* Grafana cho phép hiển thị thông tin hữu ích một cách rõ ràng, súc tích dưới
* dạng đồ thị, bảng,…
* Grafana có thư viện darshboard mạnh mẽ với nhiều nguồn dữ liệu khác nhau.
* Đồng thời Grafana cũng câp cấp khả năng tính toán và cảnh báo đến người quản trị thống qua các dịch vụ sms, mail,….

Vai trò của Grafana trong mô hình:

* Grafana request vào gnocchi api để truy xuất ra những thông tin cần thiết thể hiện trên bảng điều khiển (Dashboards) dưới dạng biểu đồ , bảng số liệu…. trên giao diện web browser, cung cấp cho người dùng cái nhìn trực quan về các số liệu.
* Grafana hỗ trợ xác thực keystone để đảm bảo tính bảo mật khi request vào gnocchi.

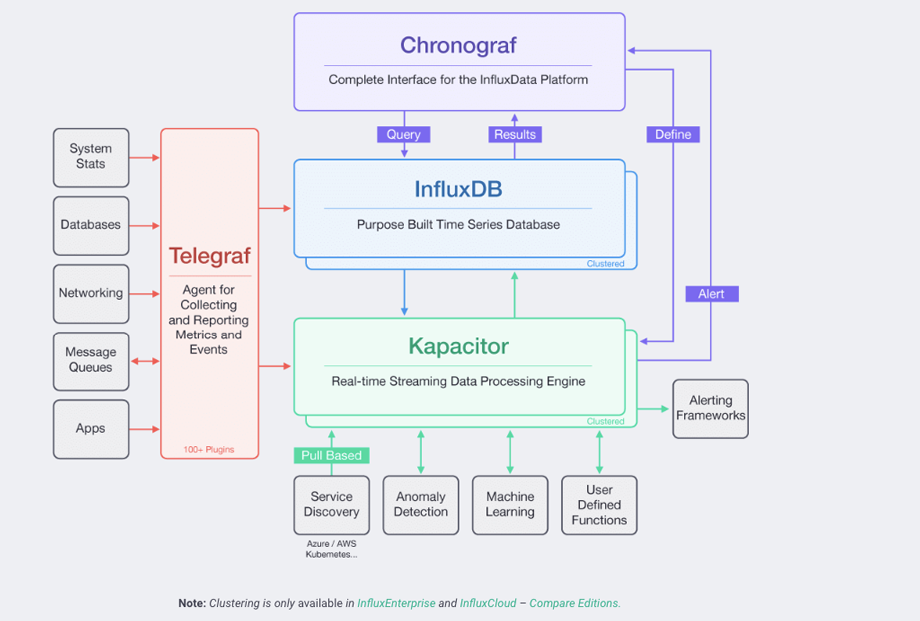


Người dùng sử dụng openstack account kết nối đến gnocchi



Các thông số của tài nguyên instance người dùng

## 2.5. TICK stack



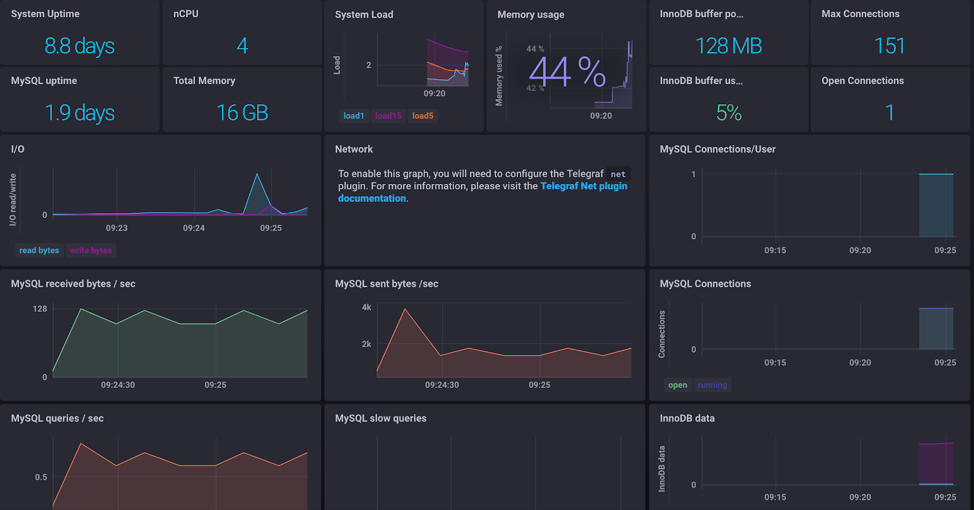
TICK stack is a tập hợp của các sản phầm opensoucre của InfluxData phục vụ monitorinng, là tập hợp của 4 project:

* **T**elegraf : vai trò collector , thu thập time-series data từ nhiều nguồn khác nhau: ví dụ như tài nguyên server, các cuộc gọi API, truy vấn DB, SMNP.
* **I**nfluxDB : vai trò storage, lưu trữ time-series data.
* **C**hronograf : vai trò visualize , hiển thi trực quan các thông số truy vấn từ influxDB thành bảng biểu.
* **K**apacitor: vai trò xử lý time-series data và cảnh báo cho người dùng, qua các kênh như email, telegram messenger

Vai trò của TICK trong mô hình:

TICK để monitor cho các host vật lý, phục vụ nhu cầu của Cloud provider. Từ TICK có thể biết được trạng thái tài nguyên và dịch vụ của các host. Đặc biêt đề đảm bảo hoạt động của openstack ta cần chú ý đến các dịch vụ sau:

* Service thuộc project openstack như nova, cinder, glance, neutron, heat
* Mysql
  + Số lượng kết nối đến, số truy vấn
  + Số byte send in, out
* Rabbit MQ:
* Số lượng file descriptors sử dụng bởi RabbitMQ
* Số lượng file descriptors sử dụng như các network sockets bởi RabbitMQ
* Dung lượng ổ cứng chiếm bởi RabbitMQ trên mỗi OpenStack Node
* Dung lượng RAM sử dụng bởi RabbitMQ trên mỗi OpenStack Node
* Memcached:
* Số bytes đã được dùng để cache
* Số bytes tối đa được cho phép trong cache
* Current open connections
* Số lượng GET/SET requests nhận bởi server mỗi giây
* Hits (số lượng GET requests thành công mỗi giây)
* Misses (số lượng GET requests bị từ chối mỗi giây)
* Read (Số lượng bytes/s gửi từ network và đọc bởi server)
* Written (số lượng bytes/s gửi từ server ra bên ngoài)



Hình ảnh về giao diện monitor host vật lý

## 2.6. Vitrage

Vitrage - OpenStack RCA (Root Cause Analysis) là dịch vụ giúp tổ chức, phân tích, các alarm, event xảy đến với hệ thống, nhằm làm rõ được những alarm này có nguồn gốc từ đâu và có thể gây ảnh hưởng thế nào.

