TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────

ĐỒ ÁN

**TỐT NGHIỆP ĐẠI HỌC**

NGÀNH CÔNG NGHỆ THÔNG TIN

**TÊN ĐỀ TÀI:**

**Xây dựng hệ thống giám sát cho đám mây xây dựng trên mã nguồn mở Openstack**

Sinh viên thực hiện : **Ngô Quang Hòa**

Lớp cntt 2.4 - K59

Giáo viên hướng dẫn: **TS.** **Nguyễn Hữu Đức**

HÀ NỘI 06-2019

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Ngô Quang Hòa

Điện thoại liên lạc 0985543162 Email: ngohoa211@gmail.com

Lớp: Cntt2-4 Hệ đào tạo: Kỹ sư

Đồ án tốt nghiệp được thực hiện tại: Trường đại học Bách Khoa Hà Nội

Thời gian làm ĐATN: Từ ngày 20/02/2019 đến 24/ 05/ 2019

2. Mục đích nội dung của ĐATN

Xây dựng hệ thông giám sát cho nền tảng đám mây xây dựng từ openstack. Tìm hiểu tích hợp các công nghệ giám sát, phân tích đưa ra được mô hình, cài đặt, và thử nghiệm trên một hệ thống đám mây cho trước.

3. Các nhiệm vụ cụ thể của ĐATN

- Tìm hiểu các quy trình của việc giám sát hệ thống

- Phân tích các yêu cầu cho việc giám sát trong môi trường đám mây

- Tìm hiểu các công cụ giám sát mã nguồn mở

- Tích hợp và cài đặt các công cụ thành một hệ thống có thể quản lý toàn diện nền tảng đám mây

- Thử nghiệm hệ thống và hướng phát triển

4. Lời cam đoan của sinh viên:

Tôi - *Ngô Quang Hòa* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của TS. Nguyễn Hữu Đức.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm*  Tác giả ĐATN  *Ngô Quang Hòa* |

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm*  Giáo viên hướng dẫn  *TS. Nguyễn Hữu Đức* |

LỜI CẢM ƠN

Lời đầu tiên, tôi xin gửi lời cảm ơn tới trường Đại Học Bách Khoa Hà Nội, viện Công nghệ thông tin và Truyền thông đã tạo cho tôi môi trường học tập quý báu và giúp tôi trải nghiệm nhiều điều cho cuộc sống. Cảm ơn các thầy cô đã truyền thụ cho tri thức, kỹ năng cho những bước trưởng thành suốt năm năm học.

Để hoàn thành đồ án này, tôi xin bày tỏ lòng biết ơn chân thành tới TS. Nguyễn Hữu Đức, đã nhiệt tình hướng dẫn và đã tạo mọi điều kiện cho tôi trong quá trình thực hiện.

Tôi cảm ơn người bạn học, cùng vô cùng thông minh và cởi mở, đã luôn quan tâm chia sẻ trong suốt quá trình học tập.

Cảm ơn đến những người đồng nghiệp trong công việc, tuy khắt khe nhưng luôn cho tôi tìm được sự giúp đỡ khi cần.

MỤC LỤC

[**DANH MỤC HÌNH VẼ** 7](#_Toc9598059)

[**DANH MỤC CÁC BẢNG** 8](#_Toc9598060)

[**CHƯƠNG 1: TỔNG QUAN** 9](#_Toc9598061)

[**1.1 Bài toán giám sát hệ thống** 9](#_Toc9598062)

[**1.2. Khảo sát một số hệ thống giám sát hiện có** 9](#_Toc9598063)

[**1.3. Mục tiêu** 10](#_Toc9598064)

[**Chức năng** 10](#_Toc9598065)

[**Hiệu năng** 10](#_Toc9598066)

[**1.3. Giải pháp** 10](#_Toc9598067)

[**1.4. Nội dung đồ án** 11](#_Toc9598068)

[**CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ** 12](#_Toc9598069)

[**2.1. Openstack** 12](#_Toc9598070)

[**2.1. Ceilometer** 13](#_Toc9598071)

[**2.2. Gnocchi** 17](#_Toc9598072)

[**2.3. Aodh** 18](#_Toc9598073)

[**2.4. Grafana** 20](#_Toc9598074)

[**2.5. TICK stack** 20](#_Toc9598075)

[**2.6. Vitrage** 22](#_Toc9598076)

[**2.7. Heat** 25](#_Toc9598077)

[**2.8. Mistral** 26](#_Toc9598078)

[**CHƯƠNG 3: PHÁT TRIỂN HỆ THỐNG GIÁM SÁT** 28](#_Toc9598079)

[**3.1. Phân tích hệ thống** 28](#_Toc9598080)

[**3.1.1. Các đối tác của hệ thống** 28](#_Toc9598081)

[**3.2.2. Các ca sử dụng của hệ thống** 28](#_Toc9598082)

[**3.2. Thiết kế hệ thống** 34](#_Toc9598083)

[**3.2.1.Bố trí các thành phần giám sát.** 34](#_Toc9598084)

[**3.2.2 Mô tả luồng hoạt động** 35](#_Toc9598085)

[**3.3. Xây dựng hệ thống.** 40](#_Toc9598086)

[**3.3.1. Các yêu cầu cho hệ thống sẵn có** 40](#_Toc9598087)

[**3.3.2. Chi tiết về việc xây dựng hệ thống**. 41](#_Toc9598088)

[**3.3.3. Chi tiết cách thiết lập cơ chế chịu lỗi cho hệ thống** 48](#_Toc9598089)

[**3.3.4. Chi tiết cách thiết lập cơ chế co dãn tài nguyên** 52](#_Toc9598090)

[**CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ** 56](#_Toc9598091)

[**4.1. Giám sát tài nguyên, dịch vụ máy chủ vật lý** 56](#_Toc9598092)

[**4.2. Giám sát tài nguyên ảo hóa** 57](#_Toc9598093)

[**4.3. Kiểm tra khả năng chịu lỗi** 59](#_Toc9598094)

[**4.3. Kiểm tra khả năng tự động co dãn tài nguyên** 59](#_Toc9598095)

[**CHƯƠNG 5: KẾT LUẬN** 60](#_Toc9598096)

[**TÀI LIỆU THAM KHẢO** 61](#_Toc9598097)

# **DANH MỤC HÌNH VẼ**

[Hình 2.1.Kiến trúc ceilometer 13](#_Toc9581116)

[Hình 2.2.Kiến trúc của gnocchi 17](#_Toc9581117)

[Hình 2.3.Kiến trúc của aodh 19](#_Toc9581118)

[Hình 2.4. Kiến trúc của thành phần TICK-stack 21](#_Toc9581119)

[Hình 2.5. Kiến trúc của thành phần vitrage 23](#_Toc9581120)

[Hình 2.6. Luồng hoạt động cho thành phần vitrage 24](#_Toc9581121)

[Hình 2.7.Kiến trúc của thành phần heat 26](#_Toc9581122)

[Hình 3.1. Biểu đồ tổng quan của hệ thống giám sát 29](#_Toc9581126)

[Hình 3.2. Biểu đồ ca sử dụng cho chức năng theo dõi thông tin máy vật lý 29](#_Toc9581127)

[Hình 3.3.Biểu đồ ca sử dụng cho chức năng quản lý cảnh báo máy vật lý 30](#_Toc9581128)

[Hình 3.4.Biểu đồ ca sử dụng cho chức năng quản lý cơ chế chịu lỗi 31](#_Toc9581129)

[Hình 3.5. Cơ chế chịu lỗi trong hệ thống đám mây 32](#_Toc9581130)

[Hình 3.6. Biểu đồ mô tả ca sử dụng người dùng theo dõi tài nguyên máy ảo 32](#_Toc9581131)

[Hình 3.7.Mô tả cơ chế co dãn tài nguyên 33](#_Toc9581132)

[Hình 3.8. Biểu đồ ca sử dụng quản lý cơ chế co dãn tài nguyên 34](#_Toc9581133)

[Hình 3.9. Sơ đồ phân bố các thành phần giám sát 35](#_Toc9581134)

[Hình 3.10. Biểu đồ trình tự mô tả người dùng xem thông tin máy vật lý 36](#_Toc9581135)

[Hình 3.11.Biểu đồ trình tự mô tả luồng xử lý dữ liệu giám sát máy vật lý 36](#_Toc9581136)

[Hình 3.12. Biểu đồ trình tự mô tả người dùng đặt cảnh báo cho tài nguyên vật lý 37](#_Toc9581137)

[Hình 3.13.Biểu đồ trình tự mô tả quá trình tự động vượt qua lỗi của hệ thống 38](#_Toc9581138)

[Hình 3.14. Biểu đồ trình tự mô tả hệ thống thu thập dữ liệu tài nguyên ảo hóa 39](#_Toc9581139)

[Hình 3.15. Biểu đồ trình tự mô tả chức năng người dùng đặt cảnh báo 40](#_Toc9581140)

[Hình 3.16. Cài đặt kết nối chronograf và influxdb 44](#_Toc9581141)

[Hình 4.1 Người dùng quản trị xem danh sách máy vật lý 56](#_Toc9581142)

[Hình 4.2 Người dùng quản trị theo dõi tài nguyên vật lý 56](#_Toc9581143)

[Hình 4.3. Người dùng đặt cảnh báo trạng thái dịch vụ openstack 57](#_Toc9581144)

[Hình 4.4.Người dùng quản trị cấu hình kênh thông báo 57](#_Toc9581145)

[Hình 4.5.Người dùng đăng nhập giám sát máy ảo 58](#_Toc9581146)

[Hình 4.6.Người dùng xem tài nguyên tất cả các máy ảo 58](#_Toc9581147)

# **DANH MỤC CÁC BẢNG**

[Bảng 2. 1. Danh sách các dữ liệu giám sát thu thập bới ceilometer 16](#_Toc9581664)

[Bảng 3. 1. Yêu cầu cho hệ thống đám mây sẵn có 41](#_Toc9581654)

[Bảng 3. 2. Bố trí các thành phần của dịch vụ giám sát 41](#_Toc9581655)

[Bảng 3. 3. Tài nguyên ước tính sử dụng cho hệ thống giám sát 41](#_Toc9581656)

# **CHƯƠNG 1: TỔNG QUAN**

## **1.1 Bài toán giám sát hệ thống**

Xây dựng hệ thống giám sát 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 cho công việc vận hành hệ thống gồm:

* Cho biết được trạng thái hoạt động của hệ thống về mặt tài nguyên (ram, cpu, network, ổ cứng) và về các dịch vụ đang được triển khai. 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 cho quá trình dò tìm và xử lý lỗi . Tiến tới có thể điều khiển hệ thống tự động vận hành và vượt qua một số sự cố nhất định.

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 điện toán đám mây, bài toán này lại trở nên rất đặc thù: người sử dụng hệ thống đám mây bản thân cũng là một người quản trị các máy chủ ảo của mình . Hệ thống giám sát phải có thể hỗ trợ được cả 2 góc nhìn: từ phía nhà cung cấp dịch vụ đám mây, và phía người sử dụng đám mây. Phía nhà cung cấp dịch vụ đám mây, mối quan tâm chủ yếu là các tài nguyên vật lý và các tiến trình chạy trên đó . Phía người sử dụng dịch vụ đám mây, mối quan tâm lại là tới phần tài nguyên ảo hóa mà mình được nhận. Đây là bài toán rất thực tế hiện nay, đòi hỏi các kỹ năng về quản trị và xây dựng hệ thống, phân tích và tích hợp các công nghệ mà sẽ được chọn để giải quyết trong đồ án này.

## **1.2. Khảo sát một số hệ thống giám sát hiện có**

Với các nhà cung cấp dịch vụ đám mây hàng đầu hiện nay, hỗ trợ cho góc nhìn phía người dùng hệ thống đám mây, các chức năng được cung cấp như sau:

* Đám mây của các nhà cung cấp VCCorp, Viettel IDC đều có cho người dùng xem giao diện giám sát tài nguyên máy ảo. Tuy nhiên dừng lại ở việc cho người dùng theo giõi các biểu đồ tài nguyên đang sử dụng: ram, cpu, iops.
* Nổi bật hơn là đám mây của VNG cung cấp dịch vụ giám sát ở trên, và thêm chức năng tự động co dãn tài nguyên, cân bằng tải để đối phó với các sự kiện cao tải trên máy ảo
* Đám mây của Amazone xây dựng dịch vụ giám sát thành một cả một nền tảng cloudwatch : dịch vụ toàn diện về giám sát và co giãn tài nguyên, hoạt động vô cùng ổn định. Tuy nhiên các để làm điều này, các máy ảo đều có cài tác tử của amazone là "OpsWorks Stacks Agent", gây một số vấn đề liên quan đến bảo mật thông tin.

Về giải pháp giám sát các tài nguyên vật lý, đây đã là bài toán có từ rất lâu và các công nghệ giám sát cũng ngày một phát triển. Nhìn chung các nhà cung cấp đám mây ít công khai giải pháp giám sát của mình. Tuy nhiên với việc các nhà cung cấp thường cam kết cho các máy ảo có dịch vụ là 99.9%, có thể thấy vấn để giám sát hạ tầng vật lý của các hãng là rất thách thức.

## **1.3. Mục tiêu**

Hiện nay có nhiều công nghệ để ảo hóa tài nguyên nhằm xây dựng hệ thống đám mây, tuy nhiên nổi bật và phổ biến nhất là mã nguồn mở Openstack. Sự phát triển nó có được là nhờ vào tính mở của cộng đồng, dễ mở rộng, không yêu cầu đặc biệt về phần cứng và tích hợp tốt với các công nghệ ảo hóa sẵn các sản phầm của bên thứ ba. Đây cũng là môi trường đám mây nằm trong phạm vi của đồ án.

