Mục lục

[**I. Tổng quan về Vitrage:** 1](#_Toc1578523)

[**1. Khái niệm:** 1](#_Toc1578524)

[2. Kiến trúc: 1](#_Toc1578525)

[3. Các luồng hoạt động cơ bản: 3](#_Toc1578526)

[II.Vitrage graph 6](#_Toc1578527)

[1. Hoạt động tạo graph 6](#_Toc1578528)

[2. Các loại datasouce: 7](#_Toc1578529)

[3. Cách cấu hình static datasource: 7](#_Toc1578530)

[4. Cách cấu hình extenal datasouce -zabbix: 11](#_Toc1578531)

[III. Vitrage template: 13](#_Toc1578532)

[1.Tổng quan về template 13](#_Toc1578533)

[2. Cú pháp template: 13](#_Toc1578534)

**I. Tổng quan về Vitrage:**

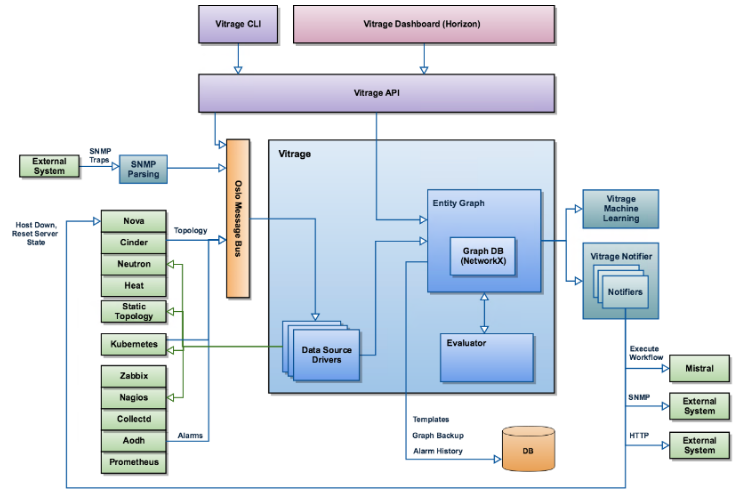
**1. Khái niệm:**

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.

Vitrage có các use-case tổng quan:

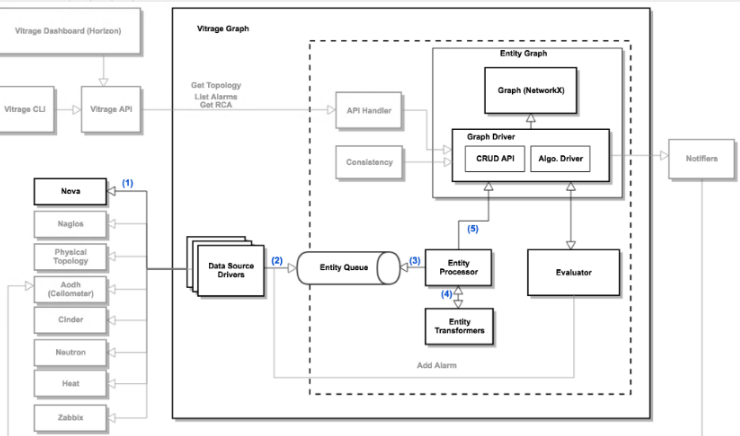
* Holistic and complete view: vitrage có thể biểu diễn mỗi quan hệ logic của các tài nguyên trong hệ thống thành grap+topology cho người xem, bao gồm cả thực thể virtual (instance, port, zone,..) hay physical (host , switch,..) kết nối với nhau. Để khi có một alarm xuất hiện từ một thành phần ta giải 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. Nhấn mạnh vào yếu tố “suy luận” để bật một deduced alarm :   
  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.  
  \*Trong mô hình HA cho một dịch vụ trên 3 VM mà cả 2 VM đều lỗi, thì vm còn lại cần bật warning  
  \*Bật warning trên vm khi thấy không đủ host để migrate nếu cần.
* 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.