Vai trò của vitrage trong mô hình :

* Đối với cloud - provider (user admin) : với trường hợp hệ thống gặp sự cố, không chỉ một mà rất nhiều alarm cùng đưa ra. Vitrage sẽ hỗ trợ được người quản trị trong troubleshot tìm ra đâu là alarm nguồn. Nó cũng có khả năng gọi workflow đã xác định trước (gọi mistral service) để sửa chữa hệ thống
* Đối với cloud - user (nomal user): user nếu tự cài hệ thống monitor của riêng mình, có thể forwarding các alarm hệ thống đó trả ra về cho vitrage, từ đó có thể vào giao diện horizon được cung cấp xem các alarm này

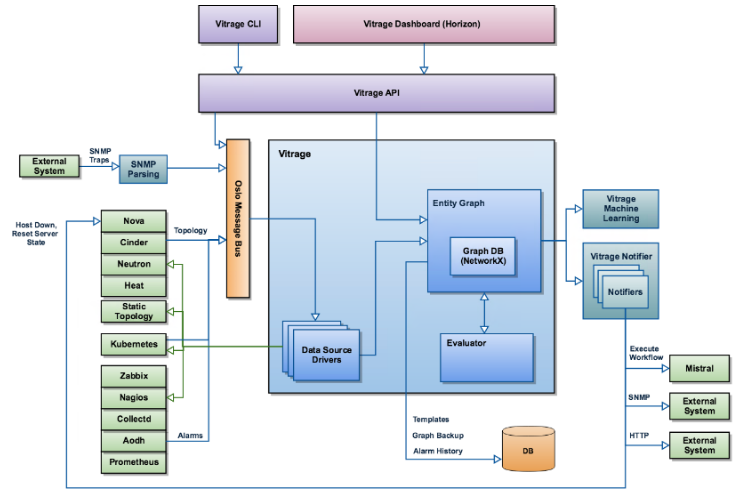
Khả năng của vitrage:

* Holistic and complete view: vitrage có thể biểu diễn mỗi quan hệ logic của các tài nguyên trong hệ thống thành grap+topology cho người xem, bao gồm cả thực thể virtual (instance, port, zone,..) hay physical (host , switch,..) kết nối với nhau. Để khi có một alarm xuất hiện từ một thành phần ta giải thích được nó sẽ ảnh hưởng tới thành phần khác như thế nào.
* Deduced alarm, state: mục đích nhắm tới là đưa ra cảnh báo về một thành phần hệ thống, ngay trước cả khi nó được phát hiện trực tiếp bởi hệ thống monitor.

Ví dụ:

Khi switch tới host gặp vấn đề, khiến ngay các instance nối với host đó cũng không monitor được, vitrage có thể đưa ra cảnh báo với insance.

* Root Cause Indicators: biểu diễn quan hệ nguyên nhân-kết quả giữa các alarm. Vitrage đọc các luật người dùng định nghĩa, xem xem có thể áp dụng luật nào vào trạng thái hiện tại và đưa các liên kết “causual” vào biểu đồ. Giúp cho việc tìm ra nguồn kích hoạt gốc của cho các alarm.

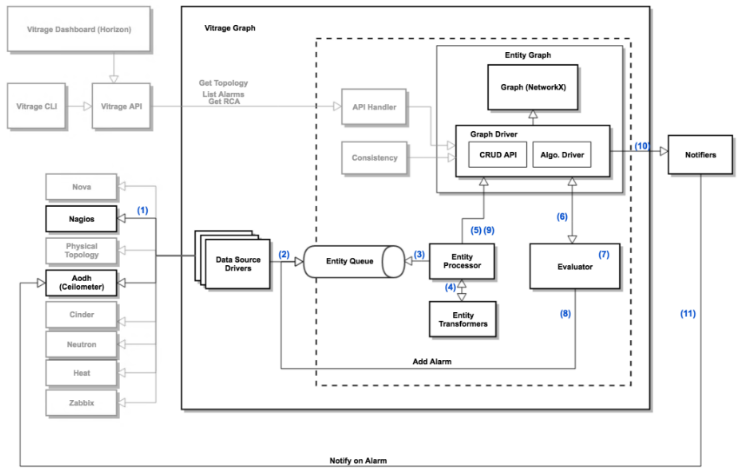


Vitrage bao gồm các thành phần:

* ClI, Drashboard, API: nhận yêu cầu người dùng, gọi đến các thành phần khác.
* Vitrage data source driver: Nhiệm vụ lấy input đầu vào thông tin về các resource hệ thống về cho vitrage graph hiển thị. Dữ liệu bao gồm thông tin như: resource có các thực thể nào? quan hệ giữa các thực thể đó. Vd: thực thể trong aodh là “alarm”, của nova là “zone” chứa “host” chứa “instance”. Các datasoucrce gồm:
* Từ nguồn openstack : aodh, ceilometer, cinder, heat, neutron, nova.
* Từ nguồn extenal monitor, gồm: collectd, zabix, nagios, prometheus
* static datasource: đọc thông tin từ các file cấu hình người dùng cấp
* Vitrage graph: Giữ các thông tin từ datasouce và trình diễn nó ra, đồng thời cũng lấy cập nhật của khối evaluator khi có. Driver để vẽ đồ thị là networkX , đồng thời cũng inplement môt số phép xử lý đồ thị khi được gọi, vd duyệt, tìm đỉnh, cạnh…
* Vitrage evaluator: Khi có thông báo từ graph về một số thay đổi của các thực thể, nó truy vấn trong các kịch bản (template do người dùng định nghĩa từ trước) và áp dụng các thay đổi trạng thái cho các thực thể (set\_state, raise\_alarm, add\_causual\_relationship)
* Vitrage notifiers: có nhiệm vụ thông báo cho các thành phần khác về các thay đổi trong trang thái hệ thống, áp dụng các action lên các dịch vụ đó. Hiện có các plugin cho: aodh, nova, mistral , snmp, webhook
* Vitrage machine leaning: Thiết kế ra nhằm giúp vitrage có thể: xem xét từ các alarm , deduce-alarm đã xuất hiện, tính rra sự liên quan giữa các alarm, gợi ý cho người dùng viết các template mới.

Ví dụ một luồng hoạt động cơ bản:

Trường hợp thêm có một alarm :



Khi phát hiện switch down, vitrage nhận được alarm từ nagios . Thiết lập kịch bản sẽ thực hiện là: tạo deduce-alarm trên các host gắn với switch, các instance gắn trên host và triger lại cho aodh.

1. Nagios gửi alarm về messeaqueue của vitrage, thành phần vitrage datasource Driver nhận thông báo.

2. Vitrage datasource Driver gửi thông tin vào vitrage entity queue

3. Entity processor polling event từ entity queue, xác định thông tin của thực thể cần thêm

4. Entity trasformers chuyển thông tin của thực thể về các đối tượng trong graph: đỉnh, cạnh, neghbors,…

5. Entity processor gọi graph api thêm đỉnh, cạnh mới.

6. Khi có bất kỳ thay đổi nào về graph thì evaluator cũng được gọi đến. Evaluator biết được có 1 đỉnh mới được add

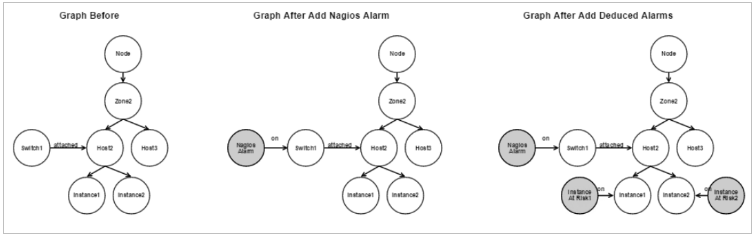
7. Evaluator xem xét các template đã có thì thấy có 1 kịch bản cần áp dụng: bật deduce-alarm error trên mỗi instance gắn với host đó.