Đồ án **“Xây dựng hệ thống giám sát cho các đám mây xây dựng trên mã nguồn mở openstack”** sẽ giải quyết các vấn đề của bài toán giám sát đám mây với các yêu cầu chức năng và hiệu năng như sau:

### **Chức năng**

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

**Nhà cung cấp dịch vụ đám mây:**

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

**Người sử dụng dịch vụ đám mây:**

* Giám sát tài nguyên: CPU, Memory, Disk, Network bandwith/throughput của máy ảo
* Thiết lập cảnh báo và có chính sách áp dụng. Ví dụ: tăng số lượng máy ảo khi các máy cao tải (auto scaling)
* Nhà cung cấp dịch vụ đám mây không có quyền cài bất kỳ thành phần giám sát nào trực tiếp trên máy ảo người dùng.

### **Hiệu năng**

* Tìm hiểu yêu cầu hiệu năng của hệ thống giám sát:
* Đề xuất RAM, CPU cần dùng cho giám sát, dung lượng disk để lưu dữ liệu giám sát.

## **1.3. Giải pháp**

Hiện nay các sản phẩm sử dụng giám sát hệ thống đã rất phong phú, các sản phẩm mã nguồn mở 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 ca sử dụng cho việc giám sát thì cũng đều đầy đủ và không nhiều khác biệt nên không sản phẩm nào thực sự ưu việt hơn tất cả. TICK-stack được lựa chọn làm giải pháp giám sát cho các thành phần vật lý, ý kiến chủ quan là do.

* Zabbix, nagios sử dụng thành phần lưu trữ là cơ sở dữ liệu SQL. Dữ liệu giám sát của máy -các điểm dữ liệu tại mỗi thời điểm có đặc tính là kích thước 1 bản ghi rất nhỏ, đổ về liên tục từng mili giây. Về lâu dài sẽ làm quá tải trên cơ sở dữ liệu.
* Graphite, Prometheus, TICK: sử dụng cơ sở dữ liệu thời gian thực(time-series database) - loại cơ sở thiết kế riêng cho việc giám sát. TICK cảm quan khi sử dụng thấy dễ dàng và có nhiều hỗ trợ từ cộng đồng.

Công nghệ để quản lý các tài nguyên ảo hóa sẽ sử dụng là Telemetry, bộ công cụ dùng để giám sát cho hệ thống đám mây nền openstack. Bao gồm các nhóm dịch vụ: ceilometer, gnocchi, aodh. Telemetry trong đồ án dùng kết hợp các dịch vụ khác từ openstack như heat, vitrage, mistral cho việc tự động phản ứng với những sự cố.

## **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 kiểm định đánh giá cho các chức năng hệ thống.
* Chương 5 Kết luận và hướng phát triển

# **CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ**

*Chương này tập trung giới thiệu về các công nghệ sẽ sử dụng trong hệ thống: các tính năng, kiến trúc, vai trò mà nó được tích hợp trong hệ thống.*

## **2.1. Openstack**

OpenStack là phần mềm mã nguồn mở sử dụng để triển khai đám mây.

Openstack ban đầu do NASA và Rackspace khởi xướng, giới thiệu lần đầu năm 2010. Hiện tại đang được phát triển bởi cộng đồng với sự tham gia của nhiều hãng lớn: AT&T, Ubuntu, IBM, RedHat, SUSE, Mirantis, vv...

Tóm lược đặc điểm của openstack:

* Thiết kế theo hướng module, OpenStack là một dự án lớn là sự kết hợp của các dự án thành phần: nova, swift, neutron, glance,vv...
* Hoạt động theo hướng mở: công khai lộ trình phát triển, công khai mã nguồn, thiết kế, cộng đồng phát triển là cộng đồng mã nguồn mở.
* Chu kì 6 tháng phát hành phiên bản mới theo thứ tự bảng chữ cái: A, B, C...(Austin, Bexar, Cactus, etc.).
* Phần lớn mã nguồn của OpenStack là python

Openstack có các dự án nền, cốt lõi như sau:

**NOVA** - Dịch vụ Compute : Quản lí các máy ảo trong môi trường OpenStack, chịu trách nhiệm khởi tạo, lập lịch, ngừng hoạt động của các máy ảo theo yêu cầu.

**NEUTRON** - Dịch vụ Networking : Cung cấp kết nối mạng như một dịch vụ (Network-Connectivity-as-a-Service) cho các máy ảo của OpenStack.

**CINDER** - Dịch vụ Block Storage : Cung cấp các khối lưu trữ bền vững (volume) để chạy các máy ảo .

**KEYSTONE** - Dịch vụ Identity : Cung cấp dịch vụ xác thực và ủy quyền cho các dịch vụ khác của OpenStack.

**GLANCE** - Dịch vụ Image : Lưu trữ và truy xuất các đĩa cài đặt của các máy ảo người dùng và cho các dịch vụ khác.

**HORIZON** - Dashboard : Cung cấp giao diện nền web cho người dùng cuối và người quản trị đảm mây để tương tác với các dịch vụ khác của OpenStack, ví dụ như vận hành các máy ảo, cấp phát địa chỉ IP và kiểm soát cấu hình truy cập các dịch vụ.

Một số các dịch vụ khác cần cài đặt trong hệ thống OpenStack (không thuộc dự án nhưng là thành phần cần thiết của hệ thống), như:

* MySQL (MariaDB) lưu trữ dữ liệu về hoạt động của người dùng, trạng thái của các máy ảo, hệ thống mạng, images,… ;
* Rabbit MQ - Message Broker sử dụng để lưu trữ, trao đổi các bản tin giữa các tiến trình trong hệ thống dùng giao thức AMQP (Advanced Message Queuing Protocol)…

## **2.1. Ceilometer**

Ceilometer là một dịch vụ của openstack. Sử dụng thu thập dữ liệu giám sát (metric) của các tài nguyên ảo hóa. Hoạt động nhờ vào khả năng lấy thông tin từ các dịch vụ khác từ openstack cộng với việc truy vấn libvirt api của thành phần ảo hóa. Kiến trúc thành phần của ceilometer được minh họa như hình 2.1



Hình 2.1.Kiến trúc ceilometer

Ceilometer bao gồm hai loại tác tử (agent):

* **Tác tử thu thập thông tin (notification agent):** lấy các thông tin từ hàng đợi thông điệp (rabbitmq) và chuyển đổi thành mẫu dữ liệu theo định dạng của ceilometer.
* **Tác tử truy vấn thông tin (polling agent):** thực hiện gọi đến OpenStack APIs hoặc một số công cụ khác (ipmid, snmpq) để thu thập thông tin theo chu kỳ đặt trước. Các tác tử thuộc loại này: **tác tử thu thập tài nguyên tính toán (agent-compute), tác tử thu thập dữ liệu snmp (agent-ipmi), Tác tử xử lý thông tin trung tâm (agent-central).**

Chi tiết các tác tử được trình bày sau đây.

* **Tác tử thu thập tài nguyên tính toán:** chạy trên các nốt dùng ảo hóa tài nguyên (compute node) thu thập thông tin giám giát của các máy ảo bằng cách gọi đến libvirt-api, dữ liệu này chuyển tới hàng đợi thông điệp để tác tử quản lý thông tin lấy về.
* Danh sách chi tiết các dữ liệu giám sát thu thập bởi tác tử này có thể xem tại [1], tuy nhiên các thông số quan trọng nhất bao gồm

| **Tên** | **Đơn vị** | **Loại Tài nguyên** | **Giải thích** |
| --- | --- | --- | --- |
| memory | MB | máy ảo | Lượng RAM cấp cho máy ảo |
| memory.usage | MB | máy ảo | Lượng RAM máy ảo được sử dụng |
| memory.resident | MB | máy ảo | Lượng RAM dùng của máy vật lý của máy ảo |
| cpu | ns | máy ảo | Thời gian sử dụng cpu |
| cpu.delta | ns | máy ảo | Khoảng cách với lần dùng cpu gần nhất |
| cpu\_util | % | máy ảo | Phần trăm cpu sử dụng |
| vcpus | vcpu | máy ảo | Số cpu ảo máy ảo được cấp |
| disk.read.requests | request | máy ảo | số lượng yêu cầu đọc |
| disk.read.requests.rate | request/s | máy ảo | Lượng yêu cầu đọc/giây(IPS) |
| disk.write.requests | request | máy ảo | Số lượng yêu cầu ghi |
| disk.write.requests.rate | request/s | máy ảo | Lượng yêu cầu ghi/giây |
| disk.read.bytes | B | máy ảo | số byte đã đọc |
| disk.read.bytes.rate | B/s | máy ảo | lượng byte đã đọc/giây |
| disk.write.bytes | B | máy ảo | số byte đã ghi |
| disk.write.bytes.rate | B/s | máy ảo | lượng byte đã ghi/giây |
| disk.device.read.requests | request | ổ cứng máy ảo | Số yêu cầu đọc của mỗi ổ cứng máy ảo |
| disk.device.read.requests.rate | request/s | ổ cứng máy ảo | lượng yêu cầu đọc của mỗi ổ cứng máy ảo/giây |
| disk.device.write.requests | request | ổ cứng máy ảo | Số yêu cầu ghi của mỗi ổ cứng máy ảo |
| disk.device.write.requests.rate | request/s | ổ cứng máy ảo | lượng yêu cầu ghi của mỗi ổ cứng máy ảo/giây |
| disk.device.read.bytes | B | ổ cứng máy ảo | số byte đã đọc của mỗi ổ cứng máy ảo |
| disk.device.read.bytes .rate | B/s | ổ cứng máy ảo | số byte đã đọc trung bình/giây mỗi ổ cứng |
| disk.device.write.bytes | B | ổ cứng máy ảo | số byte đã ghi của mỗi ổ cứng máy ảo |
| disk.device.write.bytes .rate | B/s | ổ cứng máy ảo | Lượng byte ghi trung bình/giây |
| disk.root.size | GB | máy ảo | Kích cỡ ổ cứng root |
| disk.ephemeral.size | GB | máy ảo | Kích cở ổ cứng máy ảo chiếm trên máy vật lý |
| disk.latency | ms | máy ảo | Độ trễ đọc ghi |
| disk.iops | count/s | máy ảo | Tổng yêu cầu đọc-ghi (iops\_ |
| disk.device.latency | ms | ổ cứng máy ảo | Độ trễ đọc ghi của mỗi ổ đĩa ảo |
| disk.device.iops | count/s | ổ cứng máy ảo | Tổng yêu cầu đọc ghi trên mỗi ổ đĩa ảo |
| disk.capacity | B | máy ảo | Tổng dung lượng ổ đĩa có thể dùng mà máy ảo thấy |
| disk.allocation | B | máy ảo | Tổng dung lượng ổ đĩa để dành cho máy ảo trên máy vật lý |
| disk.usage | B | máy ảo | Dung lượng ổ đĩa đã dùng |
| disk.device.capacity | B | ổ cứng máy ảo | lượng mỗi ổ đĩa có thể dùng mà máy ảo thấy |
| disk.device.allocation | B | ổ cứng máy ảo | dung lượng mỗi ổ đĩa có thể dùng mà máy ảo thấy |
| disk.device.usage | B | ổ cứng máy ảo | dung lượng thực sự ổ đĩa chiếm dụng trên máy vật lý |
| network.incoming.bytes | B | card mạng ảo | số lượng byte đi đến |
| network.incoming.bytes.rate | B/s | card mạng ảo | Trung bình lượng dữ liệu đến/giây |
| network.outgoing.bytes | B | card mạng ảo | Lượng byte đi ra |
| network.outgoing.bytes.rate | B/s | card mạng ảo | Trung bình số byte đi ra/giây |
| network.incoming.packets | packet | card mạng ảo | Lượng packet đi đến |
| network.incoming.packets.rate | packet/s | card mạng ảo | Trung bình lượng packet đi đến/giây |
| network.outgoing.packets | packet | card mạng ảo | lượng packet đi ra |
| network.outgoing.packets.rate | packet/s | card mạng ảo | Trung bình số packets/giây |
| cpu\_l3\_cache | B | máy ảo | Lượng cpu dùng làm bộ nhớ đệm |
| memory.bandwidth.total | B/s | máy ảo | Thông lượng dữ liệu vào ra của RAM tính cả cache |
| memory.bandwidth.local | B/s | máy ảo | Thông lượng dữ liệu vào ra của RAM |
| perf.cache.references | count | máy ảo | số lượng cache hit |
| perf.cache.misses | count | máy ảo | số lượng cache miss |

Bảng 2. 1. Danh sách các dữ liệu giám sát thu thập bới ceilometer

* **Tác tử xử lý trung tâm (agent-central)**
* ***Truy vấn đến dịch vụ snmp để thu thập thông tin cho quản lý tài nguyên vật lý (cpu load, cpu util, memory usage, memory swap, network throughput,…). Để thu thập dữ liệu này các máy vật lý cần mở giao thức snmp kém an toàn, đồng thời Ceilometer trong thiết kế hệ thống của đồ án được tập trung vào nhiệm vụ giám sát máy ảo nên không cần thiết dùng chức năng này***
* ***Gọi đến REST APIs của các dịch vụ OpenStack để thu thập thông tin giám sát của các tài nguyên mà các dịch vụ này quản lý, như:***
* ***OpenStack Networking cho biết số route, số port, số ip của mỗi máy ảo, Openstack Image cho biết loại hệ điều hành, kích cỡ file iso khỏi tạo,…***
* **Tác tử quản lý thông tin:**

Đọc dữ liệu trong hàng đợi thông điệp rabbitmq . Thu thập được những thông tin bao gồm:

* ***Các sự kiện, các thao tác xử lý của các dịch vụ OpenStack khác***
* ***Các dữ liệu từ các tác tử khác của ceilometer***

Tác tử nàythực hiện xử lý các sự kiện nhận được và đẩy ra các đích tương ứng. Trong đó:

* Một số sự kiện quan trong được đẩy thẳng sang khối cảnh báo, ví dụ như sự kiện máy ảo thất bại cho người dùng.
* Các dữ liệu giám sát khác được đẩy sang thành phần lưu trữ thông tin giám sát -Gnocchi.