### 2. Kiến trúc:

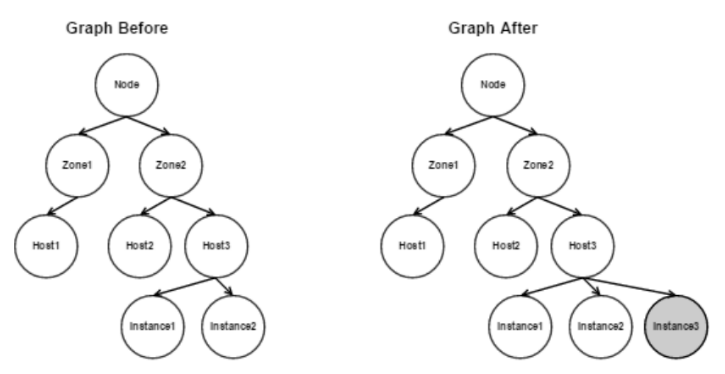


* Vitrage bao gồm các thành phần:
  + ClI, Drashboard, API: nhận yêu cầu người dùng, gọi đến các thành phần khác. Vitrage data source driver: Nhiệm vụ lấy input đầu vào thông tin về các resource hệ thống về cho vitrage graph hiển thị. Dữ liệu bao gồm thông tin như: resource có các thực thể nào? quan 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, doctor.  
    static datasource: đọc thông tin từ các file cấu hình người dùng cấp  
    ( trên là từ bản pike-queens, bản rocky có thêm kubernetes và prometheus)
  + 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.   
    Theo default khi mới cài đặt, chưa có plugin nào áp dụng.
  + 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.

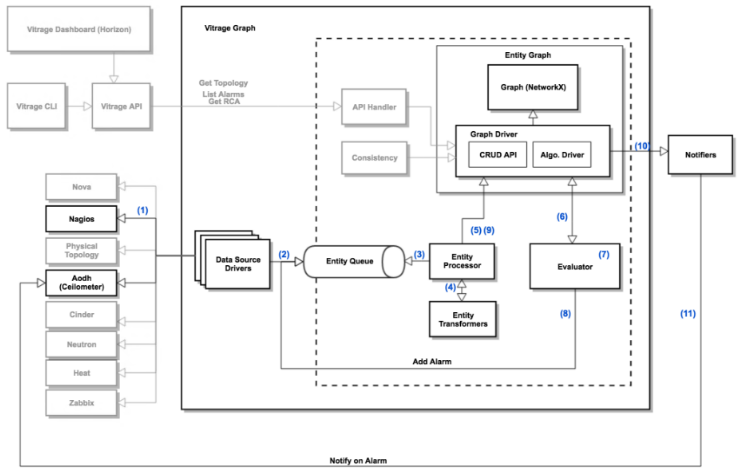
### 3. Các luồng hoạt động cơ bản:



* Trường hợp thêm một instance mới:
  + Nova tạo thêm instance , gửi thông báo lên rabbitmq topic “vitrage\_notifications”. Nova-datasouce driver nhận thông báo.
  + Nova-datasource Driver gửi thông tin vào vitrage entity queue ( bản chất queue này không phải message queue mà đơn giản là một mảng bình thường)
  + Entity processor polling event từ entity queue, xác định thông tin của thực thể cần thêm
  + Entity trasformers chuyển thông tin của thực thể về các đối tượng trong graph: đỉnh, cạnh, neghbors,...
  + Entity processor gọi graph api thêm đỉnh, cạnh mới.



* Trường hợp thêm có một alarm

  
Khi phát hiện switch down, vitrage nhận được alarm từ nagios. 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.

