brant

2017-07-31始 周记录

Bonan Ruan

Notes in Configuration of Congress with Doctor:

- http://docs.opnfv.org/en/stable-danube/submodules/doctor/docs/release/configguide/feature.configuration.html
- https://docs.openstack.org/congress/latest/cloudservices.html#driver-installation

Before

```
openstack congress datasource create doctor doctor
```

we need to install doctor driver to congress:

(commands below should be executed on every controller node server)

```
sudo -i
vim /etc/congress/congress.conf
# then find the `drivers=...` entry
# append `congress.datasources.doctor_driver.DoctorDriver` to it
service openstack-congress-server restart
```

After

```
openstack congress datasource create doctor doctor
```

add policies:(in undercloud with overcloudrc)

```
openstack congress policy rule create \
    --name host_down classification \
    'host_down(host) :-
        doctor:events(hostname=host, type="compute.host.down", status="down")'

openstack congress policy rule create \
    --name active_instance_in_host classification \
    'active_instance_in_host(vmid, host) :-
```

```
        nova:servers(id=vmid, host_name=host, status="ACTIVE")'

openstack congress policy rule create \
    --name host_force_down classification \
    'execute[nova:services.force_down(host, "nova-compute", "True")] :-
        host_down(host)'

openstack congress policy rule create \
    --name error_vm_states classification \
    'execute[nova:servers.reset_state(vmid, "error")] :-
        host_down(host),
        active_instance_in_host(vmid, host)'

# show policy rule

openstack congress policy rule show classification RULE-ID
openstack congress policy rule show action RULE-ID
```

Congress Usage:

```
openstack congress policy list
openstack congress datasource list
openstack congress –help
openstack congress datasource schema show nova
```

Datalog Language(to express Policy):

```
# e.g. 1
# More than one ip mapped to one port_id is not permitted
error(port_id, ip1, ip2) :- port(port_id, ip1), port(port_id, ip2), not
equal(ip1, ip2)

# e.g. 2
## monitor
error(vm, network) :-   nova:virtual_machine(vm)   nova:network(vm,
network)   nova:owner(vm, vm_owner)   neutron:owner(network,
network_owner)   not neutron:public_network(network)   not
same_group(vm_owner, network_owner)
same_group(user1, user2) :-   ad:group(user1, group)   ad:group(user2, group)
## to correct
execute[neutron:disconnectNetwork(vm, network)] :-   error(vm, network)

# Usage
## ground atom
<tablename>(arg1, ..., argn)
```

```
### e.g.
neutron:port_ip("66dafde0-a49c-11e3-be40-425861b86ab6", "10.0.0.1")
neutron:port_ip("66dafde0-a49c-11e3-be40-425861b86ab6", "10.0.0.2")
neutron:port_ip("73e31d4c-e89b-12d3-a456-426655440000", "10.0.0.3")
## rules
### if-then `if` is the head; `then` is the body

To create a new table out of an existing table, we write Datalog rules. A rule
is a simple if-then statement, where the if part is called the head and the
then part is called the body.  The head is always a single Datalog atom.  The
body is an AND of several possibly negated Datalog atoms. OR is accomplished
by writing multiple rules with the same table in the head.

### e.g.
has_ip(x) :- neutron:port_ip(x, y)
### e.g. (AND operator)
same_ip(port1, port2) :- neutron:port_ip(port1, ip), neutron:port_ip(port2,
ip)
### e.g. (NOT operator)
no_ip(port) :- neutron:port(port), not has_ip(port)
### e.g. (OR operator)
group(user, grp) :- ad:group(user, grp)
group(user, grp) :- keystone:group(user, grp)

## builtins
### gt (>)
plenty_of_memory(vm) :- nova:virtual_machine.memory(vm, mem), gt(mem, 100)
### simplify
port(id) :- neutron:ports(id, tenant_id, name, network_id, mac_address,
admin_state_up, status, device_owner, fixed_ips, security_groups)
#### that can be written as
port(x) :- neutron:ports(id=x)

## execute
execute[nova:servers.pause(x)] :- nova:servers(id=x, status="ACTIVE")
```

Grammars of Datalog:

```
<policy> ::= <rule>*
<rule> ::= <head> COLONMINUS <literal> (COMMA <literal>)*
<head> ::= <atom>
<head> ::= EXECUTE[<atom>]
<literal> ::= <atom>
<literal> ::= NOT <atom>
<atom> ::= TABLENAME LPAREN <arg> (COMMA <arg>)* RPAREN
<arg> ::= <term>
<arg> ::= COLUMNNAME=<term>
<term> ::= INTEGER | FLOAT | STRING | VARIABLE
```