## **2.2. Gnocchi**

Gnocchi được dùng để lưu trữ dữ liệu giám sát (metric) thu thập từ ceilometter , tức nó là kho lưu trữ dữ liệu về tài nguyên ảo hóa. Đây là một cơ sở dữ liệu chuỗi thời gian (time-series database) mã nguồn mở, xuất phát cũng là một dự án của openstack sau tách ra.

Cơ sở dữ liệu chuỗi thời gian (time series database) là kiểu database được tối ưu hóa để xử lý dữ liệu từ các phép đo liên tiếp trong thời gian thực, có chỉ mục được lập theo thời gian.



Hình 2. 2.Kiến trúc của gnocchi

Gnocchi gồm hai khối chính:

* **Khối điều khiển:**
* **gnocchi-api:** tiếp nhận yêu cầu xử lý truy vấn đọc ghi dữ liệu giám sát
* **Khu xử lý trung tâm (gnocchi metricd):** tiến trìnhthực hiện các tác vụ tính toán, thống kê,… các dữ liệu này và điều khiển lưu vào khối lưu trữ
* **Khối lưu trữ:**

gnocchi dùng 3 thành phần lưu trữ:

* **Kho index**:  dùng lưu trữ index và dữ liệu mô tả cho các bản ghi: định nghĩa loại, thuộc tính về nguồn dữ liệu, luật về xử lý với dữ liệu đó (archive policies). .Một luật xử lý này định nghĩa việc sử dụng các phép toán: mean, min, max, count,… để nhóm các dữ liệu này về một khoảng thời gian nhất định - 1 phút, 1 ngày, 7 ngày

Tùy chọn lưu trữ: MySQL, PostgreSQL.

* **Kho dữ liệu thô**: lưu trữ các dữ liệu mới gửi tới (mỗi điểm dữ liệu bao gồm cặp khóa-giá trinh: 1 nhãn thời gian và 1 giá trị tương đây là loại dữ liệu thô chưa xử lý). Tùy chọn lưu trữ: Redis, CEPH, filesystem, S3, Swift.
* **Kho dữ liệu hậu xử lý**: lưu trữ các giá trị có ý nghĩa, sau tính toán, dữ liệu thô.

Tùy chọn lưu trữ : Redis, CEPH, filesystem, S3, Swift.

Một số thành phần khác:

* Thành phần điều phối cho các tiến trình xử lý dữ liệu
* Tùy chọn: **redis, zookeeper.**
* Để đảm bảo tính nhất quán giữa các api và các luồng của thành phần xử lý gnocchi khi cùng đọc và ghi vào, Gnocchi cung cấp carbonara driver hỗ trợ việc khóa timeseries mà đang được cập nhật trên toàn bộ các tiến trình tại một giá trị nhất định.

## **2.3. Aodh**

Aodh là dịch vụ phát cảnh báo của openstack, kết nối với gnocchi để đưa ra cảnh báo khi dữ liệu vượt quá hoặc thấp hơn ngưỡng định trước.

Kiến trúc của thành phần aodh được mô tả như hình 2.3



Hình 2. 3.Kiến trúc của aodh

Kiến trúc aodh gồm 3 thành phần:

* **Api:** Cung cấp api quản lý các cảnh báo cho người dùng, Một cảnh báo chuyển giữa 3 trạng thái: ok, alarm hoặc insufficient data**.**
* **Khối suy diễn (aodh-evaluator) :**Truy vấn dữ liệu giám sát , tính toán và đánh giá cảnh báo dựa trên các dữ liệu quản trị và các luật cảnh báo với dữ liệu đó. truy vấn Gnocchi api để lấy các dữ liệu. Để thành phần này có thể chạy trên nhiều nốt controller, phải cấu hình thêm thành phần điều phối (như ZooKeeper) để đồng bộ, tránh xung đột.
* **Khối thông báo (aodh-notifier):** Thực hiện các hành động định nghĩa trong các alarms khi có sự thay đổi trạng thái của mỗi cảnh báo

Kênh gửi cảnh báo 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 queue: Mục đích cho các thành phần khác đọc và xử lý . vd: virtrage, zaquar

## **2.4. Grafana**

Grafana là một phần mềm mã nguồn mở rất nổi tiếng cho việc tạo ra giao diện giám sát hệ thống

Các đặc điểm, tính năng chính:

* Grafana là phần mềm mã nguồn mở được phát triển bởi Grafana Labs.
* Grafana cho phép hiển thị thông tin hữu ích một cách rõ ràng, súc tích dưới

dạng đồ thị, bảng,…

* Grafana có thư viện giao diện phong phú tương thích với nguồn dữ liệu khác nhau. mysql, InfluxDB, graphite và gnocchi. Cách thức truy vấn thông tin linh hoạt
* Grafana cung cấp giao diện ứng dụng web dễ nhìn bằng ngôn ngữ lập trình JavaScript và Golang,
* Grafana cũng cung cấp khả năng tính toán và cảnh báo đến người quản trị thông qua các dịch vụ sms, mail,….

Vai trò của Grafana trong mô hình:

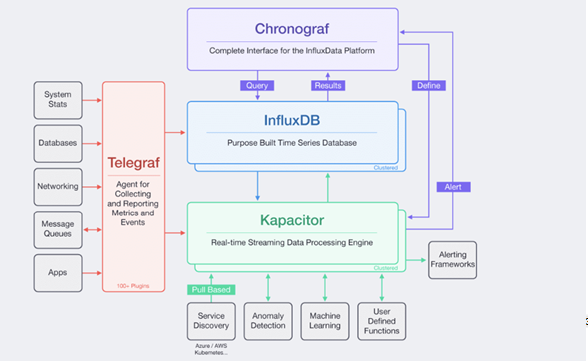
* Grafana gửi yêu cầu vào gnocchi api để truy xuất ra những thông tin về máy ảo để thể hiện trên bảng điều khiển dưới dạng biểu đồ , bảng số liệu…. trên giao diện web, giúp người dùng cái nhìn trực quan về các số liệu.
* Grafana hỗ trợ xác thực keystone để đảm bảo tính bảo mật khi request vào gnocchi.

## **2.5. TICK stack**

TICK stack is a tập hợp của các sản phầm opensoucre của InfluxData phục vụ giám sát hệ thống máy chủ, là tập hợp của 4 project:

* **T**elegraf : vai trò thu thập time-series data từ nhiều nguồn khác nhau: ví dụ như tài nguyên server, các cuộc gọi API, dịch vụ hệ thống , truy vấn DB, SMNP.
* **I**nfluxDB : vai lưu trữ time-series data.
* **C**hronograf : vai trò trình diễn, 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

Kiến trúc của TICK- stack được mô tả như hình 2.4, tham khảo từ nguồn [4]



Hình 2. 4. Kiến trúc của thành phần TICK-stack

Vai trò của TICK trong mô hình:

TICK để giám sát cho các máy chủ vật lý, phục vụ nhu cầu của nhà cung cấp dịch vụ đám mây. Từ TICK có thể biết được trạng thái tài nguyên và dịch vụ của các máy chủ. Đặc biêt đề đảm bảo hoạt động của openstack ta cần chú ý đến các dịch vụ sau:

* Dịch vụ thuộc dự án 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 sử dụng lưu thông điệp bởi RabbitMQ
* Số lượng file sử dụng như các network sockets bởi RabbitMQ
* Dung lượng ổ cứng chiếm bởi RabbitMQ trên mỗi nốt
* Dung lượng RAM sử dụng bởi RabbitMQ trên mỗi nốt
* Memcached:
* Số bytes đã được dùng để cache
* Số bytes tối đa được cho phép trong cache
* Số kết nối
* 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 máy chủ)
* Written (số lượng bytes/s gửi từ máy chủ ra bên ngoài)

## **2.6. Vitrage**

Vitrage - (OpenStack root cause analysis) là dịch vụ giúp tổ chức, phân tích, các cảnh báo, sự kiện xảy đến với hệ thống, nhằm làm rõ được những cảnh báo này có nguồn gốc từ đâu và có thể gây ảnh hưởng thế nào.

Vai trò của vitrage trong mô hình :

* Đối với nhà cung cấp dịch vụ đám mây: với trường hợp hệ thống gặp sự cố, không chỉ một mà rất nhiều cảnh báo cùng đưa ra. Vitrage sẽ hỗ trợ được người quản trị trong dò tìm ra đâu là cảnh báo nguồn. Nó cũng có khả năng gọi luồng hoạt động đã xác định trước (gọi cho dịch vụ mistral ) để giúp hệ thống vượt qua lỗi
* Đối với người sử dụng dịch vụ đám mây: cho cái nhìn tổng quan về tất cả các tài nguyên được cung cấp, cùng hiện các cảnh báo trên hệ thống khi có.

Khả năng của vitrage:

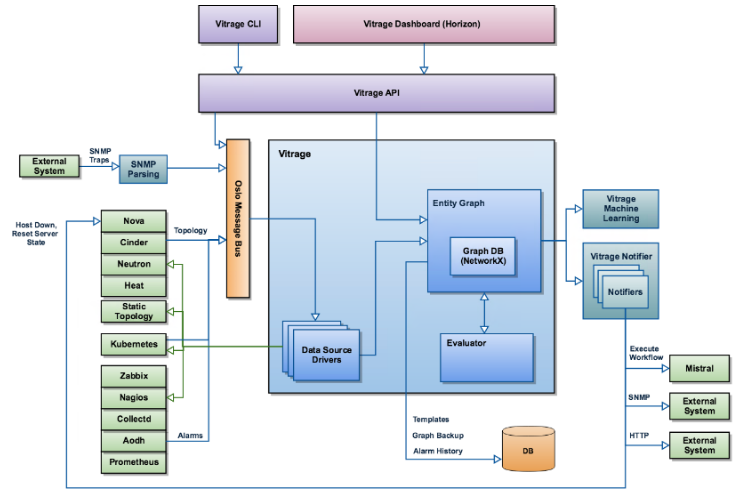
* Hiển thi đầy đủ, toàn diện các tài nguyên: vitrage có thể biểu diễn mỗi quan hệ logic của các tài nguyên trong hệ thống thành dạng đồ thị cho người xem, bao gồm cả thực thể ảo (máy ảo, cổng mạng ảo, zone,..) lẫn thực thể vật lý (máy chủ , switch,..) kết nối với nhau. Để khi có một alarm xuất hiện từ một thành phần ta giải thích được nó sẽ ảnh hưởng tới thành phần khác như thế nào.
* Suy luận ra các cảnh báo: 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 giám sát.

Ví dụ:

Khi switch tới máy chủ vật lý gặp vấn đề, khiến ngay các máy ảo nối với host đó cũng không giám sát được, vitrage có thể đưa ra cảnh báo dự báo với máy ảo.

* Phân tích nguyên nhân cảnh báo: biểu diễn quan hệ nguyên nhân-kết quả giữa các cảnh báo. 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 nhân-quả của các cảnh báo vào biểu đồ. Giúp cho việc tìm ra nguồn kích hoạt gốc của cho các cảnh báo.

Chi tiết kiến trúc của vitrage được mô tả như hình 2.5- Hình ảnh lấy từ nguồn [2]



Hình 2. 5. Kiến trúc của thành phần vitrage

Vitrage bao gồm các khối chính:

* **CLI, API, giao diện**: nhận yêu cầu người dùng, gọi đến các thành phần khác.
* **Khối quản lý nguồn dữ liệu vào** (**vitrage-datasource**): Nhiệm vụ lấy đầu vào thông tin về các tài nguyên hệ thống về cho vitrage graph hiển thị. Các nguồn dữ liệu gồm:
* Từ nguồn openstack : aodh, ceilometer, cinder, heat, neutron, nova.
* Từ nguồn hệ thống giám sát vật lý, gồm: collectd, zabix, nagios, prometheus
* Tài nguyên tĩnh: đọc thông tin từ các file cấu hình người dùng cấp

Dữ liệu bao gồm thông tin như: mô tả về loại tài nguyên và quan hệ giữa các tài nguyên đó.

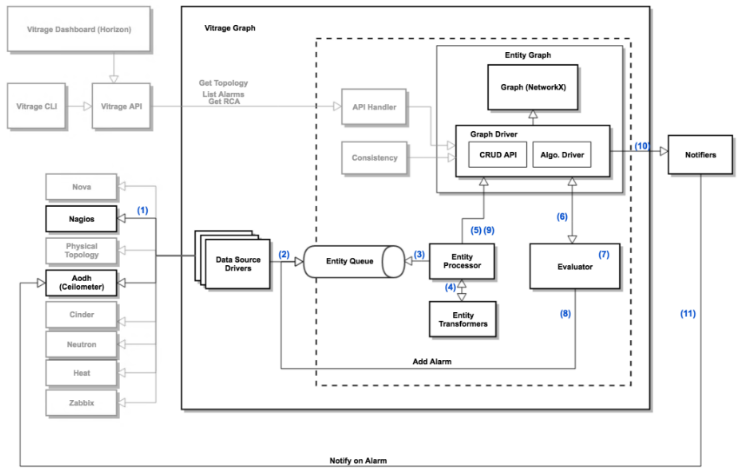
Ví dụ : đầu tiên vitrage truy vấn đến nova để lấy thông tin máy ảo, các zone-máy thuộc zone nào, tiếp là cinder- truy vấn có các ổ đĩa ảo nào- gắn với máy nào, - truy vấn aodh lấy cảnh báo..vv… Cuối cùng sẽ có được bức tranh về toàn bộ hệ thống.

Tuy nhiên cần phân biệt với việc thu thập dữ liệu của ceilometer: vitrage không thu thập dữ liệu giám sát (metric), mà chỉ thu thập các cảnh báo. Mục đích là phân tích nguyên nhân nguồn của các cảnh báo này

* **Khối quản lý đồ thị** (**vitrage-graph**): Giữ các thông tin trên và trình diễn nó ra, đồng thời cũng lấy cập nhật của khối “suy diễn” khi cần. Driver để vẽ đồ thị là networkX , đồng thời cũng cài đặt môt số phép xử lý đồ thị khi được gọi, vd duyệt, tìm đỉnh, cạnh…
* **Khối suy diễn** (**vitrage-evaluator**): Khi có thông báo từ đồ thị về một số thay đổi của các thực thể, nó truy vấn trong các kịch bản (do người dùng định nghĩa từ trước) và áp dụng các thay đổi trạng thái cho các thực thể trong đồ thị của mình (set\_state, raise\_alarm, add\_causual\_relationship)
* **Khối thông báo** (**vitrage notifiers**): có nhiệm vụ thông báo cho các thành phần khác về các thay đổi trong trang thái hệ thống, áp dụng các hành động lên các dịch vụ đó. Hiện có các đối tượng thông báo có thể là cho: aodh, nova, mistral , snmp, webhook
* Khối học máy (**vitrage machine-leaning**): Thiết kế ra nhằm giúp vitrage có thể: xem xét từ tần suất các cảnh báođã xuất hiện, tính rra sự liên quan giữa các cảnh báo , gợi ý cho người dùng viết các template mới.

Ví dụ một luồng hoạt động cơ bản:

Trường hợp thêm có một cảnh báo :



Hình 2. 6. Luồng hoạt động cho thành phần vitrage

Khi phát hiện switch down, vitrage nhận được cảnh báo từ nagios (hoặc từ tick, zabbix,…) Thiết lập kịch bản sẽ thực hiện là: suy luận ra một cảnh báo mất kết nối trên các máy vật lý gắn với switch, các máy ảo gắn trên máy này và gửi lại các cảnh báo suy luận được này cho aodh.

1. Nagios gửi cảnh báo về hàng đợi thông điệp mà vitrage đọc, thành phần vitrage datasource Driver nhận thông báo.

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

3. Entity processor truy vấn sự kiện 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 luật đã có thì thấy có 1 kịch bản cần áp dụng: bật một cảnh báo “error” trên mỗi máy ảo gắn với máy vật lý đó.

8. Evaluator đẩy cảnh báo vào entity queue vitrage

9. Đồ thị cập nhật thêm các cảnh báo mới này.

10. Graph thêm thông báo rằng có 1 cảnh báo mới trên máy ảo, đẩy thông báo này vào messasge queue

11. Vitrage Notifier gửi thêm lại 1 cảnh báo cho aodh

## **2.7. Heat**

Heat (Orchestration service) là dịch vụ giúp, tự động hóa triển khai ứng dụng trên đám mây ,về bản chất đây là công cụ để viết các đoạn mã để người dùng dựng lên hạ tầng máy ảo của mình trong môi trường đám mây (infrastructure as code in cloud) . Heat triển khai ứng dụng đã được mô tả thông qua gọi đến REST API có sẵn trong OpenStack.

Chức năng của heat:

Heat cung cấp các mẫu (template) để mô tả tài nguyên ảo hóa muốn cài đặt trên đám mây. Người dùng lấy các mẫu này từ nhà cung cấp đám mây, cộng đồng mạng hoặc tự viết theo hướng dẫn từ openstack. Nạp vào hệ thống và bổ sung các biến để triển khai tự động hạ tầng ảo hóa của mình

Hệ thống nhận các mẫu này 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.

Vai trò của heat trong hệ thống:

* Heat nhận mô tả cơ sở hạ tầng của tài nguyên cần trong file text mà con người có thể đọc và ghi được. Người dùng từ đó có thể viết mẫu một lần, dùng lại nhiều lần, chia sẻ cho người khác
* Heat có cung cấp dạng mẫu mô tả về “nhóm máy ảo co dãn” (auto scaling group) loại tài nguyên logic đặc biệt giúp quản lý kịch bản mở rộng lên xuống các tài nguyên ảo. Khi kết hợp với với aodh, giúp hệ thống co dãn tài nguyên các máy ảo tự động khi có sự kiện cảnh báo đi đến.

Kiến trúc của heat được mô tả như hình 2.7



Hình 2. 7.Kiến trúc của thành phần heat

Các thành phần trong 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-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 ( OpenStack workflow service), là dịch vụ giúp người dùng có khả năng định nghĩa, thực hiện và quản lý một nhiệm vụ hoặc một chuỗi hành động. Nó cũng định nghĩa chuỗi hành động này dưới dạng template như heat và khi thực hiện sẽ gọi đến các API của openstack, một loại dịch vụ giúp người dùng quản lý tự động các tài nguyên của mình bằng code , hay “infrastructure as code”. Nhưng khác với heat, một nó tập trung vào chuỗi hành động: bao gồm các nhiệm vụ nhỏ nhỏ hoạt động theo một cấu trúc điều khiển dạng nếu-thì IFTTT (if this, then that) như một ngồn ngữ lập trình

So sánh với Heat: heat giúp người dùng chạy kịch bản mô tả về một nhóm các tài nguyên ảo hóa, còn mistral giúp người dùng chạy kịch bản về một chuỗi các hành động . Ví dụ cùng muốn tạo máy ảo nhưng nhưng mẫu mô tả mà người dùng đưa vào của 2 bên sẽ khác biệt:

* Người sử dụng heat sẽ cung cấp mô tả: loại tài nguyên cần là gì, số lượng và mối quan hệ giữa chúng như thế nào
* Người của mistral sẽ cung cấp mô tả: hành động 1: nova.create-instance 🡪 hành động 2: cinder.create volume 🡪 hành động 3: cinder.attach-volume 🡪 nếu thành công thì gửi mail thông báo 🡪 nễu thất bại, thử lại chuỗi hành động trên 5 lần

Vai trò của mistral trong hệ thống:

* Mistral giúp ích cho nhà cung cấp dịch vụ đám mây đặt các kịch bản để tự động các thao tác với hệ thống.
* Mistral kết hợp với vitrage giúp thực hiện quá trình vượt qua lỗi cho hệ thống , cụ thể:

Ý tưởng ở đây là: khi cảnh báo về máy vật lý được gửi đến 🡪 Vitrage suy luận về tác động của cảnh báo, gọi mistral 🡪 Mistral tác động lại với các tài nguyên ảo hóa bị ảnh hưởng.

**Kết luận chương 2.**

Chương 2 đã cho thấy các công nghệ sẽ tích hợp cho hệ thống giám sát, cách mà các ứng dụng được kết nối, tiếp nhận và xử lý luồng dữ liệu giám sát.

Tiếp theo chương 3 sẽ đi sâu vào việc thiết kế và triển khai: đưa ra những ca sử dụng đối với người vận hành hệ thống, và cách bố trí các thành phần để thực hiện được những ca sử dụng này

# 

# **CHƯƠNG 3: PHÁT TRIỂN HỆ THỐNG GIÁM SÁT**

*Chương này sẽ trình bày về việc phát triển cho hệ thống, từ phân tích các đối tượng sử dụng, đên thiết kế và cài đặt cho hệ thống*

## **3.1. Phân tích hệ thống**

### **3.1.1. Các đối tác của hệ thống**

Hiện tại, mô hình đám mây triển khai từ Openstack có các đối tác sau:

**Cloud provider**: người quản trị hệ thống đám mây. Với 3 tầng cung cấp của hệ thống đám mây : infrastructure – hạ tầng máy ảo, platform – nền tảng, software - ứng dụng. thì ở đây tập trung vào tầng infrastructure, tức **cloud provider** có trách nhiệm chính là quản trị máy chủ vật lý và cung cấp ra máy ảo

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

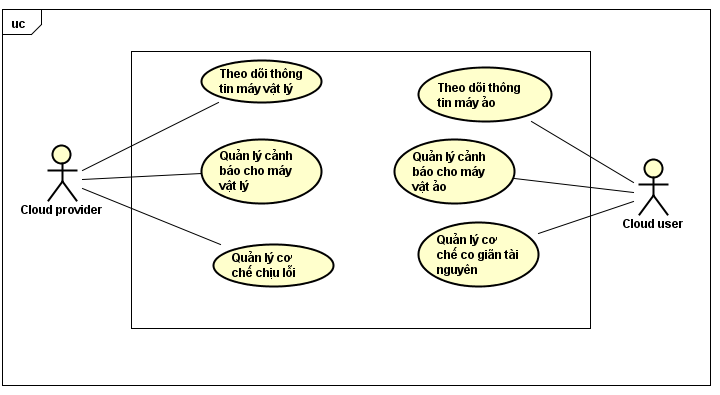
**Cloud provider** và **cloud user** được Openstack phân quyền theo mô hình RBAC :

* Dự án (project): mỗi người dùng tham gia hệ thống đám mây đều thuộc về một số dự án, ví dụ: người dùng 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,…). Kể cả **cloud provider** cũng thuộc về project admin.
* Role admin: role của cloud provider, role này có quyền quản lý vòng đời của mọi tài nguyên: tạo xóa máy ảo, volumes ảo,… Tuy nhiên không có quyền đăng nhập sử dụng các máy ảo, là tài sản sẽ bàn giao cho cloud-user. Role được gán mặc định cho **cloud provider**
* Role user: role của cloud user, role này có quyền tạo, xóa tài nguyên trong dự án của mình. Quyền truy cập sử dụng máy ảo. Role này được **cloud provider** gán cho các **cloud user** khác.

### **3.2.2. Các ca sử dụng của hệ thống**

*Các ca sử dụng dưới đây không phải là từ quá trình tự thiết kế của đồ án mà chỉ có ý nghĩa là mô tả lại các ca sử dụng từ các công nghệ mang lại. Mục tiêu là tích hợp được các ca sử dụng này vào thành chức năng hệ thống.*

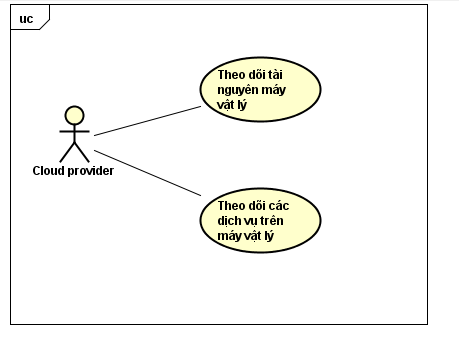
#### **Tổng quan các chức năng của hệ thống giám được mô tả trong biểu đồ tổng quan (hình 3.1)**



Hình 3.1. Biểu đồ tổng quan của hệ thống giám sát

#### **3.2.2.1. Các ca sử dụng của cloud-provider**

**Các chức năng của theo dõi thông tin máy vật lý được mô tả như hình 3.2**



Hình 3. 2. Biểu đồ ca sử dụng cho chức năng theo dõi thông tin máy vật lý

**Mô tả cụ thể cho chức năng này:**

**UC1: Theo dõi tài nguyên máy vật lý**

Mô tả:

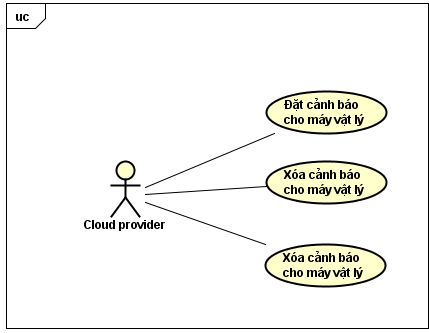
* User : cung cấp hostname
* Hệ thống: cung cấp ra giao diện bao gồm các đồ thị thể hiện dữ liệu giám sát theo tài nguyên của máy, trong từng khoảng thời gian. Cần có các biểu đồ thể hiện:
  + Tài nguyên chung của hệ thống: cpu.load, ram.usage, network
  + Tài nguyên mà các dịch vụ chiếm dùng, ví dụ như dịch vụ mysql, rabbitmq,…
* User: cung cấp khoảng thời gian mong muốn quan sát
  + Hệ thống trả về đồ thị hiển thị dữ liệu trong khoảng thời gian đó

**UC2: Theo dõi dịch vụ máy vật lý**

Mô tả:

* User : cung cấp tên dịch vụ, ví dụ dịch vụ mysql
* Hệ thống: cung cấp ra giao diện bao gồm các đồ thị thể hiện dữ liệu giám sát của dịch vụ trên tất cả các máy, trong từng khoảng thời gian. Tùy vào mỗi dịch vụ mà thông số cần quan tâm lại khác biệt
* User: cung cấp khoảng thời gian mong muốn quan sát
  + Hệ thống trả về đồ thị hiển thị dữ liệu trong khoảng thời gian đó

**Các chức năng của quản lý cảnh báo máy vật lý được mô tả trong biểu đồ hình 3.3**

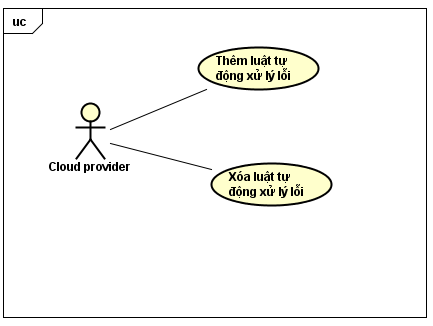


Hình 3.3.Biểu đồ ca sử dụng cho chức năng quản lý cảnh báo máy vật lý

**Mô tả cụ thể chức năng này:**

* User : cung cấp tên chỉ số giám sát, cách đo lường, ngưỡng cảnh báo và kênh thông báo. Ví dụ (memory.usage/memory.alocation) > 90% trả thì về mức critical, từ 90 đến 70 trả về warning, gửi thông báo vào email
* Hệ thống: cung cấp ra giao diện thông báo luật cảnh báo trên đã được ghi vào hệ thống. Lịch sử các cảnh báo
* Hệ thống xử lý dữ liệu giám sát để đưa cảnh báo tại thời điểm đúng với luật.
* Nếu user sửa thông tin mới cho cảnh báo
  + Hệ thống ghi nhận và sửa lại luật này
* Nếu user thông báo tắt, xóa cảnh báo này
  + Hệ thống ghi nhận và xóa luật

**Các chức năng của quản lý cơ chế chịu lỗi được mô tả trong biểu đồ hình 3.4**



Hình 3.4.Biểu đồ ca sử dụng cho chức năng quản lý cơ chế chịu lỗi

Cơ chế chịu lỗi: là cơ chế phản ứng với sự cố, gới mục tiêu là khi có lỗi xảy ra trên máy chủ vật lý thì máy ảo và các dịch vụ cung cấp người dùng không- hoặc ít bị ảnh hưởng.