Alarm về bản chất cũng là 1 entity nên các bước từ 1 đến 5 cũng giống như use case trên. vd thêm 1 entiry có type alarm thông báo switch mà host 1 attach bị down

* + Khi có bất kỳ thay đổi nào về graph thì evaluator cũng được gọi đến. Evaluator biết được có 1 đỉnh mới được add
  + 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.
  + Evaluator push alarm vào entity queue.
  + Graph update thêm các alarm mới
  + Graph thêm thông báo rằng có 1 alarm mới trên instance, đẩy thông báo này vào messasge queue
  + Vitrage Notifier định nghĩa alarm cho aodh - set state “alarm”

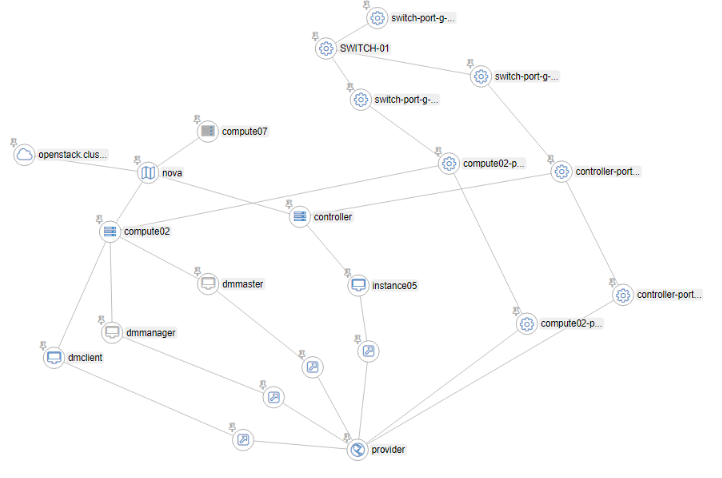
Ref:

<https://wiki.openstack.org/wiki/Vitrage>

<https://docs.openstack.org/vitrage/ocata/vitrage-use-cases.html>

## II.Vitrage graph

Với mục đích tổ chức resource, phân tích alarm của vitrage thì chức năng hiển thị ra graph là tính năng quan trọng nhất của vitrage. Graph được vitrage hiển thị có thể đưa ra mô hình các thực thể hệ thống, state + relation giữa các thực thể.



### 1. Hoạt động tạo graph

2 thành phần đóng góp hoạt động chính trong tạo graph này là vitrage datasouce-driver và vitrage graph.

Vitrage datasource-driver trách nhiệm :

* Thu thập thông tin của các service trong hệ thống (nova, neutron, cinder, heat,..), đưa ra được các thực thể (entity) và mối quan hệ (relationships) giữa chúng.  
  Mỗi thực thể có thể là :
  + một resource : host, volume, stack,...
  + một alarm : aodh alarm, zabbix alarm, nagios alarm   
    (alarm hay resoucre được phân biệt bởi thuộc tính "category" của thực thể.)
* Đồng thời cũng nó định nghĩa ra cách chuyển hóa thông tin này về các đổi tượng đỉnh, cạnh và gọi đến vitrage graph để hiển thị

Mỗi datasouce driver định nghĩa 2 class: Driver Class, Transformer Class

* Driver Class : Định nghĩa cách lấy thông tin về thực thể: resources (physical, virtual) ; alarms (Aodh, Nagios, Zabbix...) các sự cách thức lấy thông tin
* Transformer Class: Định nghĩa cách đưa được thông tin đó vào đồ thị, như: với mỗi sự kiện cần thêm, sửa xóa các đỉnh, cạnh như thế nào

Khi khởi tạo graph, datasouce driver gọi "get-all" - lấy về toàn bộ thông tin hệ thống hiện tại. Bằng cách tạo các client gọi đến api của mỗi service (nova, neutron,...)