8. Evaluator push alarm vào entity queue.

9. Graph update thêm các alarm mới

10. Graph thêm thông báo rằng có 1 alarm mới trên instance, đẩy thông báo này vào messasge queue

11. Vitrage Notifier định nghĩa alarm cho aodh - set state “alarm”



## 2.7. Heat

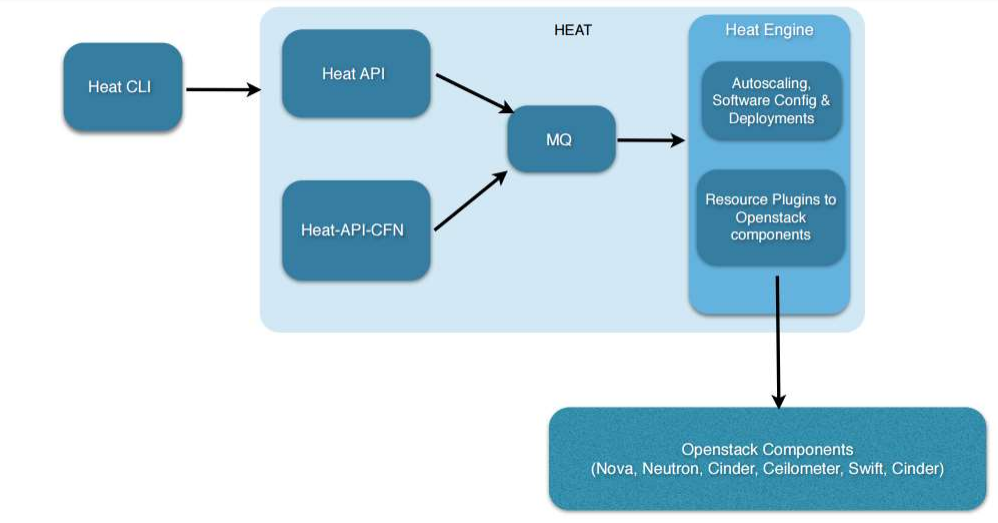
Heat project hay còn gọi là Orchestration service là dịch vụ phối hợp các ứng dụng cloud bằng cách sử dụng các định dạng template thông qua REST API có sẵn trong OpenStack. Về bản chất đây là “infrastructure as code” trong môi trường cloud

Vai trò của heat:

Heat cung cấp template để mô tả tài nguyên cần cài đặt cho hệ thống cloud.

Hệ thống nhận template và đầu vào các biến từ người dùng và tạo lượng tài nguyên theo yêu cầu bằng cách thực hiện các lời gọi đến OpenStack APIs.

* Template Heat mô tả cơ sở hạ tầng của tài nguyên cần trong file text mà con người có thể đọc và ghi được, và có thể quản lý bởi version control tool. Người dùng từ đó có thể viết code heat tự động hóa quá trình cài đặt instance của mình
* Heat có cung cấp dạng template về auto scaling server, sử dụng tài nguyên thuộc loại “scaling - group”- loại tài nguyên đặc biệt giúp quản lý kịch bản scale up, scale down . Khi kết hợp với với aodh, giúp hệ thống tự động scaling một nhóm instance khi có alarm đi đến.



Kiến trúc của heat:

* Heat-CLI: (command line client) là CLI để giao tiếp với heat-api và chạy kịch bản Orchestration
* Heat-api : thành phần cung cấp OpenStack-native REST API xử lý các yêu cầu API bằng cách gửi chúng đến heat-engine thông RPC.
* Heat-api-cfn: thành phần cung cấp API truy vấn AWS, tương thích với AWS CloudFormation và xử lý các yêu cầu API bằng cách gửi chúng tới heat-engine thông qua RPC.
* Heat-engine: thành phần chính có nhiệm vụ chính là điều phối việc khởi tạo templates và cung cấp các sự kiện cho người dùng API.

## 2.8. Mistral

Mistral là  OpenStack workflow service, project giúp người dùng có khả năng định nghĩa, thực hiện và quản lý một task hoặc một workflow. Nó cũng định nghĩa workflow này dưới dạng template như heat và khi thực hiện sẽ gọi đến các API của openstack, một loại “infrastructure as code”. Nhưng khác với heat, một workflow bao gồm các task nhỏ hoạt động theo IFTTT (if this, then that) như một cấu trúc điều khiển của một ngôn ngữ lập trình.

Heat giúp người dùng chạy kịch bản về triển khai một stack các tài nguyên ảo hóa cần, còn mistral giúp người dùng chạy kịch bản về một stack các hành động càn. Ví dụ cùng muốn tạo instance nhưng template của 2 bên sẽ khác biệt:

* Template của heat sẽ mô tả: user cần loại tài nguyên gì, số lượng và mối quan hệ giữa chúng.
* Template của mistral sẽ mô tả: action 1: nova.create-instance 🡪 action 2: cinder.create volume 🡪 action 3: cinder.attach-volume 🡪 if success send success-email 🡪 if fail retry 5 times

Vai trò của mistral trong hệ thống:

* Mistral giúp ích cho user cloud provider đặt các kịch bản để tự động các thao tác với hệ thống.
* Mistral kết hợp với vitrage giúp thực hiện quá trình auto healing instance

Cảnh báo(về host hoặc instance) gửi đến 🡪 Vitrage suy luận về tác động của cảnh báo với các 🡪 Mistral tác động lại với các tài nguyên ảo hóa của hệ thống.

# CHƯƠNG 3: PHÁT TRIỂN HỆ THỐNG GIÁM SÁT

## 3.1. Phân tích hệ thống

### 3.1. Các đối tượng hệ thống

Hiện tại, mô hình cloud triển khai từ Openstack có các đối tượng sau:

Cloud provider: người quản trị hệ thống cloud, nhà cung cấp : infrastructure – hạ tầng máy ảo, platform – nền tảng, software - ứng dụng. Ở đây tập trung vào tầng infrastructure, tức cloud provider có trách nhiệm quản trị server vật lý (host) và cung cấp ra máy ảo (instance)

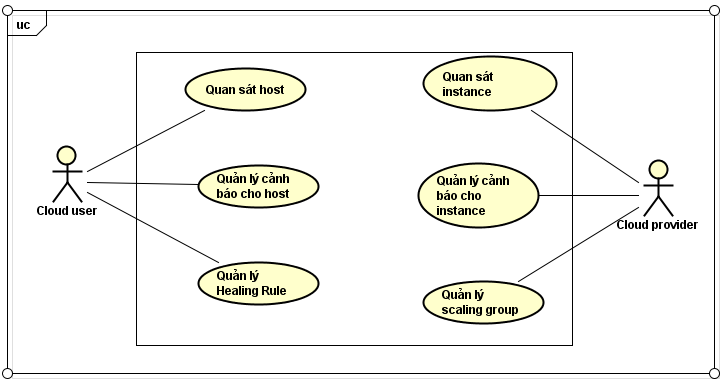
Cloud user: người sử dụng các instance từ cloud provider. Từ các instance này cài đặt lên ứng dụng của mình, nhằm phục vụ người dùng cuối.

**Cloud provider** và **cloud user** được Openstack phân quyền theo mô hình RBAC :

* Project : mỗi user đều thuộc về một số project, có vai trò như một dự án trong thực tế: ví dụ: user thuộc “dự án về hệ thống đăng ký học tập”, “dự án data mining”,.. một project có nhiều user. Mỗi project được thiết lập quy ước các giới hạn về tài nguyên sử dụng ( vd bao nhiêu ram, cpu, instance,…)
* Role admin: role của cloud provider, role này có quyền quản lý vòng đời của mọi tài nguyên: tạo xóa instance, volumes ảo,… Tuy nhiên không có quyền đăng nhập sử dụng các instance, là tài sản sẽ bàn giao cho cloud user. Role được gán mặc định cho username “admin”
* Role user: role của cloud user, role này có quyền tạo, xóa tài nguyên ảo: instance, volume,.. trong project của mình. Truy cập vào instance. Role này được user admin gán cho các user khác.

