Deprecated: Congress User Stories/Use Cases

http://goo.gl/RM3W6W

Background

This document contains user stories currently targeted for the OpenStack Congress policy framework. These stories are meant to be early work around what are the priorities for the Juno release cycle. One or more blueprints will result from each user story. See https://wiki.openstack.org/wiki/Congress#How_To_Create_a_Blueprint for the process.

Template User Story

This is a summary of the policy

Data Sources

- A list of data source plugins (OpenStack/other components) needed

Policy

- A list of datalog rules that represent the policy

Policy Actions Needed

What type of action does Congress need to take for this use case? Examples:

- Monitoring
- Proactive: Responding to queries from other components about whether a given API call will cause new violations.
- Reactive: Computing/executing actions that bring the cloud back into compliance
- Assistive: Filling out missing fields of an API call that make the call consistent with policy.
- Sub-policy: Pushing policy down to another service

+5 Public/private networks with group membership

votes: +1 thinrichs, pballand, rajdeepd, sarob, banix

There are a plethora of restrictions that cloud operators may want to place on how VMs are connected to networks. For example, suppose we want to ensure that every network a VM is connected to is either a public network or the owner of the network and the other of the VM belong to the same ActiveDirectory/Keystone/etc. group.

Data sources

- Neutron: the list of public networks and the owner of each network
- Nova: the list of networks connected to VMs and the owners of those VMs
- ActiveDirectory/Keystone: which users are members of which groups

Policy

```
error(vm) :-
        nova:instance(vm),
        nova:network(vm, network),
        not neutron:public(network),
        nova:owner(vm, vmowner),
        neutron:owner(network, netowner),
        not same_group(vmowner, netowner)
same_group(x,y) :-
        group(x,g),
        group(y,g)
group(x,g) :-
        keystone:group(x, g)
group(x,g) :-
        ad:group(x,g)
```

Policy Actions

- Monitoring (Could also illustrate proactive/reactive actions).

+2 Upgrade of administrative software

votes: +1 sarob, +1 thinrichs (if we get the datasources crystalized)

Every OpenStack release cycle, new stable OpenStack packages will be released through the various distributions. As these become available, some will want to stage and deploy into production as soon as possible. This policy would make the upgrade process reactive to the new packages being available.

Datasources

- Keystone (?): current software versions
- ??: service that tells us latest released software versions

- Upgrade service: something that we can tell to upgrade a given piece of software.

Policy

```
error(software) :-
        current_version(software, current),
        latest_version(software, latest),
        // assuming that current/latest are simple numbers so that
       // we can use kudva's builtins
        less_than(current, latest)
```

Policy Actions

- Monitoring
- Reactive

proposed features (pre-spec/blueprints details):

1) openstack upgrade policy

- problem description:
- proposed change/how to solve:
- alternatives:
- data model impact: TBD
- API impact: TBD
- security impact: TBD
- implementation plan: TBD
- dependencies: TBD

+0 Evacuation of tenants for planned outage

votes:

As part of any planned outage of resources, the users would need to either be without access or be provided access to other services. In order to meet user SLA stateful services, the users would either need to moved to new servers or the servers would need to moved themselves. This use case will not required for stateless services.

Typical enterprise applications are not stateless and will need to be considered pets rather than cattle.

Data sources

- Nova: which VM is located on which server, move VM from one server to another
- RDBMS: which servers will be down

Policy

```
error(vm) :-
        nova:virtual_machine(vm),
        nova:server(vm, server),
        rdbms:server_to_go_down(server, time_to_go_down),
```

current_time(current),
// an error if it's an hour from when the server is supposed to go down
current_time + 1 >= time_to_go_down

Policy Actions

- Monitoring
- Reactive

HA of enterprise applications will make this use case viable.
Active passive approach
Active active approach

proposed features (pre-spec/blueprints details):

+3 Automatic Evacuation on Host Failure

votes: +1 sarob, kudva, banix

This use case describes the scenario when a host is down and the vms running inside have been tagged to have full availability.
The cloud operator will be able to define at project or vm level if this has to be evacuated in case of a host failure. Once the host fails, this event is notified and all the vms which are tagged or belong to a tenant which is tagged, will be automatically evacuated to a different host, if any.
The time that the machines were out of service until they were evacuated will be considered as part of the QoS metrics and SLA.