Datalog builtins:

```
Comparison Builtin  Inputs  Description
lt(x, y)  2  True if x < y
lteq(x, y)  2  True if x <= y
equal(x, y)  2  True if x == y
gt(x, y)  2  True if x > y
gteq(x, y)  2  True if x >= y
max(x, y, z)  2  z = max(x, y)

Next are the arithmetic builtins.
Arithmetic Builtin  Inputs  Description
plus(x, y, z)  2  z = x + y
minus(x, y, z)  2  z = x - y
mul(x, y, z)  2  z = x * y
div(x, y, z)  2  z = x / y
float(x, y)  1  y = float(x)
int(x, y)  1  y = int(x)

Then are the string builtins.
String Builtin  Inputs  Description
concat(x, y, z)  2  z = concatenate(x, y)
len(x, y)  1  y = number of characters in x

Next are the builtins for manipulating dates and times. These builtins are
based on the Python DateTime object.
Datetime Builtin  Inputs  Description
now(x)  0  The current date-time
unpack_date(x, year, month, day)  1  Extract year/month/day
unpack_time(x, hours, minutes, secs)  1  Extract hours/minutes/seconds
unpack_datetime(x, y, m, d, h, i, s)  1  Extract date and time
pack_time(hours, minutes, seconds, x)  3  Create date-time with date
pack_date(year, month, day, x)  3  Create date-time with time
pack_datetime(y, m, d, h, i, s, x)  6  Create date-time with date/time
extract_date(x, date)  1  Extract date obj from date-time
extract_time(x, time)  1  Extract time obj from date-time
datetime_to_seconds(x, secs)  1  secs from 1900 to date-time x
datetime_plus(x, y, z)  2  z = x + y
datetime_minus(x, y, z)  2  z = x - y
datetime_lt(x, y)  2  True if x is before y
datetime_lteq(x, y)  2  True if x is no later than y
datetime_gt(x, y)  2  True if x is later than y
datetime_gteq(x, y)  2  True if x is no earlier than y
datetime_equal(x, y)  2  True if x == y

Last are the builtins for handling network addresses. These builtins are based
on the Python netaddr package. Both IPv4 and IPv6 are supported. For more
```

```
details see the netaddr documentation <http://pythonhosted.org/netaddr/>.
Network Address Builtins  Inputs  Description
ips_equal(x, y)  2  True if IP x is equal to IP y
ips_lt(x, y)  2  True if IP x is less than IP y
ips_lteq(x, y)  2  True if IP x is less than or equal to IP y
ips_gt(x, y)  2  True if IP x is greater than IP y
ips_gteq(x, y)  2  True if IP x is greater than or equal to IP y
networks_equal(x, y)  2  True if network x and network y are equal
networks_overlap(x, y)  2  True if the same IP is in networks x and y
ip_in_network(x, y)  2  True if IP x belongs to network y
```

https://docs.openstack.org/congress/latest/enforcement.html

```
$ openstack congress policy create reactive
$ openstack congress policy rule create reactive
'execute[nova:servers.pause(x)] :- nova:servers(id=x, status="ACTIVE")'
```

Congress with Zabbix:

Congress mainly contained:

- Datasource Driver
- Policy Engine
- Policy
- API

In https://github.com/openstack/congress/blob/master/congress/datasources/doctor_driver.py we can see the RESTful API:

```
PUT /v1/data-sources/doctor/tables/events/rows
```

And if you see https://openstack.nimeyo.com/64169/openstack-dev-congress-summit-recap you may find that Zabbix can push data to Congress through the API above.

With

```
openstacak congress datasource table schema show doctor events
```

you can see the body of `event`:

```
id
time
type
hostname
status
monitor
monitor_event_id

For updating 'events' table, the request body should be following
style. The request will replace all rows in the table with the body,
which means if you update the table with [] it will clear the table.
One {} object in the list represents one row of the table.

request body:
[
     {
          "time": "2016-02-22T11:48:55Z",
          "type": "compute.host.down",
          "details": {
               "hostname": "compute1",
               "status": "down",
               "monitor": "zabbix1",
               "monitor_event_id": "111"
          }
     },
.....
]
```