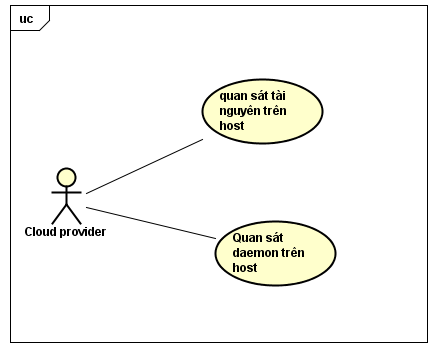
### 3.2. Mô tả uscase

#### 3.2.1. Usecase tổng quan:

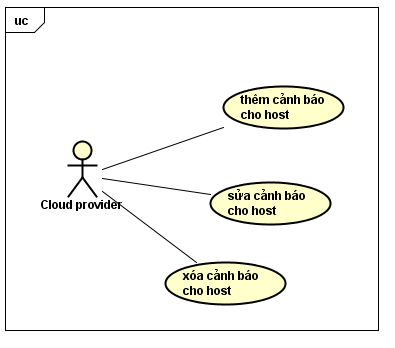


#### 3.2.2. Usecase của cloud-provider

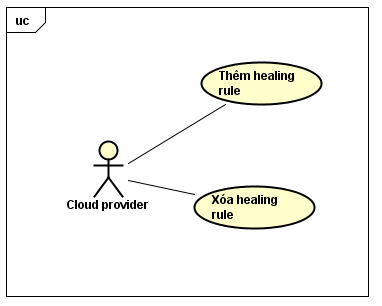
**Usecase chức năng quan sát host:**



**Usecase quản lý cảnh báo cho host:**



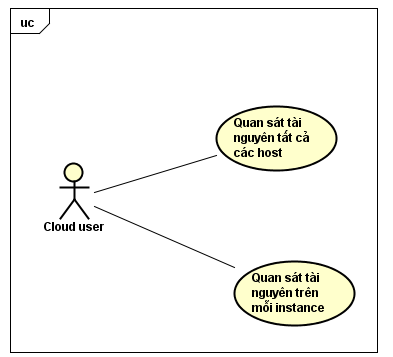
**Usecase quản lý healing rule:**



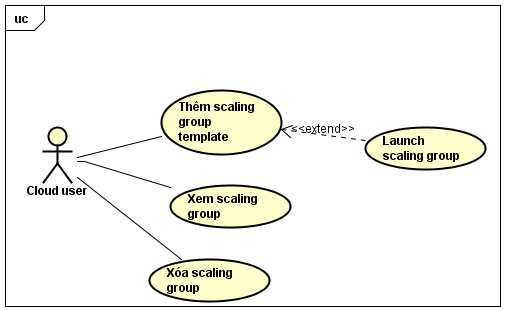
Mô tả…

#### 3.2.3. Usecase của cloud user

**Usecase quan sát instance**



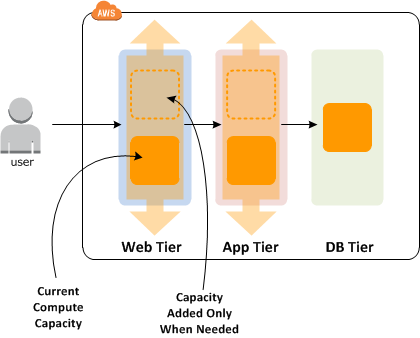
**Usecase quản lý scaling group**



Auto scaling sẽ giúp người dùng giảm bớt các server hoạt động trong thời gian thấp điểm và tự động tăng tài nguyên khi host cao tải.

Việc scale trong môi trường cloud sẽ khác với khái niệm nâng cấp server thông thường: trong thực tế, người ta sẽ không xin tăng ram, cpu cho máy ảo khi cao tải do điều này khiến máy xảy ra downtime (ngay những lúc đang phục vụ nhu cầu cao). Thay vào đó họ sẽ tăng thêm số lượng instance chạy ứng dụng và cấu hình load balancing cho chúng.

Mô hình của amazone:



App, web được bố trí trên các instance thuộc các scaling-group.

Một scaling-group bao gồm:

* Nhóm các instance hiện tại, số lượng instance tối đa, tối thiểu trong group
* Các alarm aodh đã xác định kịch bản sẽ trả về alarm khi metric của tài nguyên thế nào...
* Scaling rule : ví dụ khi có alarm cpu cao tải thì +1 instance, cpu thấp thì -1 instance

Yêu cầu về ứng dụng của người dùng: database-instance và application-instance chạy tách biệt. Ta không auto scale với database do sẽ xảy ra tình trạng thiếu nhất quán giữa các database được scale ra.

Openstack có thể thực hiện được mô hình này của amazone với các công nghệ của heat, telemetry

Cơ chế trong hệ thống:

Khi có cảnh báo alarm cao tải – thấp tải trên instance 🡪 gọi heat api để scaling up, down như scaling rule 🡪 heat thực hiện kịch bản

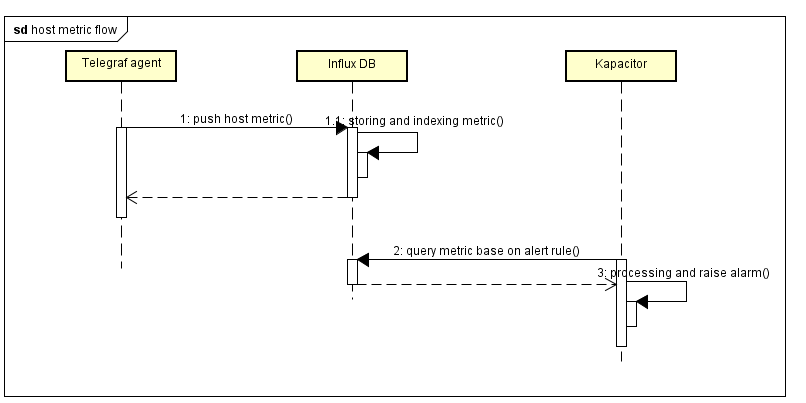
## 3.1. Thiết kế hệ thống

### 3.1.1.Bố trí các thành phần monitor.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Categories** | **Monitor Stack** | **Metrics colector** | **Storage metric** | **Alarming** | **Trigger action** |
| Host physic resource +  OpenStack daemons | TICK | Telegraf | InfluxDB | Kapacitor | Kapacitor |
| OpenStack resouces (virtural resources – volumes, images, instances, ports, networks,…) | Telemetry | Ceilometer | Gnocchi | Aodh | Vitrage + Mistral + Heat |

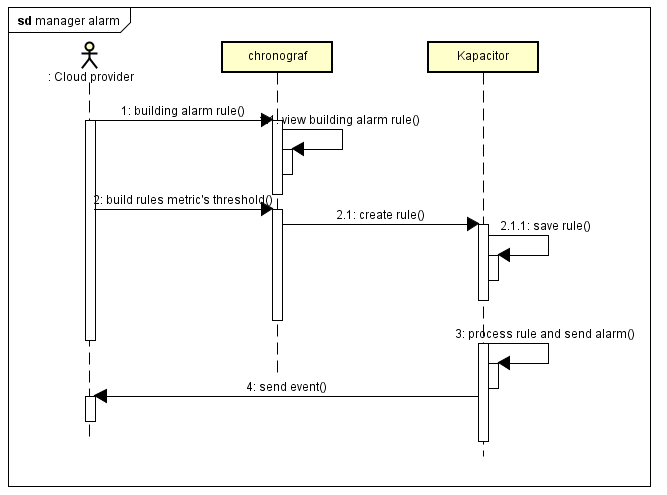
### 3.1.2 Mô tả luồng hoạt động

Luồng hoạt động để xử lý metric tài nguyên host vật lý.



