Congress Workflow (Push Model)
Authors: Jun (Jim) Xu and Yapeng Wu

1. Use case

This is a typical application developer user case[1]: 2-tier PCI enterprise application (database tier and web tier). This application can be deployed either for production or for development. In enterprise there are 3 type role will use/deploy this application: Application Developer, Cloud Operator and Compliance Officer.

For application developer, when deployed for production, it needs:
- Solid-state storage for the DB tier
- All ports but 80 closed on the web tier
- No network communication to DB tier except from the web tier
- No VM in the DB tier can be deployed on the same hypervisor as another VM in the DB tier; same for the web tier.

For Cloud operator, it needs:
- Applications deployed for production must have access to the internet
- Applications deployed for production must not be deployed in the DMZ cluster.
- Applications deployed for production should scale based on load.
- Applications deployed for development should have 1 VM instance per tier.
- Every application must use VM images signed by an administrator

For Compliance officer, it needs
- No VM from a PCI app may be located on the same hypervisor as a VM from a non-PCI app.

2. Congress Policy Program

Start from the use case, Congress will provide the basic framework to enable the description of the requirement from the use case, with the specified policy language. As of the current discussion, Datalog is the language which is evaluated as a human consumable and yet easy machine implementable language. Other languages might be considered or Datalog itself might be finalized along with the Congress project. The Evaluation of the policy language however is beyond the scope of this specification. Within this workflow specification, it will use Datalog as one language to illustrate the workflow with example.

3. Push Model Workflow

In the Push Model, it assumes Congress being the source of true, with Congress be the central component to declare, enforce, monitor and audit the policy. Certainly, it should communicate with the underneath components, as Neutron, Nova, for collecting resource status, network information, and implementing specific policy via policy driver.

Start from the above use case, assume DB tier VMs and WEB tier VMs must be on different hypervisor. A policy snippet can be written in Congress as following for the requirement:

```
Error(vm):-
app(A)
app.tier(A, tier1),
app.tier(A, tier2),
tier.type(tier1, DB),
tier.type(tier2, WEB),
  nova:virtual_machine(vm),
congress:tier(vm, is_DB, db_vm),   //locate the vm which is in tier DB
congress:tier(vm, is_WEB, web_vm),  //locate the vm which is in tier WEB
not same_hypervisor(db_vm, web_vm)

same_hypervisor(vm1, vm2):-
        nova:host(vm1, host),
        nova:host(vm2, host)
```

Upon the definition of the information, Congress will parse the code, and build cloud service tables:

```
Congress:tier
-------------
VM      Tier
-------------
vm1  DB
vm2  WEB
vm3  WEB
```

The number of VMs depends on the input from the user requirement explicitly, from the load/usage and VM capability implicitly. After this table being build, Congress will send message via the Nova driver to Nova as it specified in the code, and instrument Nova to provision the corresponding VMs. Upon successful provisioning, Congress will maintain the following VM tables:

```
Nova:network
---------------
VM      tennant
-----------------
```

Vm1 A (or company A)
Vm2 A
Vm3 A

All the newly added VM should have an entry in the Nova:network (tenant) table. Certainly upon query the Nova, billing and accounting policy can apply with the information from this table as well.

Nova:host

VM Host

Vm1 host1:UUID
Vm2 host2:UUID
Vm3 host3:UUID

If VM is moved or deleted, the table should be updated as well, with either subscribing to Nova component, or via periodic query into Nova.

In an overly simplified situation, it assumes that the Congress compute policy will map to the tables and the Congress Nova Drive can directly instrument Nova by using Nova API. It will also assume that Congress will specify the the WEB VM and DB VM in different host by passing the policy to Nova scheduler.

To complete the above use case, a network related policy snippet can be added in the policy snippet in Congress:

```
error(vm):-
app(A)
app.tier(A, tier1),
app.tier(A, tier2),
tier.type(tier1, DB),
tier.type(tier2, WEB),
  nova:virtual_machine(vm),
congress:tier(vm, is_DB, db_vm),   //locate the vm which is in tier DB
congress:tier(vm, is_WEB, web_vm),  //locate the vm which is in tier WEB
not same_hypervisor(db_vm, web_vm)

same_hypervisor(vm1, vm2):-
        nova:host(vm1, host),
        nova:host(vm2, host)

Nova:network(web_vm, network),
  neutron:network(network, packet),
```

packet:dst_port(packet, port),
not port(port, http) //admin can use default 80 as the http port, or its own port

With the introducing of network policy along with existing compute policy in the code in Congress, Congress will need to be able to provide precise and necessary information to different policy drivers from different policy segment. In the above policy snippet, Congress with its Neutron Driver will need to identify the two tiers, map it to the two End Point Group (EPG) in Neutron Group Based Policy Model, and send to Neutron. (For more information about the EPG, please refer to …)

Congress Policy Driver to Neutron

EPG provides an abstraction of topology representation in the network level, without application to manage or contracture the physical network plumbing or changes. In comparison, Congress as a topology independent language provides further abstraction, with no specific description of the network elements and corresponding topology. Still as the policy language with its policy and complete sources from underneath components, Congress can render the EPG policy rules.

Back to the use case, Congress will locate all the VMs belonging to network 'A'. Further it will identify the two tiers Tier1 and Tier2 in the network 'A'. In mapping to EPG, Congress will maintain the followin table:

Neutron:EPG

VM EPG
---
vm1 EPG1
vm2 EPG2
vm3 EPG2

Further Congresss will push the following information to Neutron via its policy driver interface:
neutron epg-create EGP2
neutron ep-add vm1 --endpoint-group EPG1
neutron epg-create EGP1
neutron ep-add vm2 --endpoint-group EPG2

In theory, there could be many rules per policy contract in EPG. Back the above use case, there is only one rule in the EPG policy contract. Upon complete parsing the Congress policy snippet, Neutron GBP will receive the following EPG policy for Neutron to enforce/implement the network side policy:
neutron classifier-create web-access --port <80 or http>
neutron policy-rule web --policy-classifier web-access --actions allow

neutron contract-create web-contract --policy-rule web

And Congress will ask Neutron to enforce that EPG policy on the VMs in the web tiers, EPG2:
neutron epg-create EGP2 --provides-contract web-contract

Neutron Policy Manager should take the EPG policy and enforce it within Neutron.

4. Cloud Service provisioing, Resource Allocation, and Error Detection

As described above, Congress will parse the policy snippet, collect compute/storage/network resource, then send policys to different cloud service components to provision the resource to the targeted status.

If there are resource updates, or policy violation triggers, Congress should receive the updates, and take defined actions to handle the updates. In the above use case, if any external traffic send to one of the web tier VMs as not the HTTP/80 port, neutron should detect the local policy violation and report it to Congress Error handling service (one of Congress reserved services).

Reference:
[1] Congress Juno Mid-Cycle Meetup Etherpad:
https://etherpad.openstack.org/p/juno-midcycle-policy-summit