Defaut cứ 10 phút, diver lại snapshot dữ liệu hiện tại và gọi get-all một lần

Khi có sự thay đổi trong hệ thống : vd thêm bớt host, có aodh alarm. Vitrage nhận biết bởi 1 trong 2 cách:

* polling mechanism: gọi đến hàm get\_changes - định nghĩa trong Driver Class
* pushing mechanism: đọc được messeage từ message queue mà service đó gửi lên, topic "vitrage\_notifications" ( chỉ có static datasouce driver dùng polling - tức sẽ đọc từ thư mục- hoặc trong database; các datasouce khác đều sử dụng pushing mechaism)

Xem thêm về cách viết 1 datasouce tại : <https://github.com/openstack/vitrage/blob/stable/queens/doc/source/contributor/add-new-datasource.rst> Vitrage graph có trách nhiệm:   
Kết nối với networkx để trình diễn đồ thị cho người dùng. Đồng thời nó cũng trả lời các truy vẫn đồ thị của khi các thành phần khác hỏi đến (vd vitrage-evaluator gọi khi thực hiện các kịch bản deduce-alarm; rca)

### 2. Các loại datasouce:

* openstack datasource: aodh, ceilometer, cinder, heat, neutron, nova.
* extenal monitor datasource: collectd, zabix, nagios, doctor.
* static datasource: đọc thông tin từ các file cấu hình người dùng cấp

Cách vitrage chuẩn hóa thông tin về trạng thái của các thực thể từ datasouce:

<https://github.com/openstack/vitrage/blob/stable/queens/doc/source/contributor/alarm-severity-config.rst>

<https://github.com/openstack/vitrage/blob/stable/queens/doc/source/contributor/resource-state-config.rst>

Các openstack datasource thì ta chỉ cần cấu hình oslo messaging notifications vào topic vitrage\_notifications; 2 nguồn còn lại thiết đặt có phức tạp hơn được đề cập ở dưới đây.

### 3. Cách cấu hình static datasource:

Mục đích thêm static datasource là giúp người dùng bổ sung vào đồ thị các thực thể chưa quản lý bởi bất kỳ datasouce nào khác, nó có thể mô tả cho đối tượng physical, virtual hoặc application vd: switch, route, service,...

Thêm vào /etc/vitrage/vitrage.conf:

[static]

directory = /etc/vitrage/static\_datasources/

changes\_interval = 10

Trong đó directory nơi chứa các file yaml khai báo static; changes\_interval 10s cho biết vitrage sẽ kiểm tra thay đổi trong thư mục 10s một lần

Template cho khai báo cho static datasouce có dạng:

metadata:

name: name template

description: description

definitions:

entities:

- static\_id: s1

type: switch

id: 12345

name: switch-1

state: active

...

- static\_id: h1

type: nova.host

id: 1

state: active

purpose: CI

...

relationships:

- source: s1

target: h1

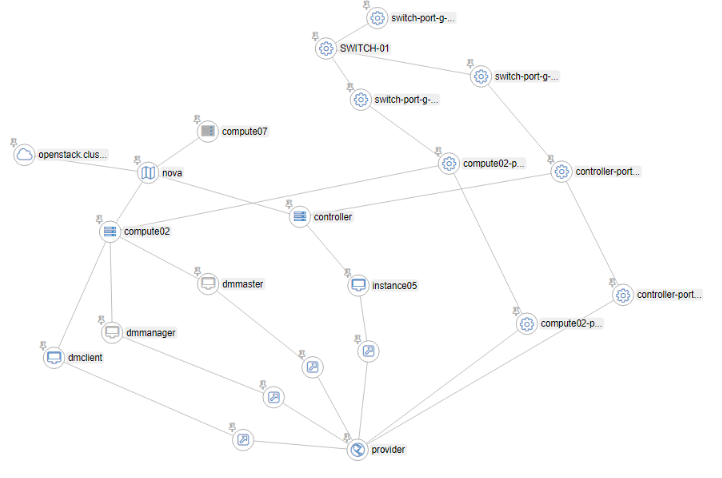
relationship\_type: attached

Mục metadata: khai báo thông tin của tài liệu

Mục definitions: khai báo các thực thể muốn thêm vào:

* **entities**: danh sách thực thể muốn thêm , trong đó:   
  **static\_id**: id dùng trong template, phạm vi nằm trong 1 template, mục đích như tên viết gọn   
  **type**: kiểu resource, có thể khai báo bất kỳ kiểu nào, có thể là kiểu có sẵn: openstack.cluster, nova.zone, nova.host, nova.instance, cinder.volume, neutron.network, neutron.port, heat.stack ; hay kiểu riêng tùy người dùng, vd port-switch, port-host,...   
  **id**: định danh, id và type cơ bản sẽ là ánh xạ 1 phần tử từ thế giới thực vào graph   
  **name**: tên resource   
  **state**: giá trị tùy chọn, nếu là resource đã có sẵn trong đồ thị ta không cần khai báo vào. Nhưng nếu đây lần đầu resource thêm vào thì buộc phải có, thường để giá trị là active hoặc avalable
* **relationships**: quan hệ giữa các thực thể :   
  **source**, **target** : static id của thực thể nguồn, đích   
  **relationship\_type**: có thể nhận bất kỳ giá trị nào. Thể hiện mối quan hệ 2 thực thể (hay nhãn của cạnh nối 2 đỉnh   
  Tuy nhiên để về sau viết template cho chính xác, relationship\_type thường liên quan đến type của source và target. Dưới đây là các quan hệ được quy ước ngầm của các datasoucre sẵn có của vitrage:
  + 'on' : aodh (alarm) on nova.instance.
  + 'contains' : openstack.cluster contains nova.zone ; nova.zone contains nova.host ; nova.host contains nova.instance ; neutron.network contains neutron.port
  + 'attached' : nova.instance attached cinder.volume ; nova.instance attached neutron.port
  + 'comprised': heat.stack comprised nova.instance ; heat.stack comprised cinder.volume ; heat.stack comprised neutron.network ; heat.stack comprised neutron.port   
    Để mô tả các entity vật lý : hiện có switch, port switch, port host, em có khai báo trong các file static datasource config các type entity:
  + switch : tương ứng với switch
  + port\_switch: port gắn với switch
  + port\_host : port vật lý gắn với host
  + port\_ovs : port virtual tạo với ovs   
    Một số relationship\_type em đặt tương ứng cho các static entity khi trên:
  + 'attached' : port\_switch attached switch ; port\_host attached nova.host ; port\_ovs attached neutron.network
  + 'connect' : port\_switch connect port\_host ; port\_host connect port\_ovs   
    (việc tách biệt từng loại port như vậy thấy sẽ có lợi tránh nhập nhằng khi viết template sau này)

Ví dụ add static cho mô hình như hình vẽ



Cluster có 2 host, được gắn vào cùng switch trên port enp4s0f0 , đồng thời mỗi port này đều kết nối vitrual port br-provider : Ta cấu static datasource như sau:

Thêm file [/etc/vitrage/static\_datasources/port\_host-controller.yaml](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/file/port_host-controller.yaml) : thêm static các port của host 1

Thêm file [/etc/vitrage/static\_datasources/port\_host-compute02.yaml](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/file/port_host-compute02.yaml) : thêm static các port của host 2

Thêm file [/etc/vitrage/static\_datasources/port\_switch-01.yaml](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/file/port_switch-01.yaml) : thêm static các port của switch

Thêm file [/etc/vitrage/static\_datasources/connect\_port.yaml](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/file/connect_port.yaml) : biểu thị mối quan hệ kết nối của các port host vào port switch

### 4. Cách cấu hình extenal datasouce -zabbix:

Mục đích là giúp vitrage có được các thông tin monitor từ các nguồn physical: port, host từ nguồn external datasource (zabbix) mang về

Cài đặt zabbix : <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-zabbix-to-securely-monitor-remote-servers-on-ubuntu-16-04>

Cấu hình mapping các alarm tại thực thể vật lý của zabbix vào thực thể trên graph:

* 1.Thiết lập monior port trên zabbix:
  + [Cấu hình zabbix để phát hiện một port của switch down](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/zabbix_conf/discover_switch_interface_down.md)
  + [Cấu hình zabbix để phát hiện một port của host down](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/zabbix_conf/discover_host_interface_down.md)
  + (mỗi port của switch cấu hình tương ứng 1 zabbix host, mỗi port của host cũng cấu hình thành 1 zabbix host)
* 2.[Cấu hình để khi triger zabbix bật lên sẽ gửi cảnh báo về vitrage ( gửi vào rabbitmq)](https://gitlab.visc.com/hoanq13/vitrage-doc/blob/master/zabbix_conf/config_zabbix_media.md)
  + Ref: <https://docs.openstack.org/vitrage/latest/contributor/zabbix_vitrage.html>
* 3/Thiết lập tại vitrage
  + Thêm vào file /etc/vitrage/vitrage.conf:
* [datasources]
* snapshots\_interval = 300
* # Names of supported plugins (list value)
* types = zabbix,nova.host,nova.instance,nova.zone,static,aodh,cinder.volume,neutron.network,neutron.port,heat.stack
* [zabbix]
* url = http://<ip>/zabbix
* password = zabbix
* user = admin
* config\_file = /etc/vitrage/zabbix\_conf.yaml
  + Mapping các thực thể của zabbix quản lý vào các static resource entity vitrage đã cấu hình .  
    Vd: theo như mô hình như trên hình , ta mapping các port của host, switch zabbix monitor vào các static datasource . Tại zabbix đã tạo đủ các host:

Thêm file zabbix\_conf.yaml:

cat /etc/vitrage/zabbix\_conf.yaml

zabbix:

- zabbix\_host: controller port enp4s0f0

type: port

name: controller-port-enp4s0f0

- zabbix\_host: compute02 port enp4s0f0

type: port

name: compute02-port-enp4s0f0

- zabbix\_host: Interface Gi1-0-21

type: port

name: switch-port-g-1-0-21

- zabbix\_host: Interface Gi1-0-20

type: port

name: switch-port-g-1-0-20

- zabbix\_host: Interface Gi1-0-22

type: port

name: switch-port-g-1-0-22

## III. Vitrage template:

### 1.Tổng quan về template

Trong vitrage, 1 template như một kịch bản cho công việc add deduce-alarm, root-cause-indicatior khi có bất kì event nào tác động đến graph (create/update/delete vertex/edge) . Vitrage -evaluator sẽ tìm kiếm trong graph các pattern mà có thể áp dụng được điều kiện thấy trong các template để áp dụng (do-action). Nếu một graph không còn thỏa mãn điều kiện đó, nó sẽ quay trở về trạng thái trước đó (undo-action)

Cách thức evaluator làm việc để chạy được chức năng này có thể xem thêm tại Evaluator Design [[1]](#footnote-1)

Template được người quản trị viết dạng file yaml, tùy vào phiên bản mà được load vào hê thống bằng các cách:

* pike : Để các template vào thư mục /etc/vitrage/templates, restart lại dịch vụ
* queen: Chạy lệnh command line :
  + vitrage template validate --type standard --path /etc/vitrage/templates/basic.yaml

Để kiểm tra cú pháp file đúng chưa.

* + vitrage template add --type standard --path /etc/vitrage/templates/basic.yaml

Để load template vào hệ thống.

* rocky: Sử dụng được giao diện web, chọn tab templates > add template

Khi một template được load mới. nó cũng được evaluator xem xét ngay có kịch bản nào được không.

### 2. Cú pháp template:

|  |
| --- |
| metadata:  name: <unique template identifier>  description: <what this template does>  definitions:  entities:  - entity:  template\_id:  category:  …  - entity:  template\_id:  category:  …  relationships:  - relationship:  template\_id :  source:  target:  relationship\_type:  - relationship:  …  scenarios:  - scenario:  condition: <if statement true do the action>  actions:  - action:  action\_type:  properties:    action\_target:  - action:  … |

Phần metadata mô tả về kịch bản áp dụng, nhận giá trị bất kỳ string nào, tuy nhiên metadata.name cần là duy nhất trong tất cả file templates

Phần definition -khai báo:

* entities: mô tả về thực thể cần tìm, cần khai báo các thuộc tính:
  + category: RESOURCE hay ALARM
  + template\_id: id trong template, sẽ được reference tại phần khai báo relationship

Các thuộc tính tùy chọn hay dùng

* + type: loại datasource - với thực thể thuộc loại:

|  |  |  |  |
| --- | --- | --- | --- |
|  | openstack-data source | extenal-monitor-datasource | static-datasource |
| “type” có thể khai báo: | aodh |  cinder.volume | heat.stack | neutron.stack | neutron.port |  nova.host |  nova.zone | nova.instance | nagios |  zabbix | bất kỳ giá trị nào:  port\_host, port\_switch, switch, app1,… |

* + name, id : khai báo giá trị ứng với entity đó trong thực tế.
* relationship: mô tả quan hệ - theo từng cặp source-target giữa các thực thể : một relationship thường có ý nghĩa mô tả một điều kiện . vd : alarm\_on\_instance.
  + template\_id : id trong template, được refrence tại phần scenario.condition
  + source: template\_id của entity source
  + target: template id của entity đích
  + relationship\_type: mối quan hệ, phụ thuộc vào entity type mà có thể có quan hệ tương ứng, gồm:

|  |  |  |  |
| --- | --- | --- | --- |
| **stt** | **entity type:** | | **relationship\_type** |
| **source** | **target** |
| 1 | openstack.cluster | nova.zone | contains |
| 2 | nova.zone | nova.host | contains |
| 3 | nova.host | nova.instance | contains |
| 4 | nova.instance | cinder.volume | attached |
| 5 | nova.instance | neutron.port | attached |
| 6 | neutron.network | neutron.port | contains |
| 7 | neutron.port | nova.instance | attached |
| 8 | heat.stack | nova.instance | comprised |

Với các datasouce static, relationship\_type cần tương ứng với khi ta khai báo các static datasource (tại /etc/vitrage/static\_datasources/ )

Phần scenarios: định nghĩa các kịch bản áp dụng:

* scenario.condition: mô tả điều kiện áp dụng, viết dưới dạng and, or các relationship , entity.
* scenario.action: mô tả hành động sẽ được áp dụng, bao gồm có:
  + raise\_alarm, set\_state, add\_causal\_relationship , mark\_down

vd: raise\_alarm:

|  |
| --- |
| action:  action\_type : raise\_alarm  properties:  alarm\_name: some problem  severity: critical  action\_target:  target: instance |

trong đó properties.severity của alarm gồm có các giá trị

CRITICAL > SEVERE > WARNING > N/A > OK

Set\_state:

|  |
| --- |
| action:  action\_type : set\_state  properties:  state: error  action\_target:  target: host |

trong đó: properties.state gồm có :

DELETED > ERROR > SUBOPTIMAL > TRANSIENT > OK

add\_causal\_relationship:

|  |
| --- |
| action:  action\_type : add\_causal\_relationship  action\_target:  source: host\_alarm  target: instance\_alarm |

**Ví dụ** : tìm tất cả entity bị gắn alarm critical để áp dụng set\_state về error

|  |
| --- |
| metadata:  id: deduced\_state\_for\_all\_with\_alarm  description: deduced state for all resources with alarms  definitions:  entities:  - entity:  category: RESOURCE  template\_id: a\_resource  - entity:  category: ALARM  severity: critical  template\_id: high\_alarm  relationships:  - relationship:  source: high\_alarm  target: a\_resource  relationship\_type: on  template\_id : high\_alarm\_on\_resource  scenarios:  - scenario:  condition: high\_alarm\_on\_resource  actions:  - action:  action\_type : set\_state  properties:  state: error  action\_target:  target: a\_resource |

**Reference**

https://docs.openstack.org/vitrage/queens/contributor/vitrage-template-format.html

1. https://docs.openstack.org/vitrage/pike/contributor/scenario-evaluator.html [↑](#footnote-ref-1)