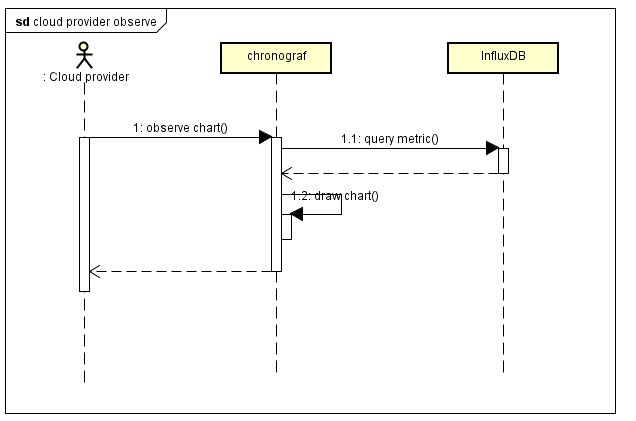
* Telegraf cài trên các host cần monitor, thu thập thông số của host, trả về thông tin thành các datapoint, một point bao gồm:
  + Time (timestamp)
  + Một measurement (ví dụ "cpu\_load")
  + Ít nhất một key-value field (giá trị đo được, ví dụ "value=0.64", hoặc "temperature=21.2")
  + key-value tags chứa metadata của value (ví dụ “host=server01”)
* Các point được push về cho influx db lưu trữ, Influx db đánh chi mục cho timestamp và tag
* Kapacitor đọc và xử lý metric từ influx db, có 2 kiểu xử lý:
  + Stream: đăng ký các metric mà ngay khi influx nhận được sẽ lấy ngay, vd: status service để monitor trường hợp service up-down
  + Batch: sau một khoảng thời gian, gửi query đến influx các khoảng nhất dữ liệu nhất đinh, vd: cứ 60s truy vấn cpu trung bình trong khoảng thời gian 5 phút, kiểm tra xem nó có vượt ngưỡng 90%

Luồng hoạt động cho chức năng cloud provider thêm cảnh báo cho tài nguyên host vật lý



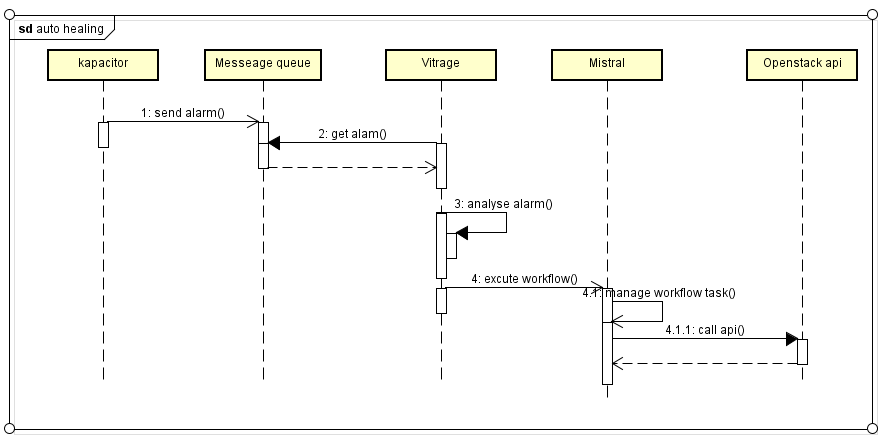
* Cloud provider vào giao diện của chronograf, chọn tạo alert rule. Chronograf hiển thị giao diện giúp người dùng tạo alert rule
* Người dùng xác nhận tạo rule, thông tin được chronograf gửi về kapacitor
* Kapacitor lưu rule và xử lý theo alert rule đã thiết lập. gửi về người dùng khi có rule được áp dụng. vd gửi mail khi cpu.system.usage >90 % .

Luồng hoạt động cho chức năng cloud provider quan sát tài nguyên host vật lý



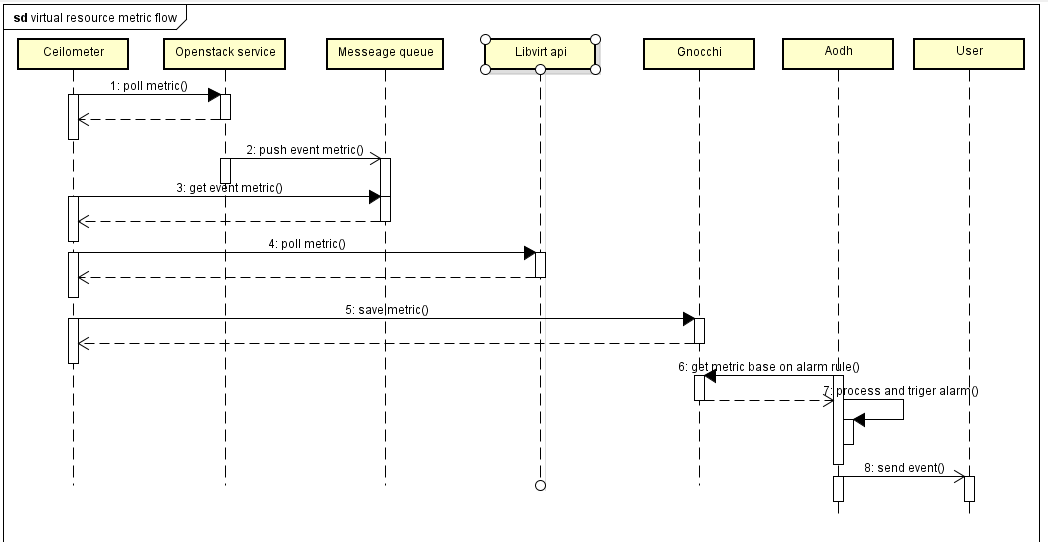
* Cloud provider vào giao diện của chronograf, chọn vào biểu đồ mô tả các metric muốn xem
* Chronograf truy vấn influxDB về metric trong một khoảng thời gian . Vẽ ra biểu đồ cho người dùng quan sát