Ví dụ cho một cơ chế xử lý lỗi:

* Khi máy chủ vật lý gặp sự cố (tắt đột ngột, nic mất kết nối, disk low latency, cpu cao tải,vv…),hệ thống tự động di tản các máy ảo trên máy này sang một nốt khác để user tiếp tục sử dụng được máy ảo bình thường, minh họa như hình 3.5



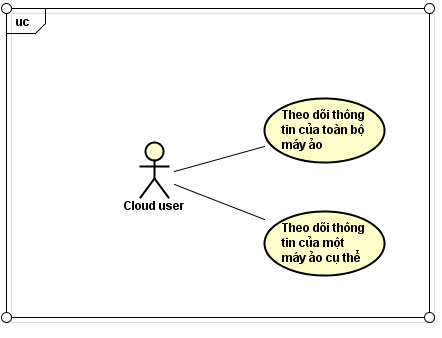
Hình 3. 5. Cơ chế chịu lỗi trong hệ thống đám mây

Mô tả ca sử dụng:

* User : cung cấp kịch bản cho việc tự động xử lý lỗi, kịch bản bao gồm:
  + Mô tả cảnh báo sẽ đến với hệ thống: là cảnh báo của tài nguyên vật lý hay ảo hóa. Đối tượng cụ thể cảnh báo chỉ ra là gì: nốt vật lý nào, card mạng nào, switch nào, hoặc máy ảo nào
  + Mô tả hệ thống cách ứng phó , ví dụ: gửi email cho quản trị, tự động thay thế máy ảo, di chuyển máy ảo về khu vực máy vật lý không có lỗi
* Hệ thống: Ghi lại các luật trên, khi có sự kiện cảnh báo xảy đến sẽ thực hiện đúng như mô tả của kịch bản. Chỉ ra được những kịch bản nào đã được áp dụng.

#### **3.2.2.2. Các ca sử dụng của Cloud-user**

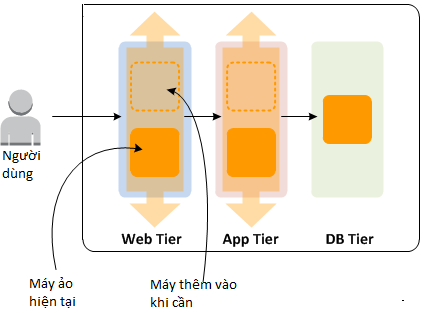
**Ca sử dụng theo dõi thông tin tài nguyên máy ảo được mô tả trong hình 3.6**



Hình 3. 6. Biểu đồ mô tả ca sử dụng người dùng theo dõi tài nguyên máy ảo

**Ca sử dụng quản lý cơ chế co dãn tài nguyên**

Co dãn tài nguyên (auto scaling) sẽ giúp người dùng giảm bớt các máy ảo hoạt động trong thời gian thấp điểm và tự động tăng tài nguyên khi máy cao tải. Kịch bản cho một nhóm tài nguyên co dãn được mô tả như hình 3.7



Hình 3.7.Mô tả cơ chế co dãn tài nguyên

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 có nhu cầu phục vụ cao). Thay vào đó họ sẽ tăng thêm số lượng máy ảo chạy ứng dụng và cấu hình cân bằng tải cho chúng.

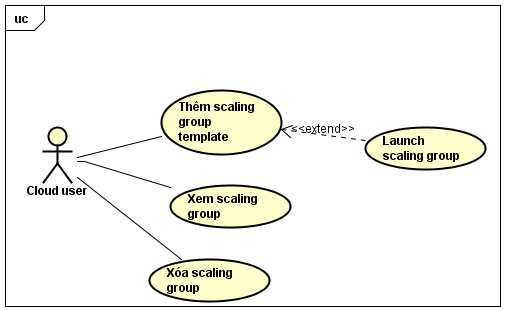
App, web được bố trí trên các máy ảo thuộc các nhóm tài nguyên co dãn hay **scaling-group**.

Một **scaling-group** bao gồm:

* Nhóm các máy ảo hiện tại, số lượng máy ảo tối đa, tối thiểu trong group
* Các alarm aodh đã chuẩn bị trước
* Scaling rule : ví dụ khi có cảnh báo cpu cao tải thì +1máy ảo, cpu thấp thì -1 máy ảo

Yêu cầu về ứng dụng của người dùng: các máy chạy database và máy chạy application tách biệt nhau. Ta không thể co dãn tài nguyên 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 với các công nghệ của heat, telemetry



Hình 3. 8. Biểu đồ ca sử dụng quản lý cơ chế co dãn tài nguyên

Mô tả ca sử dụng:

* User : cung cấp một template cho việc tự động co dãn tài nguyên, template bao gồm.
  + Scaling group: chỉ ra thông số ram, cpu, ổ đĩa cần dùng cho mỗi máy ảo thuộc nhóm, số lượng máy ảo tối đa, tối thiểu và mặc đinh trong nhóm
  + Scaleup policy: chính sách cho một lần mở rộng tài nguyên: số máy ảo được thêm vào
  + Scaledown policy: chính sách cho một lần rút nhỏ tài nguyên: số máy ảo được bớt đi
  + Alarm high: luật cảnh báo định ra khi nào thì cảnh báo tài nguyên vượt ngưỡng
  + Alarm low: luật cảnh báo đinh ra khi nào thì cảnh báo tài nguyên dưới ngưỡng
* User tạo yêu cầu triển khai cho nhóm tài nguyên này:
  + Hệ thống: tạo ra scaling group với số lượng các máy ảo mặc định. Đặt ra các cảnh báo về tài nguyên
  + Nếu có cảnh báo “alarm high”, hệ thống tự động mở rộng tài nguyên như scaleup policy
  + Nếu có cảnh báo “alarm low”, hệ thống tự động giải phóng tài nguyên như scaledown policy

## **3.2. Thiết kế hệ thống**

### **3.2.1.Bố trí các thành phần giám sát.**

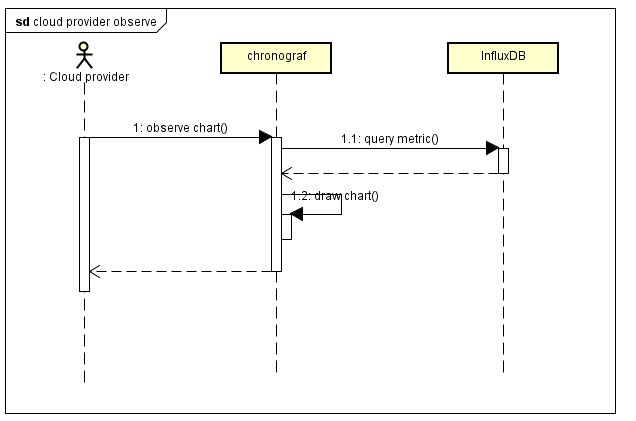
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Yêu cầu** | **Nhóm công nghệ** | **Thu thập dữ liệu** | **Lưu trữ dữ liệu** | **Phát cảnh bảo** | **Thực hiện ứng phó** |
| Máy vật lý+  Tiến trình OpenStack | TICK | Telegraf | InfluxDB | Kapacitor | Kapacitor |
| Tài nguyên ảo OpenStack (máy ảo, ổ cứng, ports, networks,…) | Telemetry | Ceilometer | Gnocchi | Aodh | Vitrage + Mistral + Heat |



Hình 3. 9. Sơ đồ phân bố các thành phần giám sát

### **3.2.2 Mô tả luồng hoạt động**

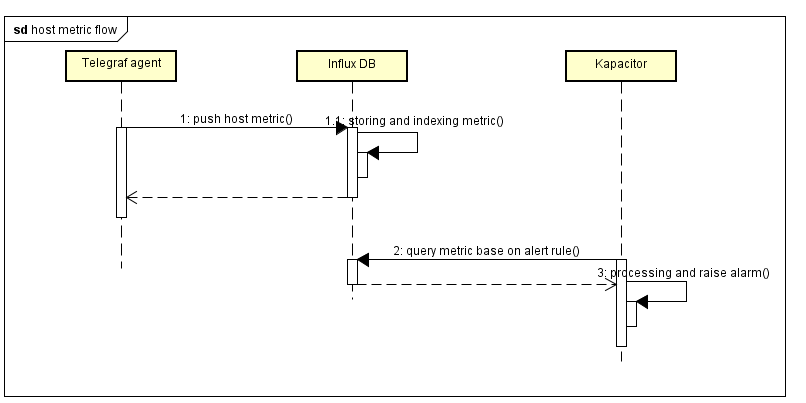
Luồng hoạt động cho chức năng cloud provider quan sát tài nguyên máy chủ vật lý



Hình 3. 10. Biểu đồ trình tự mô tả người dùng xem thông tin máy vật lý

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

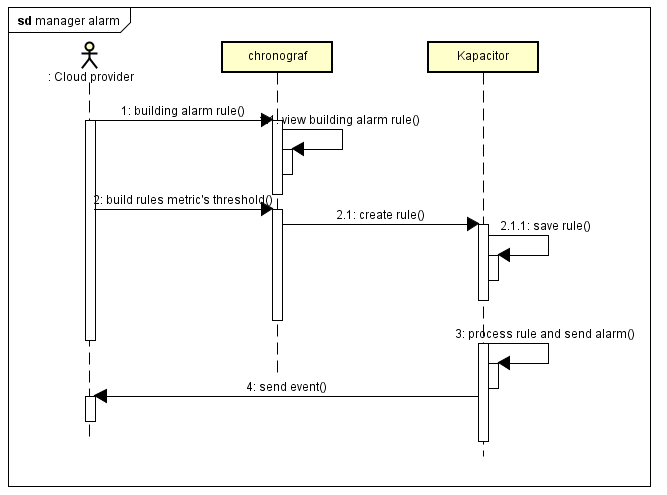
Luồng hoạt động để xử lý metric tài nguyên máy vật lý.



Hình 3. 11.Biểu đồ trình tự mô tả luồng xử lý dữ liệu giám sát máy vật lý

* Telegraf cài trên các máy vật lý cần giám sát, thu thập thông số và trả về thông tin thành các điểm dữ liệu, một điểm bao gồm:
  + Nhãn thời gian timestamp
  + Một thông số đo (ví dụ "cpu\_load")
  + Ít nhất một truờng key-value giá trị của thông số (giá trị đo được, ví dụ "value=0.64", hoặc " percentage=21.2")
  + Các nhãn gồm các key-value mô tả về tài nguyên (ví dụ “hostname=server01”)
* Các điểm được đẩy về cho influx db lưu trữ, Influx db đánh chi mục cho timestamp và các nhãn
* Kapacitor đọc và xử lý dữ liệu từ influx db, có 2 kiểu xử lý:
  + Stream: đăng ký các dữ liệu mà ngay khi influx nhận được sẽ lấy ngay, vd: trạng thái service 0-1 để giám sát 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%

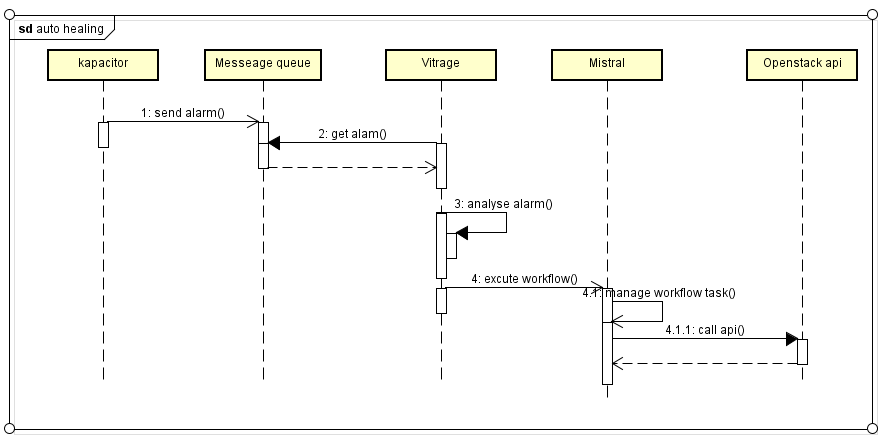
Luồng hoạt động cho chức năng cloud provider thêm cảnh báo cho tài nguyên host vật lý



Hình 3. 12. Biểu đồ trình tự mô tả người dùng đặt cảnh báo cho tài nguyên vật lý

* Cloud provider vào giao diện của chronograf, chọn tạo luật cảnh báo. Chronograf hiển thị giao diện giúp người dùng tạo luật.
* Người dùng xác nhận tạo luật, thông tin được chronograf gửi về kapacitor
* Kapacitor lưu lại luật, truy vấn xử lý các dữ liệu theo luật đã thiết lập. gửi về người dùng khi có luật bị áp dụng. vd gửi mail khi cpu.system.usage >90 % .