Then, how to use Openstack API? Here's one example:

```
# find the service's endpoint
openstack endpoint show congress
## assume we get 192.168.37.12:1789
openstack endpoint show keystone
## assume we get 192.168.37.12:5000/v2.0
```

Firstly, ask authentication:

```
curl -i 'http://192.168.37.12:5000/v2.0/tokens' -X POST -H "Content-Type:
application/json" -H "Accept: application/json"  -d '{"auth": {"tenantName":
"admin", "passwordCredentials": {"username": "admin", "password": "xxxx"}}}'
```

then you get one token

```
{"access": {"token": {"issued_at": "2017-08-02T08:44:02.000000Z", "expires":
"2017-08-02T09:44:02Z", "id": "7824347b5d6947818e13b61c6364aa46",
"tenant":...
```

then use the token to visit Congress:

```
curl -i -X GET http://192.168.37.12:1789/v1/policies  -H "X-Auth-Token:
7824347b5d6947818e13b61c6364aa46"
```

For more operations about Congress API, see
https://docs.openstack.org/congress/latest/api.html.
How can Zabbix send message to Congress?

https://www.zabbix.com/documentation/3.2/manual/config/notifications/media/script

`man curl` may help (search for `PUT`).

Architecture:

```
Zabbix -data-> Congress -policy_act_change_state-> Nova/Neutron/Cinder -
notification-> Ceilometer -event-> Aodh -alarm-> DoctorManager(juju?) -action-
> Nova
```

Nova Notification:

Our nova version is 14.0.8, wich may be in the Newton Openstck. So see (1)
https://docs.openstack.org/newton/config-reference/compute/config-options.html
and (2)https://docs.openstack.org/oslo.messaging/latest/configration/opts.html#oslo-messaging-notifications for details.
To send `compute.instanc.update` notification, my configuration in `/etc/nova/nova.conf` is:

```
notify_on_state_change=vm_and_task_state
[oslo_messaging_notifications]
driver=messaging
```

In (2) we know the default `topic` is `notifications`. And the details about `compute.instance.update` notification can be found in
https://docs.openstack.org/nova/14.0.7/notifications.html which may be sth like this:

```
{
    "event_type": "instance.update",
    "payload": {
        "nova_object.data": {
```

```
                ...
                "state_update": {
                     "nova_object.data": {
                          "new_task_state": null,
                          "old_state": "building",
                          "old_task_state": null,
                          "state": "building"
                     },
                     ...
                },
                ...
           },
      ...
}
```

Also, you can see more here: https://docs.openstack.org/newton/config-reference/telemetry/samples/event_definitions.yaml.html

In `event_pipeline.yaml`:

```
---
sources:
     - name: event_source
       events:
           - "*"
       sinks:
           - event_sink
sinks:
     - name: event_sink
       transformers:
       publishers:
           - notifier://
           - notifier://?topic=alarm.all
```

Aodh Alarm:

https://docs.openstack.org/aodh/latest/contributor/event-alarm.html

https://docs.openstack.org/aodh/latest/admin/telemetry-alarms.html

https://docs.openstack.org/aodh/latest/admin/telemetry-alarms.html#event-based-alarm

Create alarm:

```
aodh alarm create \
  --type event \
  --name instance_update \
  --description 'Instance update' \
  --event-type "compute.instance.update" \
  --enable True \
  --query "traits.state=string::error" \
  --alarm-action 'http://xxx.xxx/xxx' \
  --ok-action 'log://' \
  --insufficient-data-action 'log://'

# we can also use
--query "traits.instance_id=string::INSTANCE_ID;traits.state=string::error"
# to set more limitations

# to disable
aodh alarm update --enabled False ALARM_ID
# to delete
aodh alarm delete ALARM_ID
```

Alarm action details:

The document has not go into details about the POST action, but we can see more in code:

https://github.com/openstack/aodh/blob/master/aodh/notifier/rest.py

```
body = {
     'alarm_name': alarm_name,
     'alarm_id': alarm_id,
     'severity': severity,
     'previous': previous,
     'current': current,
     'reason': reason,
     'reason_data': reason_data
}
headers['content-type'] = 'application/json'
kwargs = {'data': jsonutils.dumps(body), 'headers': headers}
```

So with `--alarm-action 'http://xxx.xxx/xxx'`, what we need to do is find out the MANO's relative API. And the `aodh-notifier` will help us to fill the request body and send it.

Learn much from https://lingxiankong.github.io/2017-07-04.html

Zhian Hou

Yuan Zang