Luồng hoạt động thực hiện auto healing khi có alarm trả về

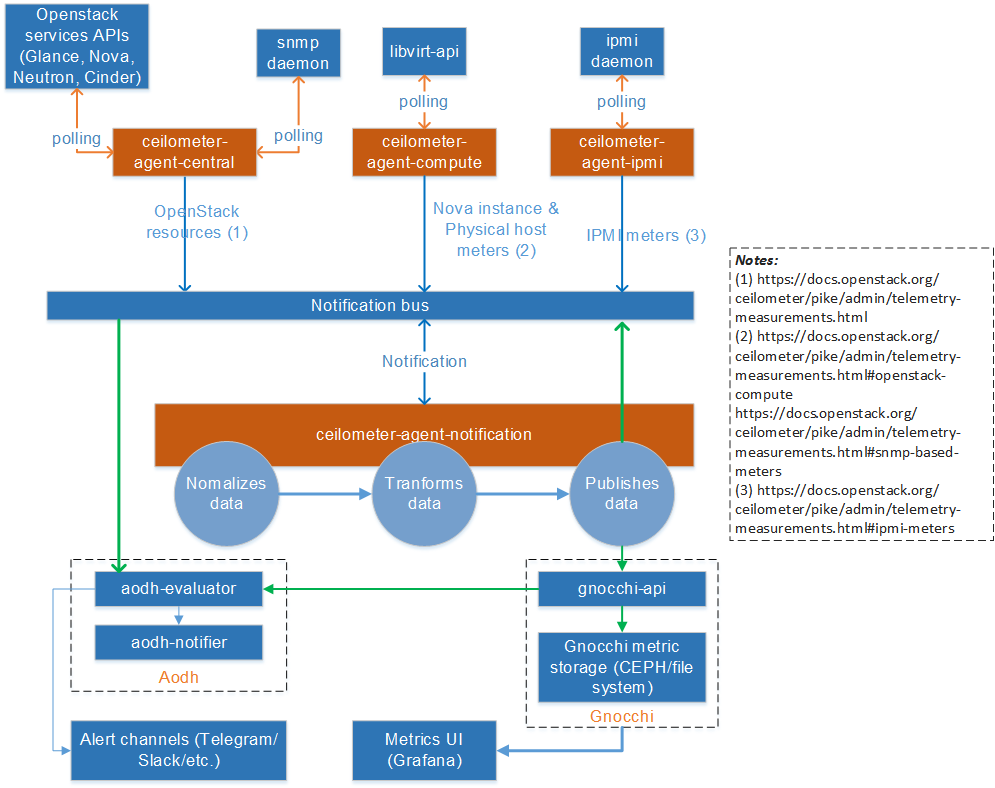


* Kapacitor trả về alarm :có lỗi tại một thành phần của host, gửi vào messeage queue
* Vitrage nhận alarm này, xử lý (theo kịch bản mà người quản trị cloud-provider đã định từ trước). vd kịch bản nêu ra: host down sẽ khiến tất cả instance down theo hướng xử lý trong kịch bản sẽ là gọi workflow evacutate các inance này
* Vitrage tìm các instance bị ảnh hưởng, gọi workflow cho các instance đó
* Mistral nhận được lời gọi workflow từ vitage và thực hiện các task.
* Mistral gọi nova api để tác động lên instance

Luồng hoạt động để xử lý metric tài nguyên ảo hóa instance

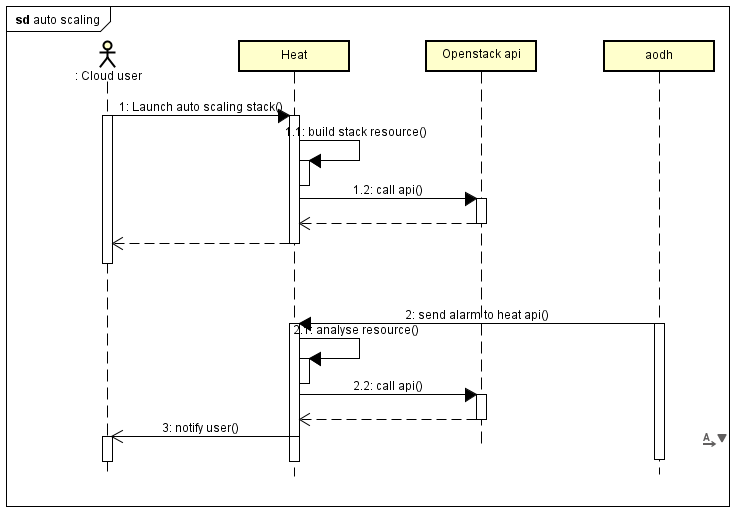


* Ceilometer truy vấn các api của project khác của openstack để lấy về thông tin của insance, điển hình như id, os, volume attach
* Các openstack service khi có một sự kiện đến instance cũng gửi các event đó đến messeage queue, vd như instance create, shutdown, resize,…. Ceilometer lắng nghe trên queue các sự kiện này
* Ceilomter agent trên host compute của hệ thống truy vấn libvirt api để lấy ra được các metric về instance , vd như cpu usage, mem usage, disk iops,..
* 3 nguồn metric trên được ceilometer đẩy về cho gnochi lưu trữ
* Gnocchi lưu trữ , hậu xử lý dữ liệu (gộp các data point thành các điểm data của 1 giây, 1 phút, 60 phút...) và đánh chỉ mục cho data
* Aodh từ rule được người dùng thiết lập truy vấn thông tin cần thiết, xử lý xem có metric nào quá ngưỡng để trả về cảnh báo cho người dùng



Hình ảnh về luồng hoạt động của các microservice của các dịch vụ này

Luồng hoạt động thực hiện auto scaling .



* User tạo template mô tả về scaling group. Gọi đến heat api (dùng giao diện dashboard horizon của openstack) để launch stack
* Heat gọi openstack api tạo ra tài nguyên đúng như người dùng yêu cầu. bao gồm
  + Các instance , volume
  + Định nghĩa về alarm xuất hiện khi metric cao (thấp) quá ngưỡng
  + Luật scale up, down khi có alarm
* Khi có sự kiện aodh phát hiện ra metric cao-thấp bất thường
  + Aodh gọi đến heat api. Tín hiệu đến heat : “hãy scale in-out tài nguyên ”
  + Heat nhận được lời gọi này, tự động tạo thêm instance, volum, chạy script khởi động,..
  + Heat thông báo người dùng trên giao diện dashboard horion

## 3.3. Xây dựng hệ thống.

### 3.3.1. Các yêu cầu cho hệ thống sẵn có

Hệ thống monitor của em có thể tích hợp với hệ thống cloud sẵn có bao gồm các yêu cầu về phần mềm như sau:

|  |  |
| --- | --- |
| Hệ điều hành | Ubuntu 16.04 xenial |
| Máy controller | Cài đủ các service core của openstack:  Keystone, Nova, Glance, Neutron, Glance, Cinder |
| Máy compute | Cài đủ các service core của openstack:  neutron-agent , nova-compute cinder-volume. |
| Storage | Share storage: ceph, glusterfs, nfs… làm backend của cinder |

### 3.3.2. Chi tiết về việc xây dựng hệ thống.

**a) Các service cài đặt thêm**

|  |  |
| --- | --- |
| Máy controller | Telegraf  Ceilometer (agent-notification agent-central)  Gnocchi(api, metricd)  Aodh(api, evaluator, notifier, listener)  Vitrage, Heat, Mistral  Horion (thêm plugin dashboard Heat, vitrage, mistral) |
| Máy compute | Telegraf |
| Máy monitor-center | InfluxDB, Kapacitor, Chronograf, Grafana |

Trong đó, máy controller và máy monitor-center có thể nhập làm một.

**b) Ước lượng tài nguyên cần cho các service monitor**

Tài nguyên host dành cho dịch vụ monitor:

|  |  |  |  |
| --- | --- | --- | --- |
|  | RAM | CPU | Disk |
| Máy controller | +5 GB | + 4 core | + 10GB |
| Máy compute | +1 GB | + 2 core |  |
| Máy monitor-center | +4 GB | + 2 core | +5 GB với mỗi node cần monitor |

Các số liệu em lấy từ thông tin các service chạy trong môi trường lab: monitor 3 server.

Tính toán dung lượng của database gnocchi cần cho metric của instance:

* Kích thước dữ liệu lưu trữ tại mỗi datapoint:
  + Trường hợp tệ nhất: 8.04 bytes/point
  + Trường hợp tốt nhất: 0.05 bytes/point
* Số loại metric tối đa tại mỗi instance tạo ra:
  + Resource type instance: 44
  + Resoucre type volume disk: 15

**🡪** tại một thời điểm, dung lượng metric tối đa 1 instance tạo ra:

8.04 \* (44+15) = 475 bytes

* Gộp 60s datapoint thành 1 metric-gộp 60s ( phép toán avg, sum,count…) (A)
* Gộp 60 các metric 60s trên thành 1 metric gộp 3600s (B)
* Nhóm các metric A: lưu trong 7 ngày
* Nhóm các metric B: lưu trong 365 days

Vậy một năm sẽ có 24\*7\*60 loại A, và 365\*24 metric loại B

🡪 Lượng tài nguyên chiếm trong database gnocchi, sử dụng danh cho 1 instance là:

[7\*24\*60 + 365\*24] \* 475 bytes = 8949000 byte = 9MB

Đối host vật lý dùng influxDB lưu metric, tuy em không có cách nào tính toán về lý thuyết, tuy nhiên theo như em tham khảo tại hệ thống thực, gồm 20 server vật lý, mỗi tuần sẽ có thêm khoảng vài GB dữ liệu được ghi thêm vào.

**c) Cài đặt hệ thống**

Hệ thống được cài đặt tự động từ ansible, download code ansible tại:

<https://github.com/ngohoa211/bash>

Cần chuẩn bị nhập vào các biến trong code ansible như sau:

File inventory/host:

|  |  |  |
| --- | --- | --- |
| **Tên biến** | **Giải thích** | **Ví dụ** |
|  |  |  |

# CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ

**Tại môi trường lab, các service được tổ chức như sau**



## 4.1.Tự động evacuate instance khi host down

*Đây là kịch bản do cloud provider thực hiện từ các chức năng của hệ thống.*

**a) Kịch bản**

Monitor thông tin:

* host-down: compute-01 không trả về metric trong 120 s, recheck 60s 1 lần

Auto healing:

* Di tản các host từ node bị down sang host khác



Cơ chế hoạt động: TICK monitor host và cảnh báo về host down 🡪 Gửi alarm cho vitrage 🡪 vitrage xác định host down sẽ khiến instance trên đó ảnh hưởng 🡪 vitrage gọi mistral thực hiện workflow evacuate host

(Volumes của máy ảo nằm trên share storage (ceph) nên dù có host down vẫn có khả năng tạo lại instance đó trên host mới )

Đây là kịch bản do cloud provider thực hiện, sử dụng các chức năng của hệ thống

**b) Cách thực hiện:**

**Cấu hình monitor:**

Tại kapacitor, ta định nghĩa alarm host down như sau:

Tạo file tick\_script.txt:

|  |
| --- |
| var db = 'telegraf'  var rp = 'autogen'  var measurement = 'cpu'  var groupBy = ['host']  var whereFilter = lambda: ("cpu" == 'cpu-total') AND ("host" == 'controller' OR "host" == 'node2' OR "host" == 'node3')  var period = 5m  var name = 'host offline'  var idVar = name + '-{{.Group}}'  var message = 'host offline'  var levelTag = 'level'  var messageField = 'message'  var durationField = 'duration'  var triggerType = 'deadman'  var threshold = 0.0  var data = stream  |from()  .database(db)  .retentionPolicy(rp)  .measurement(measurement)  .groupBy(groupBy)  .where(whereFilter)  var trigger = data  |deadman(threshold, period)  .message(message)  .id(idVar)  .levelTag(levelTag)  .messageField(messageField)  .durationField(durationField)  .exec('/etc/kapacitor/kapacitor\_vitrage.py', 'rabbit:// rabbit\_user: rabbit\_pass @controller') |

Ở script trên:

.exec(‘/usr/bin/python’, ‘/etc/kapacitor/kapacitor\_vitrage.py’,‘rabbit://rabbit\_user:rabbit\_pass@controller’)

Script này định nghĩa khi có alarm thì sẽ gửi thông tin alarm này về messeaqueue của openstack. Vitrage sẽ đọc được alarm này và cho vào graph

File kapacitor\_vitrage.py download tại <https://github.com/openstack/vitrage/blob/master/vitrage/datasources/kapacitor/auxiliary/kapacitor_vitrage.py>

Chạy lệnh:

|  |
| --- |
| kapacitor define check\_host\_down -tick tick\_script.txt |

**Cấu hình healing rule**

Tạo file /etc/vitrage/kapacitor\_conf.yaml , mục đích ánh xạ alarm từ kapacitor vừa nhận được đến các đối tượng của openstack :

|  |
| --- |
| kapacitor:  - alert:  host: compute-(.\*)  vitrage\_resource:  type: nova.host  name: ${kapacitor\_host}  - alert:  host: (.\*)  vitrage\_resource:  type: nova.instance  name: ${kapacitor\_host} |

vd: nếu có alarm trên máy có hostname “compute-01” sẽ map với nova.host “compute01” ,

tương tự hostname “compute-02” sẽ map với nova.host “compute-02” …

Cấu hình healing rule tại vitrage: nếu có alarm host mất kết nối, quyết định rằng host down , gọi workflow mistral :evacuate instance

vi ~/host\_down\_scenario.yaml

|  |
| --- |
| metadata:  version: 2  name: deduced\_alarm\_for\_all\_host\_in\_error  type: standard  description: raise deduced alarm for all hosts in error  definitions:  entities:  - entity:  category: ALARM  name: host offline  type: kapacitor  template\_id: host\_alarm  - entity:  category: RESOURCE  type: nova.host  template\_id: host  - entity:  category: RESOURCE  type: nova.instance  template\_id: instance  relationships:  - relationship:  source: host\_alarm  target: host  relationship\_type: on  template\_id : alarm\_on\_host  - relationship:  source: host  target: instance  relationship\_type: contains  template\_id : host\_contains\_instance  scenarios:  - scenario:  condition: alarm\_on\_host and host\_contains\_instance  actions:  - action:  action\_type : raise\_alarm  properties:  alarm\_name: instance\_offline  severity: critical  action\_target:  target: instance  - action:  action\_type **: execute\_mistral**  properties:  workflow: **evacuate\_instance**  input:  instance\_id: get\_attr(instance, id)  retries: 5 |

vitrage template create --type standard --path ~/host\_down\_scenario.yaml

Tại mistral định nghĩa **evacuate\_instance** workflow trên:

vi ~/evacutate\_instance.yaml

|  |
| --- |
| ---  version: '2.0'  instance\_evacuate:  type: direct  input:  - instance\_id  tasks:  get\_instance\_status\_before:  action: nova.servers\_find id=<% $.instance\_id %>  publish:  instance\_name: <% task(get\_instance\_status\_before).result.name %>  status\_before: <% task(get\_instance\_status\_before).result.status %>  host\_before: <% task(get\_instance\_status\_before).result["OS-EXT-SRV-ATTR:host"] %>  on-success: evacuate\_instance  on-error: send\_error\_email  evacuate\_instance:  action: nova.servers\_evacuate server=<% $.instance\_id %>  retry:  delay: 10  count: 10  on-success: wait\_for\_instance\_rebuild  on-error: send\_error\_email  wait\_for\_instance\_rebuild:  action: nova.servers\_find id=<% $.instance\_id %> status="REBUILD"  retry:  delay: 2  count: 30  on-complete: wait\_instance\_status\_active  wait\_instance\_status\_active:  action: nova.servers\_find id=<% $.instance\_id %> status="ACTIVE"  retry:  delay: 10  count: 30  on-complete: get\_instance\_status\_after  get\_instance\_status\_after:  action: nova.servers\_find id=<% $.instance\_id %>  publish:  status\_after: <% task(get\_instance\_status\_after).result.status %>  host\_after: <% task(get\_instance\_status\_after).result["OS-EXT-SRV-ATTR:host"] %>  on-complete: check\_diffrent\_host  check\_diffrent\_host:  action: std.noop  on-complete:  - send\_success\_email: <% $.status\_before = $.status\_after and $.host\_before != $.host\_after %>  - send\_error\_email: <% $.status\_before != $.status\_after or $.host\_before = $.host\_after %>  send\_error\_email:  action: std.email  input:  to\_addrs: 'admin@demo.com'  subject: ERROR evacuate vm  body: |  We try to evacuate vm <% $.instance\_id> when host have problem  Please look at mistral workflow <% execution().id %> for more detail  from\_addr: admin@demo.com  smtp\_server: smtp.google.com  smtp\_password: SECRET  send\_success\_email:  action: std.email  input:  to\_addrs: [admin@demo.com]  subject: SUCCESS evacuate vm  body: |  We evacuate vm <% $.instance\_id> when host have prolem.  Please look at mistral workflow <% execution().id %> for more detail  from\_addr: admin@demo.com  smtp\_server: smtp.google.com  smtp\_password: SECRET |

mistral workflow-create ~/evacutate\_instance.yaml

**c) Thử nghiệm.**

Thành công: khi tắt một host, hệ thống tự động chuyển các instance hiện có sang host còn lại.

## 4.2.Tự động live migrate instance từ host cao tải sang host ít tải hơn.

*Đây là kịch bản do cloud provider thực hiện từ các chức năng của hệ thống.*

...

## 4.3.Tự động scale group-instance khi có cảnh báo về tài nguyên

*Đây là kịch bản do cloud user thực hiện từ các chức năng của hệ thống.*

**a) Kịch bản**

Monitor thông tin cpu\_util: phần trăm sử dụng cpu của instance

Auto scaling:

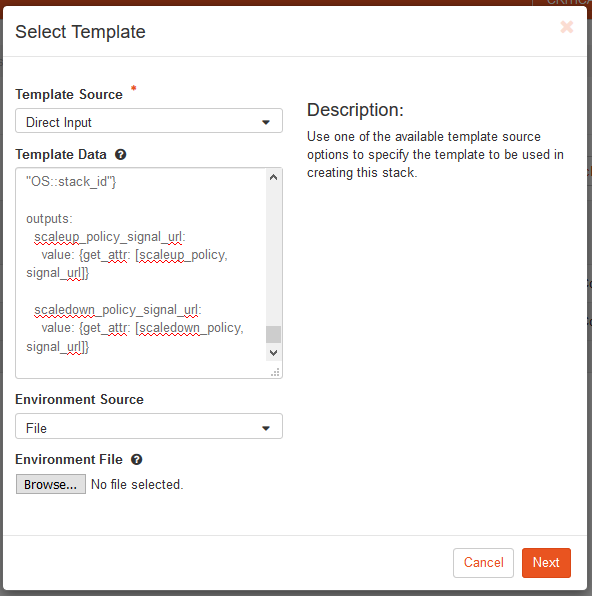
* Group instance mới tạo bao gồm 1 instance.
* Nếu có cảnh báo cpu > 80% tự động tạo thêm 1 instance
* Nếu có cảnh báo cpu < 5% giảm bớt 1 instance
* Số instance tối đa trong group là 3, tối thiểu trong group là 1

**b) Cách thực hiện:**

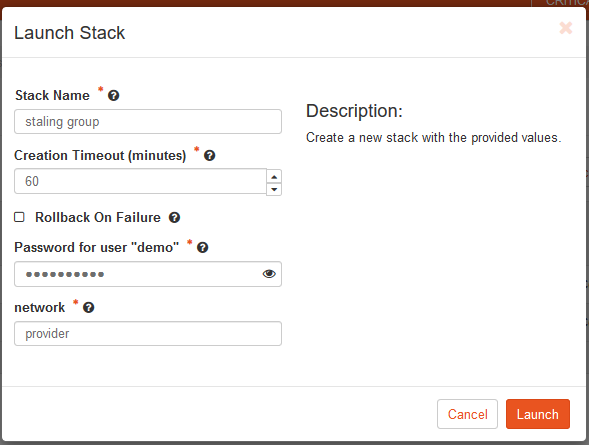
User tạo template scaling-group của như sau:

|  |
| --- |
| heat\_template\_version: 2016-10-14  description: Example auto scale group, policy and alarm  parameters:  network:  type: string  description: Network ID to use for the instance.  resources:  scaleup\_group:  type: OS::Heat::AutoScalingGroup  properties:  cooldown: 300  desired\_capacity: 1  max\_size: 3  min\_size: 1  resource:  type: OS::Nova::Server  properties:  image: cirros  flavor: m1.tiny  key\_name: mykey  networks:  - network: { get\_param: network }  scaleup\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaleup\_group }  cooldown: 300  scaling\_adjustment: 1  scaledown\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaleup\_group }  cooldown: 300  scaling\_adjustment: -1  cpu\_alarm\_high:  type: OS::Aodh::GnocchiAggregationByResourcesAlarm  properties:  description: Scale up if CPU > 80%  metric: cpu\_util  aggregation\_method: mean  granularity: 60  evaluation\_periods: 1  threshold: 80  resource\_type: instance  comparison\_operator: gt  alarm\_actions:  - str\_replace:  template: trust+url  params:  url: {get\_attr: [scaleup\_policy, signal\_url]}  query:  str\_replace:  template: '{"=": {"server\_group": "stack\_id"}}'  params:  stack\_id: {get\_param: "OS::stack\_id"}  cpu\_alarm\_low:  type: OS::Aodh::GnocchiAggregationByResourcesAlarm  properties:  metric: cpu\_util  aggregation\_method: mean  granularity: 60  evaluation\_periods: 1  threshold: 5  resource\_type: instance  comparison\_operator: lt  alarm\_actions:  - str\_replace:  template: trust+url  params:  url: {get\_attr: [scaledown\_policy, signal\_url]}  query:  str\_replace:  template: '{"=": {"server\_group": "stack\_id"}}'  params:  stack\_id: {get\_param: "OS::stack\_id"}  outputs:  scaleup\_policy\_signal\_url:  value: {get\_attr: [scaleup\_policy, signal\_url]}  scaledown\_policy\_signal\_url:  value: {get\_attr: [scaledown\_policy, signal\_url]} |

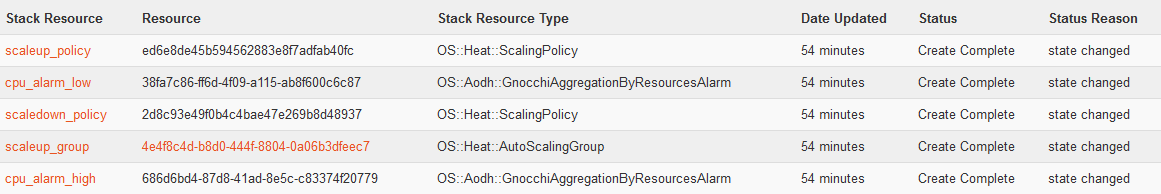
User vào giao diện web horizon, tab Orchestrasion > stack > lauch stack



Nhập các biến đầu vào cho template:



Các resource của stack sau khi tạo xong:



**c) Thử nghiệm**

Launch stack trên ta sẽ được một stack gồm 1 instance, ssh vào instance này và chạy test cpu

|  |
| --- |
| #dd if=/dev/zero of=/dev/null &  #dd if=/dev/zero of=/dev/null &  #dd if=/dev/zero of=/dev/null & |

Sau một thời gian sẽ có alarm cảnh báo cpu cao , hiển thị trên màn hình của người dùng

Thành công: sau 60 s hệ thống sẽ tự động scale stack thành 2 instance

# CHƯƠNG 5: KẾT LUẬN

*Phát triển thêm , những tính đóng góp, so sánh với cách monitor thủ công, các giải pháp monitor đã có trên thị trường cloud…*

# TÀI LIỆU THAM KHẢO

[1] Patrick “Making Facebook Self-Healing” xem tại <https://code.fb.com/data-center-engineering/making-facebook-self-healing/>

[2] openstack community “ceilometer document” xem tại <https://docs.openstack.org/ceilometer/latest/>

[3] gnocchi team “gnocchi document” xem tại <https://gnocchi.xyz/>

[5] Cuong nguyen “monitor system” xem tại <https://devopsz.com/monitoring-system-part-1/>

[6] openstack community “self-healing-sig” xem tại <https://docs.openstack.org/self-healing-sig/latest/use-cases/heat-mistral-aodh.html>

[7] readhat company “auto scaling for instance ” xem tại <https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/11/pdf/auto_scaling_for_instances/Red_Hat_OpenStack_Platform-11-Auto_Scaling_for_Instances-en-US.pdf>