Data sources

- Nova: which VM is located on which server, move VM from one server to another
- RDBMS: which VM is high-availability
- Ceilometer?: which hosts are down

Policy

error(vm) :-
nova:virtual_machine(vm),
nova:server(vm, server),
rdbms:high_available(vm),
ceilometer:server_down(server)

Policy Actions

- Monitoring
- Reactive

proposed features (pre-spec/blueprints details):
1) disaster recovery for stateful virtual machines policy

- problem description: host is down and the vms running inside have been tagged to have full availability
- proposed change/how to solve: By not using local disk to run the VMs, the VMs can be restarted on another compute node. Once heartbeat failure is detected, the dead hypervisor ndoe would be taken out of the cluster, and a monitor service would schedule the VMs to be restarted.
- alternatives: aggressively attempt to restart the dead hypervisor node through IPMI or PDU hard restart.
- data model impact: TBD
- API impact: TBD
- security impact: TBD
- implementation plan: TBD
- dependencies: TBD

+1 Noisy Neighbor V-V

votes: +1 sarob,
Similar to planned tenant evacuation, except one container or VM would be moved and/or isolated. In this case, the VM would get moved to another hypervisor.
proposed features (pre-spec/blueprints details):
1) noisy neighbor policy

- problem description
- proposed change/how to solve: when using oversubscription, if memory, io, or cpu gets over X during Y period, move the VM.
- alternatives: disable oversubscription
- data model impact: TBD
- API impact: TBD
- security impact: TBD
- implementation plan: TBD
- dependencies: TBD

+0 Noisy Neighbor V-P

votes:
Similar to planned tenant evacuation, except one container or VM would be moved and/or isolated. In this case, the VM would get “transformed” into a baremetal instance.
proposed features (pre-spec/blueprints details):
1) noisy neighbor policy

- problem description
- proposed change/how to solve: when using oversubscription, if memory, io, or cpu gets over X during Y period, move the VM.
- alternatives: disable oversubscription
- data model impact: TBD
- API impact: TBD
- security impact: TBD
- implementation plan: TBD
- dependencies: TBD

Security use cases

+6 Compromised VM

votes: +1 banix, kudva, skn_, pballand, thinrichs, ramki

IDS service notices malicious traffic originating from an insider VM trying to send packets to hosts inside and outside of the tenant perimeter. As this is detected, some reactive response would need to be taken, such as isolating the offending VM from the rest of the network. This policy would facilitate one of the reactive responses to be invoked when a compromise is reported by an IDS service.

Data Sources

- IDS (intrusion detection service VM): IP address of the offending VM
- RepDBMS (reputation DBMS): database of known bad external IP addresses (may be replaced by API-based real time checks from many public blacklists)

Policy

```
error(vm) :-
        nova:virtual)_machine(vm),
        ids:ip_packet(src_ip, dst_ip),
        neutron:port(vm, src_ip),       //finds out the port that has the VM's IP
        rep_dbms:ip_blacklist(dst_ip).
```

Policy Actions

- Monitoring: report/log the incident including the VM's IP address, external IP, etc.
- Reactive: Invoke the neutron API to add a new rule to the port's security group that blocks all traffic to/from the VM's IP address (this is one of multiple ways of isolating the compromised VM)

Proposed features (pre-spec/blueprints details)

TBD

+3 Vulnerable software

votes: +1 banix, kudva, skn_

Vulnerability scanner reports a VM running an unpatched OS or application software, for instance, when a new vulnerability is reported and a patch is made available by the software vendor. As this is reported, some reactive measures would need to be taken, such as redirecting the traffic for this VM to another. This policy would facilitate a reactive response to be taken when such an incident is reported.

