Contents

[CHƯƠNG 1: ĐẶT VẤN ĐỀ 2](#_Toc8290430)

[1. Lý do chọn đề tài 2](#_Toc8290431)

[2. Mô tả 2](#_Toc8290432)

[3. Mục tiêu 3](#_Toc8290433)

[3.1.Chức năng 3](#_Toc8290434)

[3.2.Hiệu năng 3](#_Toc8290435)

[CHƯƠNG 2: CÔNG NGHỆ SỬ DỤNG 4](#_Toc8290436)

[3.1. Ceilometer 4](#_Toc8290437)

[3.2. Gnocchi 11](#_Toc8290438)

[3.3. Aodh 13](#_Toc8290439)

[3.4. Granfana 15](#_Toc8290440)

[3.5. TICK stack 16](#_Toc8290441)

[3.4. Vitrage 18](#_Toc8290442)

[3.4. Heat 21](#_Toc8290443)

[3.5. Mistral 22](#_Toc8290444)

[CHƯƠNG 3: THIẾT KẾ CHI TIẾT HỆ THỐNG 22](#_Toc8290445)

[CHƯƠNG 4: KỊCH BẢN SỬ DỤNG 24](#_Toc8290446)

[**4.1.Tự động evacuate instance khi host down:** 24](#_Toc8290447)

[**4.1.Tự động scale group-instance khi có cảnh báo tài nguyên trên máy ảo** 28](#_Toc8290448)

[**4.1.Tự động live migrate instance từ host cao tải sang host ít tải hơn.** 31](#_Toc8290449)

[CHƯƠNG 5: TÍNH HIỆU QUẢ VÀ ĐÓNG GÓP CỦA HỆ THỐNG 31](#_Toc8290450)

[TÀI LIỆU THAM KHẢO 31](#_Toc8290451)

# CHƯƠNG 1: ĐẶT VẤN ĐỀ

## 1. Lý do chọn đề tài

Giám sát 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 tốt giúp ta không những sẽ cho ta biết được tình trạng của hệ thống gồm tài nguyên (ram, cpu, network, ổ cứng) sử dụng và các dịch vụ cần quan tâm, mà còn có khả năng giúp thực hiện những tác động vào hệ thống khi những thông số trên trở nên bất thường. Chính điều này sẽ giúp các dịch vụ ta cung cấp cho người dùng hoạt động hiệu quả, bởi chính những lúc hệ thống cao tải bất thường cũng là lúc nhu cầu người dùng là cao nhất.

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. Hiện nay các nhà cung cấp dịch vụ cloud trong nước đều đã có cách giám sát của riêng mình nhưng chưa ai có thể giải quyết tốt ở cả 2 góc nhìn này. Chính vì vậy, em chọn đề tài **“Xây dựng hệ thống giám sát trong triển khai openstack”**

## 2. Mô tả

Hiện tại, mô hình cloud triển khai từ Openstack (opensource platform cloud computing phổ biến nhất), 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.

End user: người dùng cuối, truy cập đến các instance của cloud user để sử dụng các ứng dụng.

Các đối tượng được Openstack phân quyền theo mô hình RBAC :

Cloud provider và cloud user đều là các được coi là người dùng sử dụng openstack:

* 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 trước 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.

Cloud monitor: hiện tại openstack có cung cấp các service giúp monitor các tài nguyên ảo : instance, volume, network,… với các thông số cho ra gồm (ram, cpu, iops,…), chia thành các microservice với mỗi service thực hiện một nhiệm vụ trong luồng quản lý. Tuy nhiên openstack không monitor các hạ tầng vật lý của cloud provider (vd host, switch,…) , cũng không thể lấy thông số các process hay status application trong host hay instance.

Nhiệm vụ của em sẽ là tìm cách cài đặt tổ chức cấu hình các microservice này,kết hợp các giải pháp monitor khác ngoài openstack để thực hiên mục tiêu.

## 3. Mục tiêu

### 3.1.Chức năng

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

**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, instance
* Alarm về 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:**

* Xem được metric về tài nguyên đang sử dụng.
* Có alarm khi có metric vượt ngưỡng. Thiết lập chính sách áp dụng. Ví dụ: tăng số lượng instance khi các máy cao tải (auto scaling)
* Hệ thống monitor riêng mà bản thân người dùng quyết định cài trên instance của mình có thể tương tác với hệ thống Cloud monitor.
* Cloud provider không có quyền cài bất kỳ thành phần monitor nào trên instance người dùng.

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

### 3.2.Hiệu năng

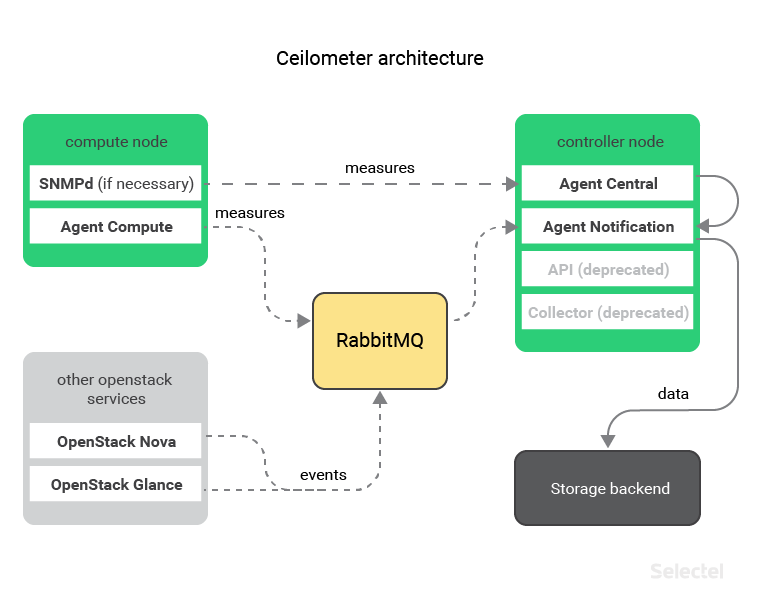
Tìm hiểu yêu cầu hiệu năng của hệ thống monitor:

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

# CHƯƠNG 2: CÔNG NGHỆ SỬ DỤNG

## 3.1. Ceilometer

Ceilometer là một dịch vụ của openstack. Sử dụng thu thập metric của các tài nguyên ảo hệ thống, dựa vào thông tin lấy từ các service khác openstack, cộng với 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:
* <https://docs.openstack.org/ceilometer/pike/admin/telemetry-measurements.html#telemetry-compute-meters>
* 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 Networking**

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| ip.floating | Gauge | ip | ip ID | Pollster | Existence of IP |

* **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 |

* **OpenStack Images**

| **Name** | **Type** | **Unit** | **Resource** | **Origin** | **Note** |
| --- | --- | --- | --- | --- | --- |
| **Meters added in the Mitaka release or earlier** | | | | | |
| image.size | Gauge | image | image ID | Pollster | Size of the uploaded image |
| **Meters removed as of Ocata release** | | | | | |
| Image | Gauge | image | image ID | Pollster | Existence of the image |

Ví dụ: truy vấn thông tin một resource thuộc loại **“image”** được kết quả như bên dưới và đối chiếu với danh sách các metrics liệt kê bên trên. Trong đó:

* Metric **“image.size”** có **origin** là **pollster** 🡪 do **ceilometer-agent-central** thu thập

|  |
| --- |
| root@controller02:/# openstack metric resource show 46d092eb-f3f5-4883-a75a-f3e8e4750784  +---------------------------+-----------------------------------------------------------------------------------  | Field | Value |  +---------------------------+-----------------------------------------------------------------------------------  | created\_by\_project\_id | 2498895c3f484bf89eed68a1f4b6cfbc |  | created\_by\_user\_id | e8b821cce5104ee494a304b1ba39d50e |  | creator | e8b821cce5104ee494a304b1ba39d50e:2498895c3f484bf89eed68a1f4b6cfbc |  | ended\_at | None |  | id | 46d092eb-f3f5-4883-a75a-f3e8e4750784 |  | metrics | image.download: b413c9aa-91e7-417b-af0e-03919ded2e61 |  | | image.serve: 241bec7f-96d5-49f9-a028-3c90be82bcc6 |  | | image.size: f50b074c-dbdb-4382-bd40-1c46bda23c6c |  | original\_resource\_id | 46d092eb-f3f5-4883-a75a-f3e8e4750784 |  | project\_id | 3c8d3df70d2f4c23a1e8783932001080 |  | revision\_end | None |  | revision\_start | 2018-08-20T06:42:46.816820+00:00 |  | started\_at | 2018-08-20T06:42:46.816789+00:00 |  | type | image |  | user\_id | None |  +---------------------------+---------------------------------------------------------- |

* **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 |

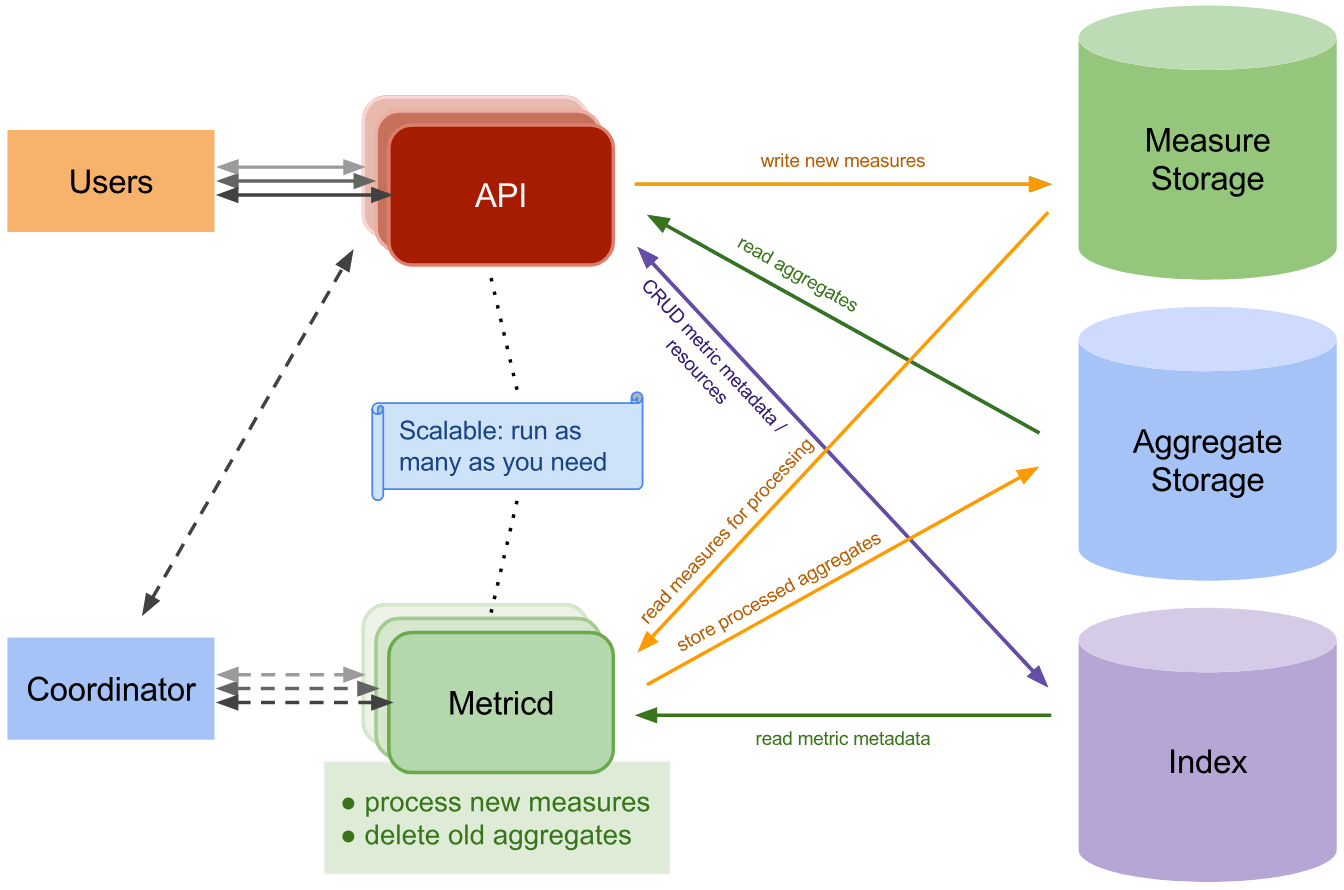
* ***OpenStack Images***

| **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 |

## 3.2. Gnocchi

Gnocchi được dùng để lưu trữ metric từ ceilometter. 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:**

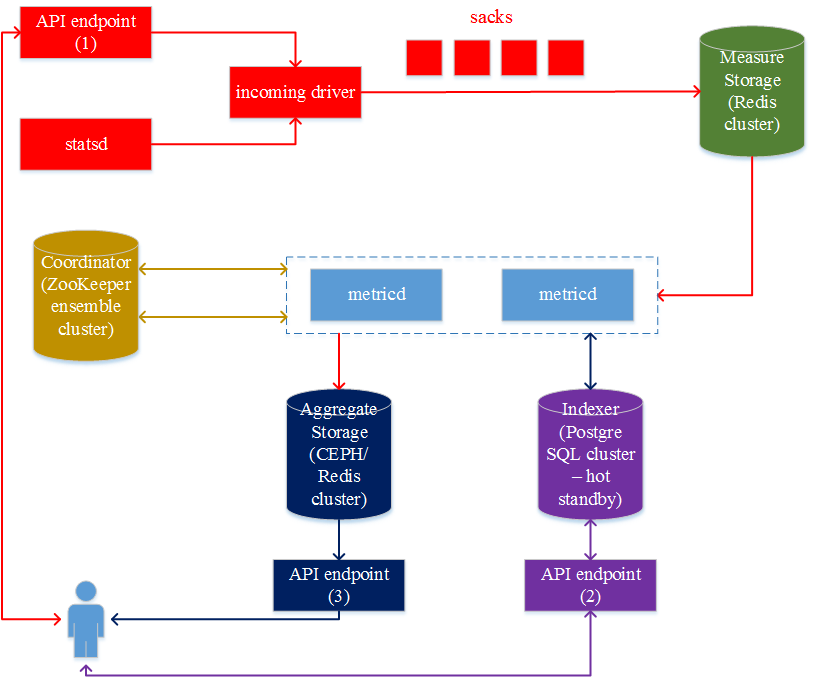
Các backend storage, gnocchi dùng 3 thành phần backend:

* 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:

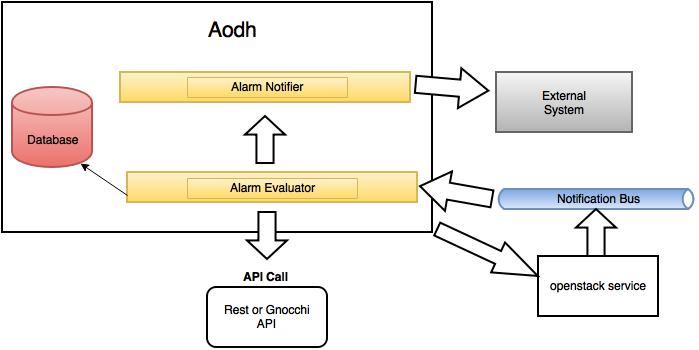
* Các metrics samples (các giá trị measures raw mới thu thập được của các metrics) từ ceilometer gửi tới được xử lý bởi một gnocchi API endpoint hoặc statsd daemon.
* Các entries thu thập được gửi tới incoming driver, tại đây các entries này sẽ được chia nhỏ thành từng phần gọi là sacks rồi lưu vào Incoming 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.

## 3.3. 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

## 3.4. Granfana

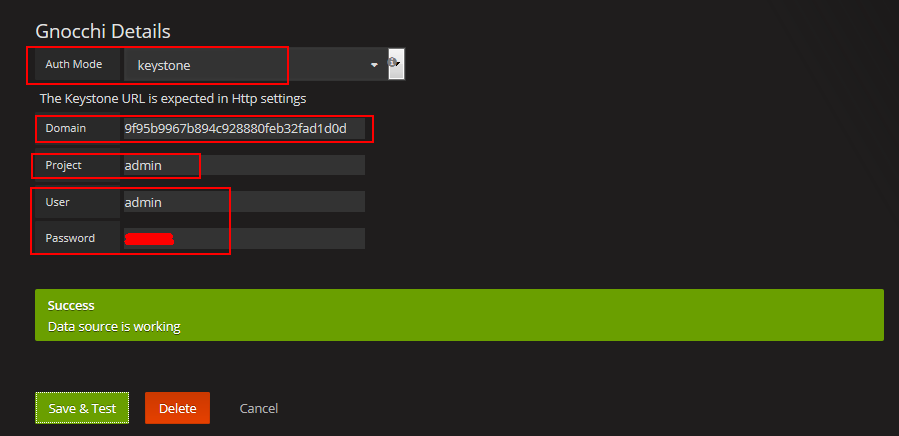
Granfana là một opensource rất nổi tiếng cho việc tạo drashboard và trình diễn dữ liệu thành các bảng biểu.

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.

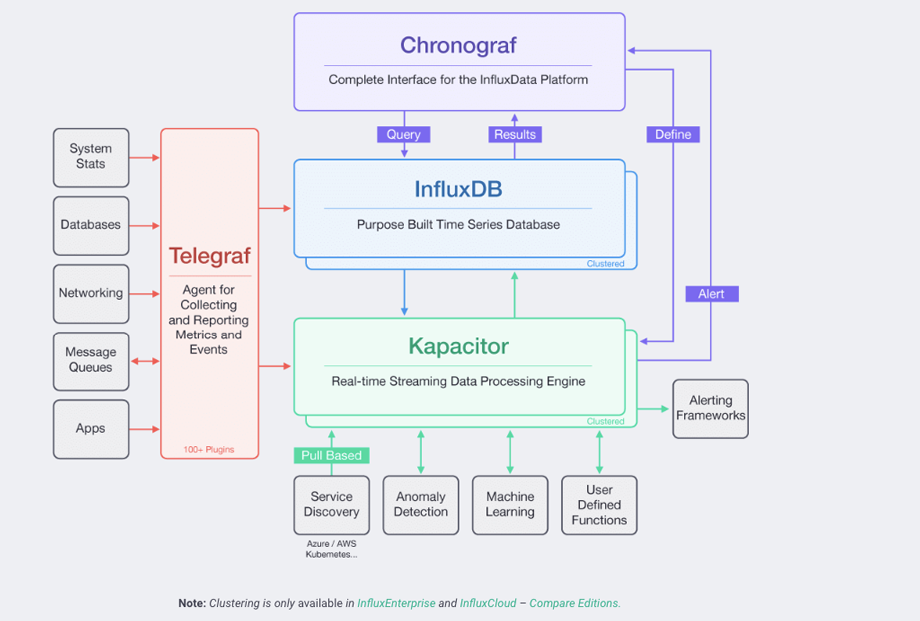


Người dùng sử dụng openstack account kết nối đến gnocchi



Các thông số của tài nguyên instance người dùng

## 3.5. TICK stack



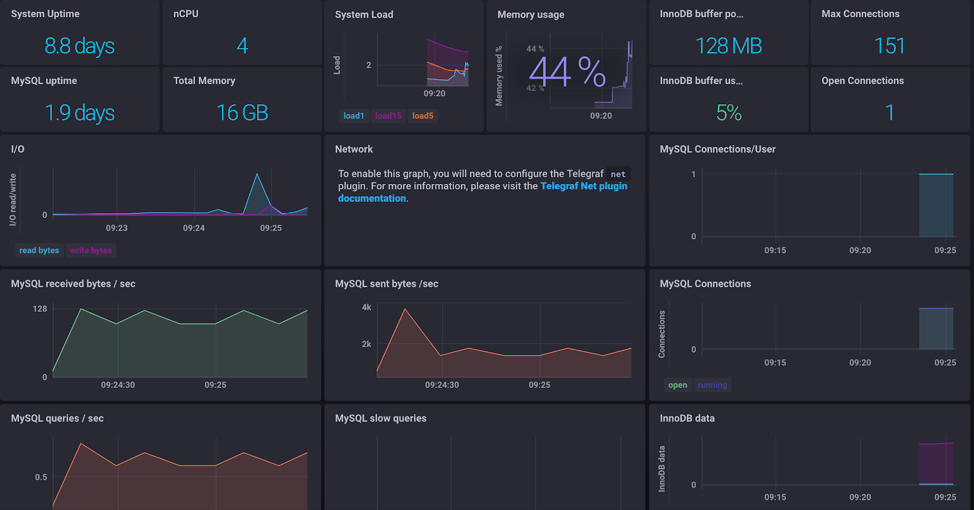
TICK stack is a tập hợp của các sản phầm opensoucre của InfluxData phục vụ monitorinng, là tập hợp của 4 project:

* **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 provides alerting and detects anomalies in time-series data.

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
* 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)
* Ceph storage
* Ceph status
* I/O operations
* I/O bandwidth
* OSD status
* Storage utilization



Hình ảnh về giao diện monitor host vật lý

## 3.4. 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

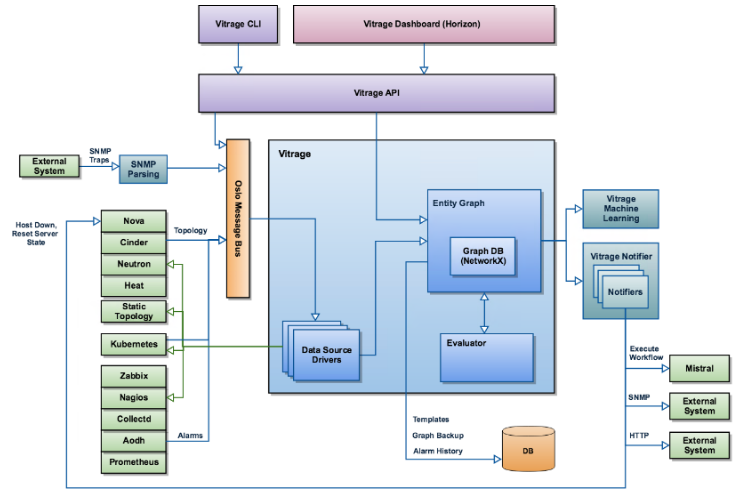
Khả năng của vitrage:

* 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.

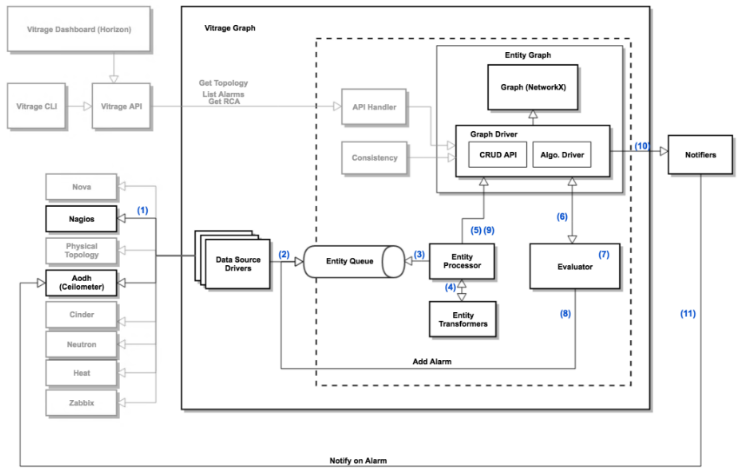


Vitrage bao gồm các thành phần:

* 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

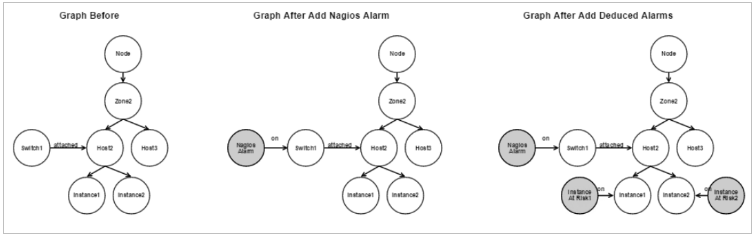
7. Evaluator xem xét các template đã có thì thấy có 1 kịch bản cần áp dụng: bật deduce-alarm error trên mỗi instance gắn với host đó.

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”



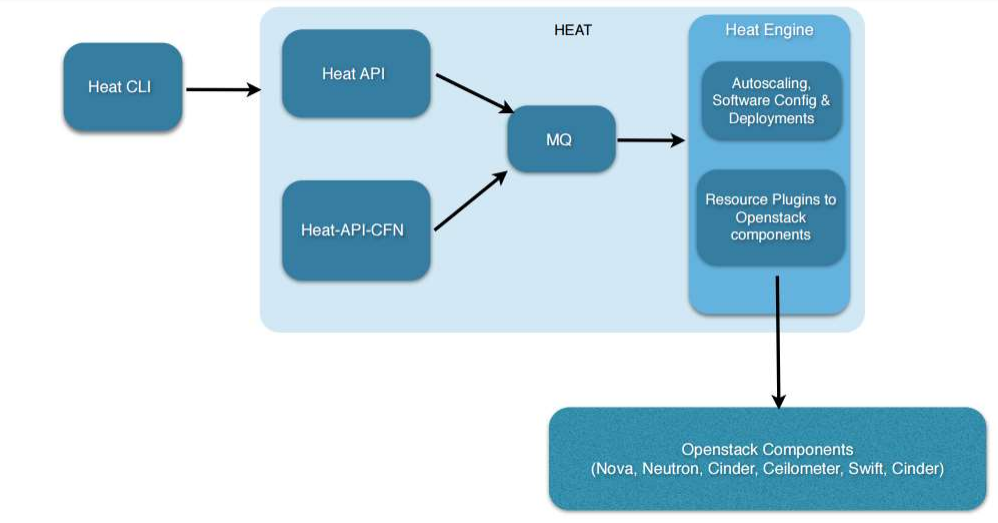
## 3.4. Heat

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 dựa trên orchestration để mô tả ứng dụng cloud bằng cách thực hiện các lời gọi đến OpenStack API để tạo các ứng dụng cloud.

* Template Heat mô tả cơ sở hạ tầng của ứng dụng cloud trong file text mà con người có thể đọc và ghi được, và có thể quản lý bởi version control tool.
* Template chỉ định mỗi quan hệ giữa các tài nguyên (vd: volume được kết nối đến server). Điều này cho phéo Heat gọi đến các OpenStack API để tạo ra tất cả các cơ sở hạ tầng đúng thứ tự để khởi động ứng dụng của người dùng.
* Heat có cung cấp dạng template về auto scaling server, sử dụng resource type “scaling - group”. Tích hợp được với aodh alarm, giúp scaling một stack server khi có alarm đi đến



Kiến trúc của heat:

* Heat-CLI: (command line client) là CLI để giao tiếp với heat-api và chạy lệnh 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.

## 3.5. 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. 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.

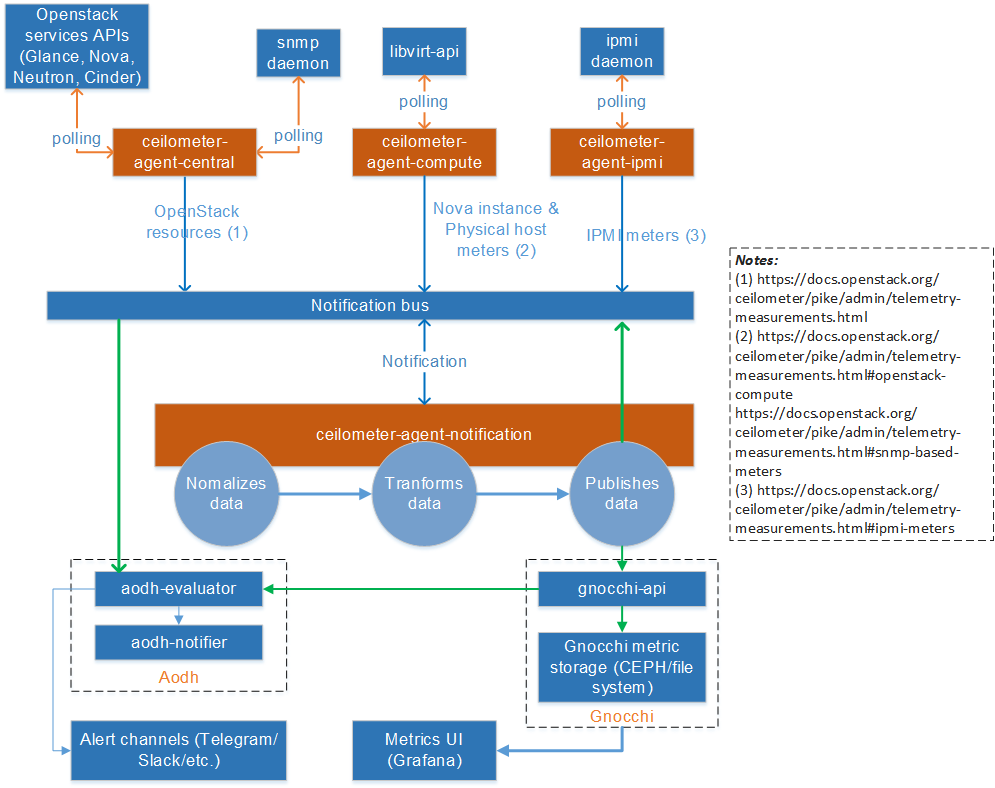
Heat giúp người dùng chạy kịch bản về triển một stack các resource, còn mistral giúp người dùng chạy kịch bản về một stack các hành động.

# CHƯƠNG 3: THIẾT KẾ CHI TIẾT HỆ THỐNG

Bố trí monitor :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Categories** | **Monitor System** | **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 |

Luồng hoạt động:



*Tổ chức các service trong môi trường lab….*



*Tổ chức các service trong môi trường thực tế, đảm bảo HA…*

# CHƯƠNG 4: KỊCH BẢN SỬ DỤNG

## **4.1.Tự động evacuate instance khi host down:**

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 )

**Thực hiện:**

**Cấu hình monitor:**

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

Tạo file tick\_script.txt:

|  |
| --- |
| var period = 120s  var every = 60s  var sys\_data = stream  |from()  .database('telegraf')  .measurement('system')  .groupBy('host')  |window()  .period(period)  .every(every)  sys\_data|deadman(1.0, period)  .message(‘**host offline**’)  .stateChangesOnly()  .exec(‘/usr/bin/python’, ‘/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 vitrage:**

**Cấu hình vitrage**

Sửa file **/etc/vitrage/vitrage.conf:**

|  |
| --- |
| [DEFAULT]  notifiers = nova,mistral  [datasources]  types = **kapacitor**,zabbix,nova.host,nova.instance,nova.zone,static\_physical,aodh,cinder.volume,neutron.network,neutron.port,heat.stack  [kapacitor]  config\_file = /etc/vitrage/kapacitor\_conf.yaml |

Tạo file /etc/vitrage/kapacitor\_conf.yaml , mục đích ánh xạ alarm vitrage 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 vitrage xử lý alarm đến: gọi workflow mistral:

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: zone\_alarm  target: zone  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**  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\_fail\_email\_0  evacuate\_instance:  action: nova.servers\_evacuate server=<% $.instance\_id %>  retry:  delay: 10  count: 10  on-success: wait\_for\_instance\_rebuild  on-error: get\_instance\_status\_when\_fail  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:  action: std.noop  on-complete:  - send\_success\_email: <% $.status\_before = $.status\_after and $.host\_before != $.host\_after %>  - send\_fail\_email\_1: <% $.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

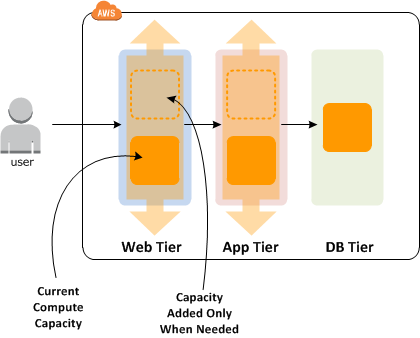
mistral sẽ gọi đến nova api để evacuate instance, send mail thông báo kết quả sau khi quá trình kết thúc

## **4.1.Tự động scale group-instance khi có cảnh báo tài nguyên trên máy ảo**

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.

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
* Scaling rule : ví dụ khi có alarm cpu host cao thì +1 instance, cpu host thấp thì -1 instance

Yêu cầu về ứng dụng của người dùng: database-host và application-host 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 service heat, telemetry

Cơ chế trong openstack:

Telemetry cảnh báo alarm cao tải – thấp tải trên instance 🡪 gọi heat api để scaling up, down như kịch bản của scaling group 🡪 heat thực hiện kịch bản

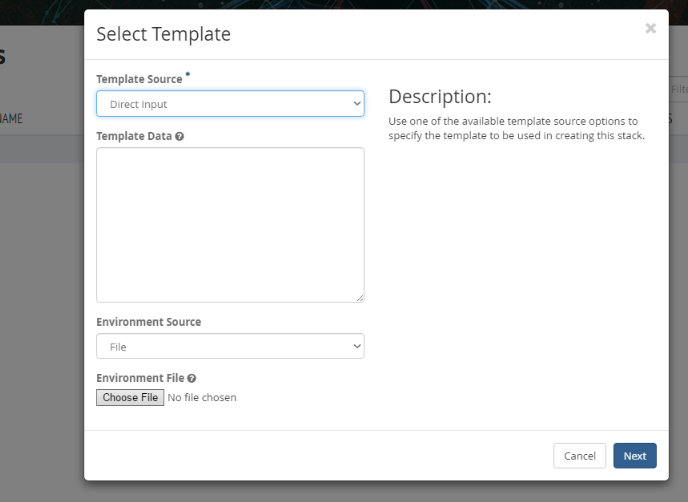
Thực hiện:

Các kịch bản này chính người sử dụng cloud sẽ thực hiện chứ không cần đến cloud admin hỗ trợ:

User tạo template scaling-group của như sau:

|  |
| --- |
| heat\_template\_version: 2014-10-16  description: Example auto scale group, policy and alarm  resources:  scaleup\_group:  type: OS::Heat::AutoScalingGroup  properties:  cooldown: 60  desired\_capacity: 1  max\_size: 3  min\_size: 1  resource:  type: OS::Nova::Server::Cirros    scaleup\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaleup\_group }  cooldown: 60  scaling\_adjustment: 1    scaledown\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaleup\_group }  cooldown: 60  scaling\_adjustment: -1    cpu\_alarm\_high:  type: OS::Ceilometer::Alarm  properties:  meter\_name: cpu\_util  statistic: avg  period: 60  evaluation\_periods: 1  threshold: 50  alarm\_actions:  - {get\_attr: [scaleup\_policy, alarm\_url]}  comparison\_operator: gt    cpu\_alarm\_low:  type: OS::Ceilometer::Alarm  properties:  meter\_name: cpu\_util  statistic: avg  period: 60  evaluation\_periods: 1  threshold: 10  alarm\_actions:  - {get\_attr: [scaledown\_policy, alarm\_url]}  comparison\_operator: lt |

Paste nội dung trên vào giao diện web horizon:



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

|  |
| --- |
| #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 cảnh báo cpu cao và stack sẽ scale thành 2 instance

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

…

# CHƯƠNG 5: TÍNH HIỆU QUẢ VÀ ĐÓNG GÓP CỦA HỆ THỐNG

*Tính toán tài nguyên cần thiết để chạy monitor, dung lượng input, output metric vào ra,…*

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

[3] readhat company “read\_had\_openstack\_platform” 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>