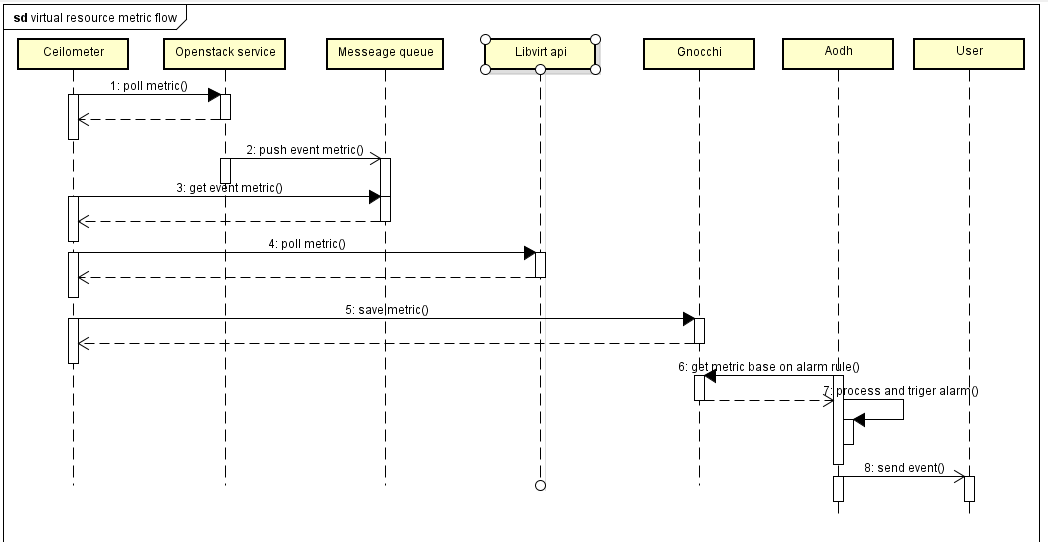
Luồng hoạt động thực hiện tự động vượt qua lỗi khi phát hiện cảnh báo trả về



Hình 3. 13.Biểu đồ trình tự mô tả quá trình tự động vượt qua lỗi của hệ thống

* Kapacitor trả về cảnh báo :có lỗi tại một thành phần của host, gửi vào messeage queue
* Vitrage nhận cảnh báo này, xử lý (theo kịch bản mà người quản trị cloud-provider đã định từ trước). ví dụ kịch bản nêu ra: máy vật lý tắt đột ngôt sẽ khiến tất cả máy ảo tắt theo, hướng xử lý trong kịch bản sẽ là gọi chuỗi hành động để di tản các máy ảo trong vùng ảnh hưởng
* Vitrage tìm các máy ảo bị ảnh hưởng, gọi sang mistral thông báo cần chạy các chuỗi hành động(workflow) di tản máy ảo, đưa ra các id máy ảo.
* Mistral nhận được lời gọi workflow từ vitage và thực hiện các nhiệm vụ con trong workflow
* Mistral gọi nova api, neutron api,… để tác động lên máy ảo

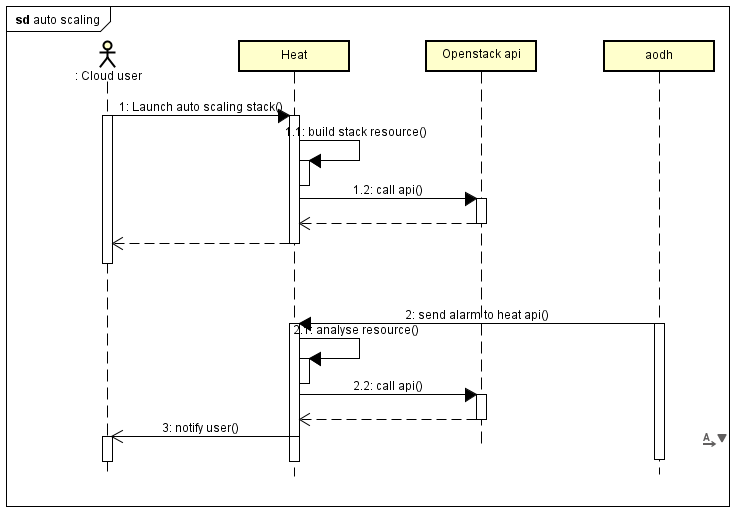
Luồng hoạt động để xử lý metric tài nguyên ảo hóa.



Hình 3. 14. Biểu đồ trình tự mô tả hệ thống thu thập dữ liệu tài nguyên ảo hóa

* Ceilometer truy vấn các api của project khác của openstack để lấy về thông tin của máy ảo, điển hình như ip, các ổ cứng gắn vào.
* Các openstack service khi có một sự kiện đến với tài nguyên ảo cũng gửi các sự kiện đó đến messeage queue, vd như create, shutdown, resize,…. Ceilometer lắng nghe trên hàng đợi thông điệp các sự kiện này
* Ceilomter agent trên host compute của hệ thống truy vấn libvirt api để lấy ra được các metric về máy ảo, 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 khi nào giá trị dữ liệu quá ngưỡng để trả về cảnh báo cho người dùng

Luồng hoạt động thực hiện tự động co dãn tài nguyên.



Hình 3. 15. Biểu đồ trình tự mô tả chức năng người dùng đặt cảnh báo

* Người dùng tạo mô tả về scaling group. Sử dụng dùng giao diện dashboard horizon của openstack để giao tiếp với heat
* Heat gọi openstack api tạo ra tài nguyên đúng như người dùng yêu cầu. bao gồm
  + Các máy ảo , ổ đĩa ảo, network
  + Định nghĩa về cảnh báo xuất hiện khi các dữ liệu giám sát lên 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 máy ảo, volume ảo, 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 giám sát được 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 dịch vụ cốt lõi của openstack:  Keystone, Nova, Glance, Neutron, Glance, Cinder |
| Máy compute | Cài đủ các dịch vụ core của openstack:  neutron-agent , nova-compute cinder-volume. |
| Lưu trữ ổ cứng ảo | Share storage: ceph, glusterfs, nfs… làm của nơi lưu trữ tập trung |

Bảng 3. 1. Yêu cầu cho hệ thống đám mây sẵn có

### **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 giám sát-center | InfluxDB, Kapacitor, Chronograf, Grafana |

Bảng 3. 2. Bố trí các thành phần của dịch vụ giám sát

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

Mô hình hệ thống thử nghiệm giống mô tả ở hình hình 3.8

**b) Ước lượng tài nguyên cần cho các dịch vụ giám sát**

Tài nguyên host dành cho dịch vụ giám sát:

|  |  |  |  |
| --- | --- | --- | --- |
|  | RAM | CPU | Disk |
| Máy controller | +3 GB | + 4 core | + 10GB |
| Máy compute | +1 GB | + 2 core |  |
| Máy monitor-center | +2 GB | + 2 core | +5 GB với mỗi node cần giám sát |

Bảng 3. 3. Tài nguyên ước tính sử dụng cho hệ thống giám sát

Các số liệu trên được lấy từ thông tin hệ thống chạy trong môi trường lab: giám sát 3 server.

Tính toán dung lượng của database gnocchi cần cho lưu dữ liệu giám sát của một máy ảo:

* Kích thước dữ liệu lưu trữ cho mỗi điểm dữ liệu:
  + Trường hợp tệ nhất: 8.04 bytes/point
  + Trường hợp tốt nhất: 0.05 bytes/point
* Số loại dữ liệu tối đa tại mỗi máy ảo tạo ra:
  + Dữ liệu về máy ảo: 44
  + Dữ liệu về ổ cứng máy ảo: 15

**🡪** tại một thời điểm, dung lượng dữ liệu tối đa mà 1 máy ảo tạo ra:

8.04 \* (44+15) = 475 bytes

* Gộp 60s điểm dữ liệu thành 1 bản ghi-60s ( phép toán trung bình cộng, tổng,đếm…) (A)
* Gộp 60 các điểm dữ liệu 60s trên thành 1 bản ghi-3600s (B)
* Nhóm các bản ghi A: lưu trong 7 ngày
* Nhóm các bản ghi B: lưu trong 365 days

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

🡪 Lượng tài nguyên chiếm trong database gnocchi, sử dụng dành cho 1 máy ảo là:

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

Đối máy chủ vật lý dùng influxDB lưu dữ liệu giám sát, tuy thực không có cách nào tính toán về lý thuyết, tuy nhiên theo như như tham khảo kinh nghiệm quản trị hệ thống thực và tài liệu từ influxdb, với hệ thống giám sát cho 20 máy chủ vật lý, mỗi tuần sẽ có thêm khoảng vài GB dữ liệu được ghi thêm vào.

**c) Cài đặt hệ thống**

**Cài đặt thủ công hệ thống**

*Việc cài đặt mà Đồ án giới thiệu ở đây chỉ tóm lược các bước, giai đoạn quan trọng nhất, các bước móc nối giữa các thành phần. Chi tiết hơn về cài đặt xin tham khảo các tài liệu [5], [6], [7], [8], [9], [10]*

**Bước 1: Cài đặt influxdb**

Vào máy monitor-central và thực hiện các bước sau:

**Cài infiluxdb**

curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -

source /etc/lsb-release

echo "deb https://repos.influxdata.com/${DISTRIB\_ID,,} ${DISTRIB\_CODENAME} stable" | sudo tee /etc/apt/sources.list.d/influxdb.list

sudo apt-get update & sudo apt-get install influxdb

Bắt đầu vào dao diện influx console và tạo các user đầu tiên

# influx

> CREATE USER "**sammy**" WITH PASSWORD **'sammy\_admin'** WITH ALL PRIVILEGES

**Bước 2: Cài telegraf trên các máy vật lý muốn giám sát**

sudo apt-get install telegraf

sửa file /etc/telegraf/telegraf.conf, khai báo influxdb mà dữ liệu nó sẽ đẩy về, bật thu thập dữ liệu cho các dịch vụ như mysql, rabbit, redis….

|  |
| --- |
| **[[outputs.influxdb]]**  urls = ["http://monitor-center:8086"]  database = "telegraf"  username = "sammy"  password = "sammy\_admin"  **[[inputs.mysql]]**  servers= ["mysql\_user:password@tcp(127.0.0.1:3306)/"]  **[[inputs.rabbitmq]]**  url = "http://controller:15672"  username = "guest"  password = "guest"  **[[inputs.apache]]**  urls = ["http://controller/horizon?auto"]  # nếu url có thiết đặt bảo mật  username = <user>  password = <password>  …. |

sudo systemctl restart telegraf

**Bước 3: Cài đặt kapacitor**

sudo apt-get install kapacitor

Sửa file /etc/kapacitor/kapacitor.conf nhằm chỉ ra influxdb mà nó lấy dữ liệu

|  |
| --- |
| [[influxdb]]  name = "monitor-center"  urls = ["http:// monitor-center:8086"]  username = "sammy"  password = "sammy\_admin" |

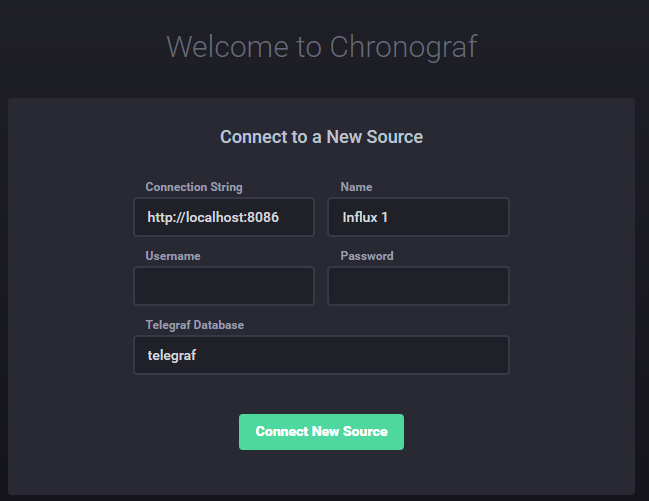
sudo systemctl start kapacitor

**Bước 4: cài đặt Chronograf làm giao diện cho TICK**

wget https://dl.influxdata.com/chronograf/releases/chronograf\_1.7.11\_amd64.deb

sudo dpkg -i chronograf\_1.7.11\_amd64.deb

Vào máy cài đặt trên, cổng 8888 ta thấy được giao diện web của chronograf. Thiết lập kết nối với influxdb, dao diện sẽ như hình 3.6. Điền vào username, password của user trên influxdb



Hình 3. 16. Cài đặt kết nối chronograf và influxdb

**Bước 5. Cài đặt Ceilometer, gnocchi và aodh**

Tại máy controller, cài đặt các gói cần thiết cho cả 3 dịch vụ

add-apt-repository cloud-archive:queens & apt update -y

apt-get install gnocchi-api gnocchi-metricd python-gnocchiclient \

ceilometer-agent-notification ceilometer-agent-central \

aodh-api aodh-evaluator aodh-notifier aodh-listener python-aodhclient

Tạo các cơ sở dữ liệu trong mysql và gán quyền

mysql -u root -p

> CREATE DATABASE aodh;

> GRANT ALL PRIVILEGES ON aodh.\* TO 'aodh'@'localhost' \

IDENTIFIED BY 'AODH\_DBPASS';

> GRANT ALL PRIVILEGES ON aodh.\* TO 'aodh'@'%' \

IDENTIFIED BY 'AODH\_DBPASS';

> CREATE DATABASE gnocchi;

> GRANT ALL PRIVILEGES ON gnocchi.\* TO 'gnocchi'@'localhost' \

IDENTIFIED BY 'GNOCCHI\_DBPASS';

> GRANT ALL PRIVILEGES ON gnocchi.\* TO 'gnocchi'@'%' \

IDENTIFIED BY 'GNOCCHI\_DBPASS';

Tạo các user endpoint trong keystone, từ đây người dùng có thể truy cập đến dịch vụ thông qua các endpoint này.

openstack user create --domain default --password-prompt ceilometer

openstack role add --project service --user ceilometer admin

openstack service create --name ceilometer --description "Telemetry" metering

openstack user create --domain default --password-prompt gnocchi

openstack service create --name gnocchi --description "Metric Service" metric

openstack role add --project service --user gnocchi admin

openstack user create --domain default --password-prompt aodh

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

openstack service create --name aodh --description "Telemetry" alarming

openstack endpoint create --region RegionOne alarming public http://controller:8042

openstack endpoint create --region RegionOne metric public http://controller:8041

Sửa file cấu hình cho các dịch vụ aodh, ceilometer, gnocchi. Tại các file /etc/gnocchi/gnocchi.conf, /etc/aodh/aodh.conf, /etc/ceilometer/ceilometer.conf ,cấu hình liên kết chúng với rabbitmq, keystone và mysq. Ví dụ cho aodh:

|  |
| --- |
| [DEFAULT]  ...  transport\_url = rabbit://openstack:RABBIT\_PASS@controller  ...  [keystone\_authtoken]  ...  www\_authenticate\_uri = http://controller:5000  auth\_url = http://controller:5000  memcached\_servers = controller:11211  auth\_type = password  project\_domain\_id = default  user\_domain\_id = default  project\_name = service  username = aodh  password = AODH\_PASS  ...  [database]  ...  connection = mysql+pymysql://aodh:AODH\_DBPASS@controller/aodh |

**Bước 6: cấu hình thu thập thông tin**

Cài tác tử thu thập thông tincho của ceilometer lên các máy compute.

apt-get install ceilometer-agent-compute

Tại các file cấu hình của cinder, glance, heat, nova, neutron cũng cấu hình để các dịch vụ này gửi sự kiện của mình về messeage queue để ceilometer thu thập

Ví dụ như cinder, file /etc/cinder/cinder.conf

|  |
| --- |
| [Default]  ...  transport\_url = rabbit://openstack:RABBIT\_PASS@controller  ...  [oslo\_messaging\_notifications]  driver = messagingv2 |

Các dịch vụ khác cũng cấu hình tương tự

**Bước 7: Cài đặt Vitrage, heat, mistral.**

Đây cũng là các dịch vụ của openstack cài đặt cũng qua các bước như tạo keystone user, endpoint, khởi tạo cơ sở dữ liệu mysql và thêm thông tin về messeage queue, keysonte và database vào file cấu hình, tương tự như bước 5. Chi tiết có thể xem tại [8], [9], [10], [11]. Tuy nhiên cần chú ý các điểm sau:

Cấu hình cho vitrage lấy các nguồn dữ liệu từ kapacitor, aodh, và các dịch vụ nền khác của openstack: file /etc/vitrage/vitrage.conf

|  |
| --- |
| [datasources]  types = **kapacitor,aodh**,nova.host,nova.instance,nova.zone,static,cinder.volume,neutron.network,neutron.port,heat.stack |

Đăng ký cho vitrage có thể gọi mistral để thực hiện workflow

|  |
| --- |
| [DEFAULT]  notifiers = nova,mistral |

Tại các dịch vụ khác, đăng ký sẽ gửi thông tin các sự kiện cho vitrage, ví dụ từ file /etc/aodh/aodh.conf.

|  |
| --- |
| [oslo\_messaging\_notifications]  driver = messagingv2  topics = notifications,vitrage\_notifications |

**Cài đặt tự động hệ thống**

Hệ thống được cài đặt tự động từ ansible. Hướng dẫn cho cài đặt ansible:

<https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-16-04>

Dowload code ansible tại:

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

Code này chứa kịch bản cho cài đặt hệ thống giám sát.

Sau khi cài đặt ansible và dowload code cài đặt vào host đó. Ta cần chuẩn bị các biến đầu vào trong code ansible như sau:

File inventory/host:

|  |  |  |
| --- | --- | --- |
| **Tên biến** | **Giải thích** | **Ví dụ** |
| [all] | Danh sách các server cần cài đặt, ip ssh tới | [all]  server01 ansible\_ssh\_host=192.168.1.211  server02 ansible\_ssh\_host=192.168.1.212  server03 ansible\_ssh\_host=192.168.1.213 |
| [controller] | Danh sách server làm chức năng controller | [controller]  server01 |
| [compute] | Danh sách server làm chức năng compute | [compute]  server02  server03 |
| [monitor-center] | Server làm giám sát-center | [monitor-center]  server01 |
| [telegraf] | Danh sách server cài telegraf agent | [telegraf]  server01  server02  server03 |

File group\_var/all.yml

|  |  |  |
| --- | --- | --- |
| **Tên biến** | **Giải thích** | **Ví dụ** |
| openstack\_release | Phiên bản openstack | queens |
| rabbit\_user | Rabbit user | openstack |
| rabbit\_passwd | Rabbit passwd | Dumpbswin |
| rabbit\_host | Rabbit host | 192.168.1.211 |
| rabbit\_port | Rabbit port | 5672 |
| mysql\_user | Mysql user | Root |
| mysql\_host | Mysql host | 192.168.1.211 |
| mysql\_pass | Mysql password | Welcome123 |
| mysql\_port | mysql port | 3306 |
| memchached\_host | memchached\_host | 192.168.1.211 |
| memcached\_port | memcached\_port | 11211 |
| memcached\_pass | memcached\_pass | Welcome123 |
| keystone\_env.admin\_pass | Password user admin | Welcome123 |
| keystone\_env.region | Region | RegionOne |
| keystone\_env.project\_domain\_id | Domain project | defaut |
| keystone\_env.user\_domain\_id | Domain user | default |

Cài đặt:

* Đảm bảo ssh từ host ansible tới các host cần cài (user ssh và key khai báo trong ansible.conf)
* Đảm bảo các host cần cài kết nối được internet ( lệnh *apt-get update* thành công)
* Init ansible

ansible-playbook init.yml

* Chạy lệnh để cài TICK stack :

ansible-playbook TICK.yml

* Chạy lệnh để cài Telemetry stack:

ansible-playbook telemetry.yml

* Chạy lệnh để cài mitral heat vitrage:

ansible-playbook triger\_service.yml

### **3.3.3. Chi tiết cách thiết lập cơ chế chịu lỗi cho hệ thống**

Đây là kịch bản do cloud-provider thực hiện, dựa trên các chức năng của hệ thống

**a) Kịch bản**

Giám sát thông tin:

* Máy vật lý tắt: compute-01 không trả về dữ liệu giám sát trong 120 s, recheck 60s 1 lần

Thiết lập vượt qua lỗi:

* Di tản các máy ảo từ host bị down sang host khác

(ổ đĩa ảo của máy ảo nằm trên kho lưu trữ tập trung là ceph nên dù có host down vẫn có khả năng tạo lại máy ảo đó trên nốt vật lý khác )

**b) Cách thực hiện:**

**Cấu hình cảnh báo:**

Cấu hình đặt alarm tại kapacitor dịch vụ, ta định nghĩa alarm host down như sau:

Tạo file tick\_script.txt:

|  |
| --- |
| dbrp "telegraf"."autogen"  var period = 120s  var every = 60s  var sys\_data = stream  |from()  .measurement('system')  .groupBy('host')  var trigger = sys\_data  |deadman(0.0, period)  .message('host offline')  .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 hàng đợi thông điệp của openstack. Vitrage sẽ đọc được alarm này và cho vào đồ thị của mình

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  kapacitor endnable check\_host\_down |

**Cấu hình luật tự động hồi phục tài nguyên ảo (healing rule)**

Cấu hình healing rule tại vitrage: nếu có cảnh báo máy vật lý mất kết nối, quyết định rằng máy đó đã tắt , gọi workflow mistral : di tản các máy ảo sang máy khác

vi ~/host\_down\_scenario.yaml

Hoặc 1 trường hợp khác, live migrate khi host cao tải:

|  |
| --- |
| metadata:  version: 2  name: host high resource usage scenario  type: standard  description: raise deduced alarm and call live migrate instance when host high resource usage  definitions:  entities:  - entity:  category: ALARM  name: host high resource usage  type: kapacitor  template\_id: host\_alarm  - entity:  category: ALARM  name: instance\_resource\_problem  template\_id: instance\_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  - relationship:  source: instance\_alarm  target: instance  relationship\_type: on  template\_id : alarm\_on\_instance  scenarios:  - scenario:  condition: **alarm\_on\_host and host\_contains\_instance**  actions:  - action:  action\_type : raise\_alarm  properties:  alarm\_name: instance\_resource\_problem  severity: critical  action\_target:  target: instance  - action:  action\_type : **execute\_mistral**  properties:  workflow: **instance\_live\_migrate**  input:  instance\_id: get\_attr(instance, id)  retries: 5  - scenario:  condition: alarm\_on\_host and host\_contains\_instance and alarm\_on\_instance  actions:  - action:  action\_type : add\_causal\_relationship  action\_target:  source: host\_alarm  target: instance\_alarm |

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

Tại mistral định nghĩa **evacuate\_instance** workflow trên: đầu vào của workflow này sẽ là id máy ảo, khi workflow được vitrage gọi thực hiện, nó sẽ di tản các máy ảo này sang một máy vật lý mới.

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

### **3.3.4. Chi tiết cách thiết lập cơ chế co dãn tài nguyên**

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

**a) Kịch bản**

Giám sát thông tin cpu\_util: phần trăm sử dụng cpu của máy ảo

Tự động co dãn tài nguyên:

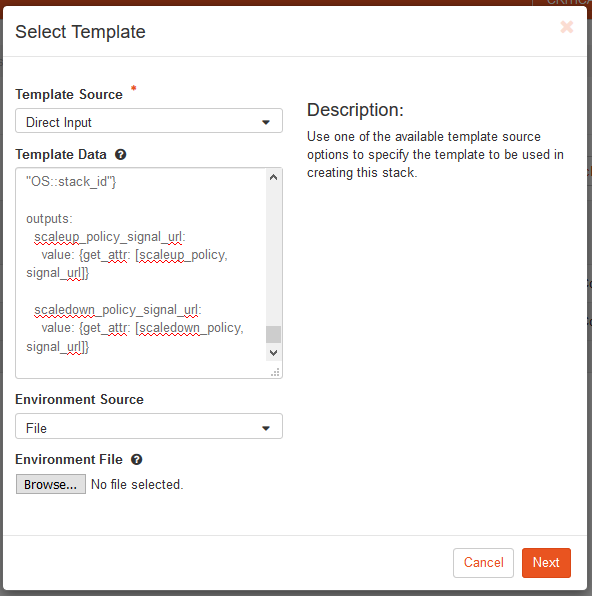
* Nhóm các máy ảo khi mới tạo chỉ bao bao gồm 1 máy ảo
* Nếu có cảnh báo cpu > 80% tự động tạo thêm 1 máy ảo
* Nếu có cảnh báo cpu < 5% giảm bớt 1 máy ảo
* Số máy ảo tối đa trong nhóm là 3, tối thiểu trong nhóm là 1

**b) Cách thực hiện:**

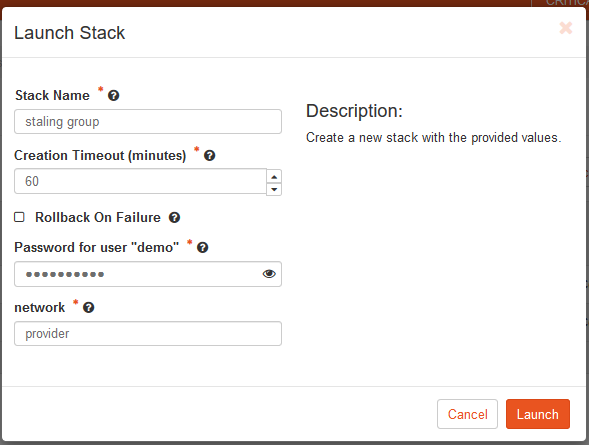
User tạo template scaling-group của như sau:

|  |
| --- |
| 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.  max\_size:  type: string  description: max number of instance.  min\_size:  type: string  description: min number of instance.  desired\_size:  type: string  description: defaut number of instance.  metric\_watch:  type: string  description: metric for alarm rule  default: cpu\_util  scale\_down\_threshold:  type: string  description: threshold metric then scale down  default: 0.5  scale\_up\_threshold:  type: string  description: threshold metric then scale down  default: 80  start\_script:  type: string  description: script run when host start  default: echo "hello world"  key\_ssh:  type: string  description: ssh key name  default: mykey  resources:  scaling\_group:  type: OS::Heat::AutoScalingGroup  properties:  cooldown: 300  desired\_capacity: { get\_param: desired\_size }  max\_size: { get\_param: max\_size }  min\_size: { get\_param: min\_size }  resource:  type: OS::Nova::Server  properties:  image: cirros  flavor: m1.tiny  key\_name: { get\_param: key\_ssh }  networks:  - network: { get\_param: network }  metadata: {"metering.server\_group": {get\_param: "OS::stack\_id"}}  user\_data:  |  # !/bin/bash  $SHELL\_SCRIPT  # ...  scaleup\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaling\_group }  cooldown: 300  scaling\_adjustment: 1  scaledown\_policy:  type: OS::Heat::ScalingPolicy  properties:  adjustment\_type: change\_in\_capacity  auto\_scaling\_group\_id: { get\_resource: scaling\_group }  cooldown: 300  scaling\_adjustment: -1  alarm\_high:  type: OS::Aodh::GnocchiAggregationByResourcesAlarm  properties:  description: Scale up if metric > theshold  metric: { get\_param: metric\_watch }  aggregation\_method: mean  granularity: 60  evaluation\_periods: 1  threshold: { get\_param: scale\_up\_threshold }  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"}  alarm\_low:  type: OS::Aodh::GnocchiAggregationByResourcesAlarm  properties:  metric: { get\_param: metric\_watch }  aggregation\_method: mean  granularity: 60  evaluation\_periods: 1  threshold: { get\_param: scale\_down\_threshold }  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]} |

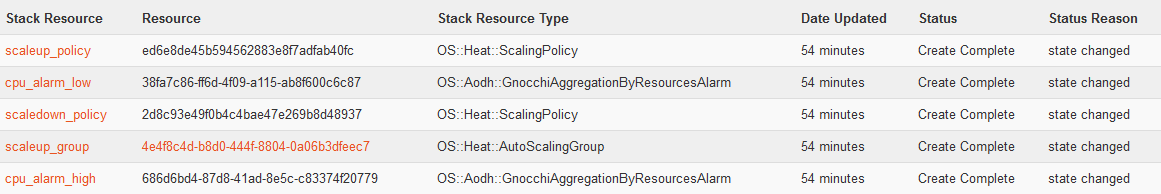
User vào giao diện web horizon, tab Orchestrasion > stack > launch stack



Nhập các biến đầu vào cho template:



Các tài nguyên của stack sau khi tạo xong:



**Kết luận chương 3.**

Chương 3 đã đưa ra các chi tiết về việc xây dựng nên hệ thống giám sát, sự kết hợp móc nối giữa các thành phần để thực hiện được cho thực hiện các nhu cầu chức năng của người dùng. Chi tiết về cách thiết lập chịu lỗi cho hệ thống: từ phát hiện, cảnh báo và tự động vượt qua lỗi và cách người sử dụng tạo một nhóm tài nguyên tự động co dãn theo các cảnh báo nó nhận được.

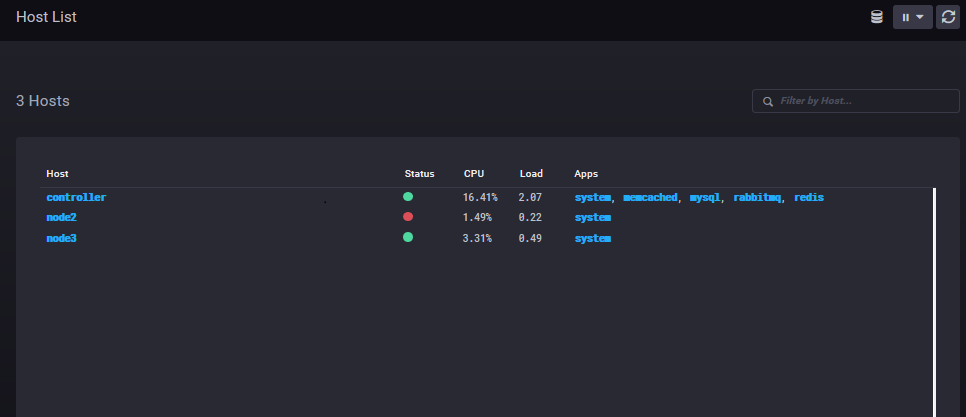
Chương 4 sẽ đi vào thử nghiệm hệ thống để xem kiểm tra những lợi ích mà hệ thống có thể mang lại.

# 

# **CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ**

## **4.1. Giám sát tài nguyên, dịch vụ máy chủ vật lý**

Cloud-provider vào giao diện quản trị tài nguyên vật lý, lưa chọn máy chủ cần theo dõi:



Hình 4.1 Người dùng quản trị xem danh sách máy vật lý

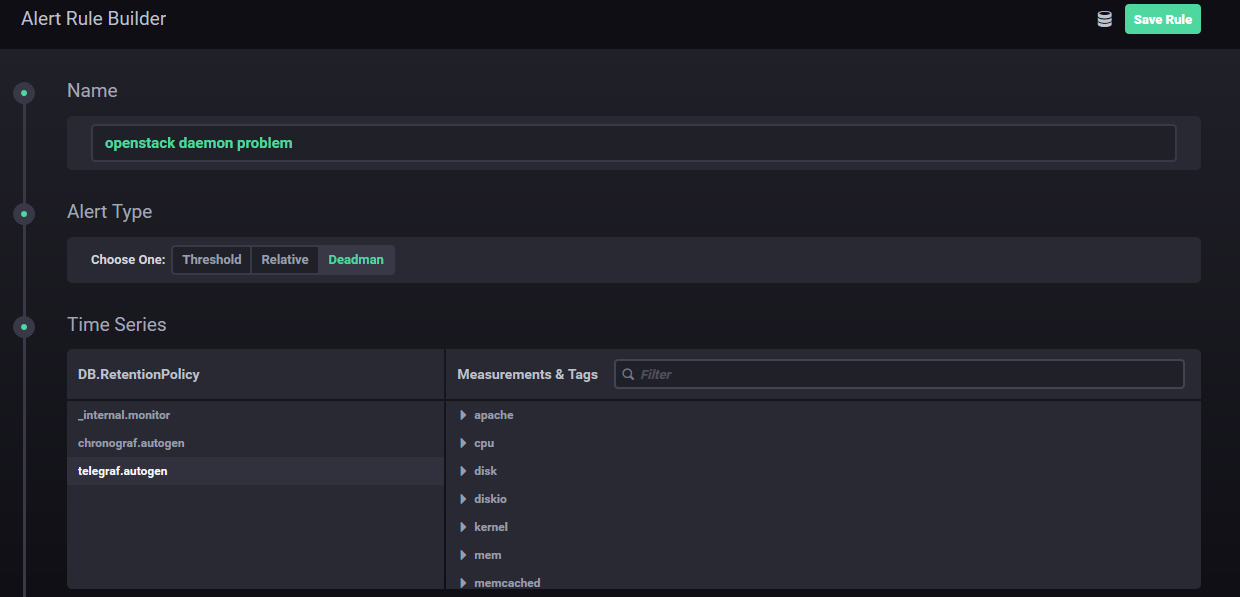
Cloud-provider chọn theo dõi tài nguyên máy vật lý



Hình 4. 2 Người dùng quản trị theo dõi tài nguyên vật lý

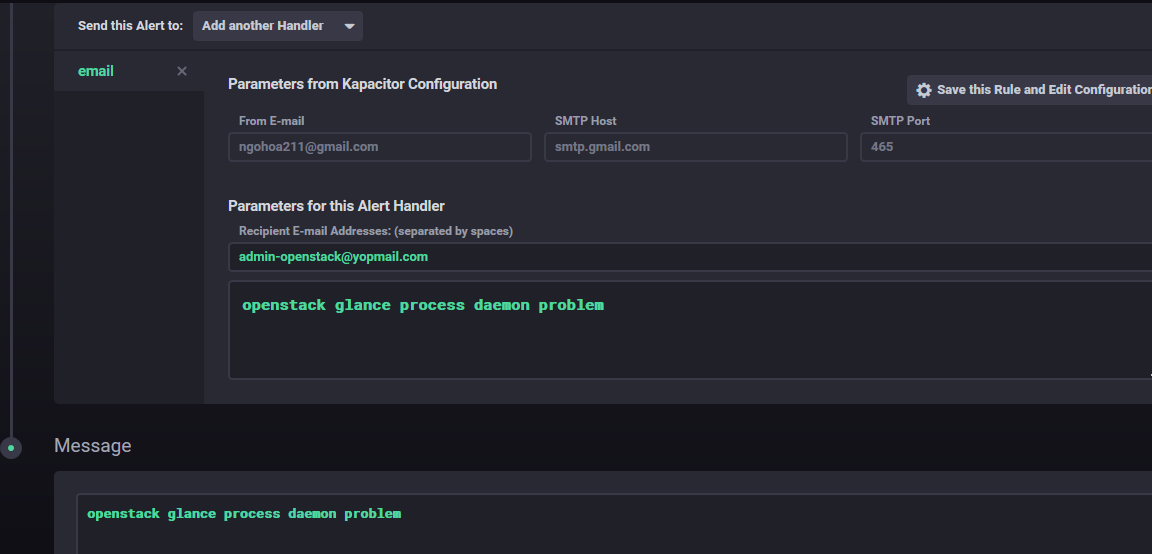
Cloud provider đặt luật cảnh báo khi dịch vụ openstack bị ngắt:

* Chọn tên cảnh báo “openstack daemon problem”
* Kiểu: deadman- tức không nhận được dữ liệu giám sát thì sẽ bật cảnh báo
* Chọn loại dữ liệu “system”



Hình 4. 3. Người dùng đặt cảnh báo trạng thái dịch vụ openstack

Người dùng đặt kệnh thông báo về email khi có cảnh báo được bật



Hình 4 4.Người dùng quản trị cấu hình kênh thông báo

**Thử nghiệm cảnh báo của hệ thống:**

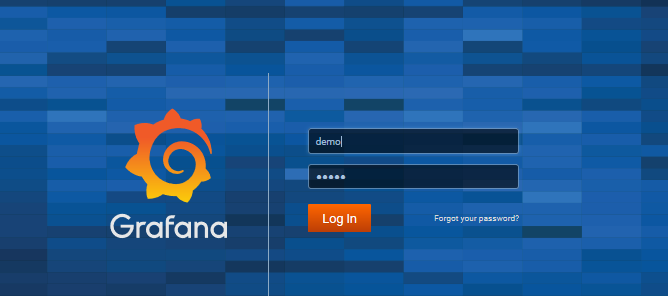
**Thực hiện tạo sự kiện:**

* Remote vào máy vật lý và tắt dịch vụ openstack glance

Kết quả -thành công: Có mail cảnh báo về dịch vụ openstack bị tắt

## **4.2. Giám sát tài nguyên ảo hóa**

Người dùng đăng nhập vào grafana:



Hình 4. 5.Người dùng đăng nhập giám sát máy ảo

Người dùng chọn xem tài nguyên trên tất cả các máy ảo



Hình 4. 6.Người dùng xem tài nguyên tất cả các máy ảo

Người dùng xem tài nguyên của một máy ảo cụ thể.



## **4.3. Kiểm tra khả năng chịu lỗi**

Ở đây ta kiểm tra các kết quả của việc thiết lập như mục 3.3.3

**Thử nghiệm.**

Thực hiên: Tắt một máy vật lý (compute node)

Kết quả -thành công: có cảnh báo máy vật lý tắt và tự động các máy ảo trên nó được di chuyển sang các máy còn lại

## **4.3. Kiểm tra khả năng tự động co dãn tài nguyên**

Ở đây ta kiểm tra các kết quả của việc thiết lập như mục 3.3.4

**Thử nghiệm**

Với stack gồm máy ảo được tạo ra, ssh vào máy ảo 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 cảnh báo cpu cao tải , hiển thị trên màn hình của người dùng, hệ thống sẽ phải phản ứng là thêm tài nguyên cho cụm.

* Kết quả -thành công: sau 60 s hệ thống sẽ tự động mở rộng stack thành 2 máy ảo

Với stack 2 máy ảo hiện tại, thực hiện tắt hết các tiến trình đang chạy của các máy , chỉ trừ các tiến trình nguồn của hệ thống.

* Kết quả -thành công: sau 60s hệ thống tự động rút nhỏ tài nguyên trong stack thành 1 máy ảo

# 

# **CHƯƠNG 5: KẾT LUẬN**

**Các kết quả đạt được**

Tìm hiểu sử dụng các phần mềm các mã nguồn mở: TICK và các mã nguồn mở openstack, để triển khai ra hệ thống giám sát cho môi trường đám mây. Hệ thống đáp ứng được những yêu cầu cơ bản trong việc quản trị cho một hệ thống đám mây. Thực hiện được các chức năng như tự động co dãn tài nguyên tương đối khó mà hiện nay tại việt nam ít nhà cung cấp cloud làm được.

Với cộng đồng openstack, khi thực hiện đồ án cũng đã đóng góp được 3 “commit” với hơn 1000 dòng mã nguồn thêm vào cho dự án Vitrage, liên quan đến việc tích hợp đẩy cảnh báo từ kapacitor vào vitrage.

(Xem tại <https://www.stackalytics.com/?release=all&user_id=ngohoa211&metric=commits> ).

**Các hạn chế cần được phát triển thêm**

Môi trường thử nghiệm còn rất nhỏ so với thực tế nên sẽ còn nhiều ca sử dụng trong việc giám sát hệ thống mà còn chưa nắm hết được .Các kịch bản cho phần cơ chế chịu lỗi cần bổ sung thêm từ các kinh nghiệm vận hành.

Hệ thống cần phát triển thêm để đảm bảo high aviability. Khả năng HA với các công nghệ hiện tại:

* Với TICK stack
  + Hiện tại influxDB, kapacitor hỗ trợ bản cluster nhưng chỉ với bản enterprise có trả phí
  + <https://www.influxdata.com/blog/influxdb-clustering/>
* Với telemetry:
  + Telemetry có khả năng hỗ trợ đầy đủ HA cho các microservice của mình. Tuy nhiên do tài nguyên môi trường lab không đủ để thử nghiệm.

Giao diện của người dùng lấy từ giao diện mặc định của openstack tương đối khó sử dụng. Không có một web chung thống nhất cho vận hành tất cả các hành động giám sát.

Chưa thử nghiệm các kênh cảnh báo cho người dùng như telegram, sms,...

# 

# **TÀI LIỆU THAM KHẢO**

[1] openstack community “ceilometer measurements document” xem tại <https://docs.openstack.org/ceilometer/latest/admin/telemetry-measurements.html>

[2] vitrage team “vitrage achitecture” xem tại <https://wiki.openstack.org/wiki/Vitrage>

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

[4] Influxdata comunity “influxdb getting-started ” xem tại <https://docs.influxdata.com/influxdb/v1.7/introduction/getting-started/>

[5] Influxdata company “install and configuration TICK stack ” xem tại <https://docs.influxdata.com/platform/installation/oss-install/>

[6] Openstack community “install and configuration ceilometer” xem tại <https://docs.openstack.org/ceilometer/latest/install/install-base-ubuntu.html>

[7] Openstack community “install and configuration aodh” xem tại <https://docs.openstack.org/aodh/latest/install/install-ubuntu.html>

[8] Openstack community “install and configuration vitrage” xem tại <https://docs.openstack.org/vitrage/latest/install/install-rdo.html#manual>

[9] Openstack community “install and configuration mistral” xem tại <https://docs.openstack.org/mistral/latest/install/installation_guide.html>

[10] Openstack community “install and configuration heat ” xem tại <https://docs.openstack.org/heat/pike/install/install-ubuntu.html>

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

[12] 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>

[14] Amazone company “Amazone Ec2 Auto Scaling” xem tại <https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html>