Data Sources

- VulScanner: <IP, port> that runs a vulnerable service

Policy

```
error(vm) :-
        nova:virtual_machine(vm),
        vulscanner:vulnerable(vul_ip, vul_port),
        neutron:port(vm, vul_ip).
```

Policy Actions

- Monitoring: log/report when a new vulnerable service is reported
- Reactive: take one of many corrective actions based on configuration parameters.
    - Option 1: deny all incoming traffic to this service ip/port
    - Option 2: install a new patched service in a new VM and redirect the traffic

Proposed features (pre-spec/blueprints details)

TBD

+2 Honeypot redirection for inbound traffic

votes: +1 banix , skn_

IDS service sees some suspicious inbound traffic and unsure of its legitimacy but needs a reactive measure such as redirecting the traffic to a honeypot created and deployed on-the-fly to understand it better. This policy needs to facilitate such mechanisms.

Data Sources

- IDS: <src_ip:src_port, dst_ip:dst_port> for the suspicious traffic

Policy

Policy Actions

- Monitoring
- Reactive: Use sub-policy implemented by Neutron and HoneyPot service

+1 Virtual Machines Placement

votes: +1 pettori

There are many examples where one would want Prod VMs to be on only Prod Hypervisor and no dev/test VMs (PCI for example), or a Virtual Machine that needs lot of throughput should go on a server that has 10G NIC and is not too utilized

Data Sources

- Nova
- Neutron

Policy

```
error(vm) :-
        nova:instance(vm),
        nova:stage(vm,stagevm),
        nova:compute(server),
        nova:stage(server,stageserver),
        not same_stage_group(stagevm, stageserver)
same_stage_group(x,y) :-
        stage_group(x, g),
        stage_group(y, g)
stage_group(“dev”, “devtest”)
stage_group(“test”, “devtest”)
stage_group(“prod”, “prod”)
```

Policy Actions Needed

What type of action does Congress need to take for this use case? Examples:

- Proactive: Responding to queries from other components about whether a given API call will cause new violations.
- Reactive: Computing/executing actions that bring the cloud back into compliance

+0 Trusted Compute Pool

votes:

Hardware enforced trust as an attribute for policy declaration. Only deploy VMs in a pool with hardware attributes. Geofencing as a direct scheduler based on attributes like VM and hosts. Intel was interested in this option.

Data Sources

-

Policy

Policy Actions

+ 4 Placement and Scheduling for Cloud & NFV Systems based on Policy Constraints (PSCN)

votes: +4 ramki, dilip, norival, madhu

The goal of PSCN is optimized placement and scheduling of Cloud/NFV systems based on resource constraints and service request requirements. The initial key goals are as follows

· IaaS scheduling across DCs of service provider #1 for supporting asynchronous replication model for state data as specified in the service request requirements by service provide #2 [NFV-USE-CASE]. The primary goal is to optimize resource utilization of network and energy in the infrastructure of service providers #1and #2. This applies to scheduling within a provider too.

· NFVIaaS placement across distributed NFV DCs of service provider #1 with various resource constraints such as compute, network and service request requirements as specified by service provider #2 [NFV-USE-CASE]. The primary goal is to optimize resource utilization of compute and network in the NFV infrastructure of service provider #1. This applies to placement within a provider too.

Data Sources

- Nova (Compute): VM – Tenant, Virtual Network Function (VNF), Server, NFV Data Center, Service Provider (needed for inter-provider use cases)
- Neutron (Network): Inter-DC Network - Tenant, Virtual Network Function (VNF), Service Provider (needed for inter-provider use cases), static network bandwidth, dynamic network bandwidth utilization
- Nova (Energy): Server power consumption in various idle states, Dynamic power consumption in servers (leverage IPMI or other standards)

Policy

- Hourly/daily/weekly time windows for scheduling certain category of application workloads (a good example would be asynchronous replication across DCs)
- Restrict VM placement only to certain DCs
- Maximum energy consumption per DC
- Dedicate capacity for example VMs in a DC, inter-DC network bandwidth etc. for handling certain types of workloads

Policy Actions

- Notify applications of precise time windows (order of minutes) for scheduling workloads

References:

[NFV-USE-CASE] “ETSI GS NFV 001Network Functions Virtualization (NFV); Use Cases,” http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[NFV-ARCH] “NFV Architectural Framework,” http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[NFV-REQ] “NFV Virtualization Requirements,” http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf