OSM Hackfest – Session 6
Performance & Fault Management
Gianpietro Lavado (Whitestack)
Performance and Fault Management capabilities have made important progress in Release FIVE. Metrics collection is now automatic, based on descriptor definitions, and supported from both infrastructure and VNFs (through VCA).
**MON Architecture**

**Brief review of MON’s architecture**

- **mon-central**
  - Creates alarms and handles VIM accounts

- **kafka**

- **alarm_requests (POL)**

- **vim_account (LCM)**

- **mysql**

- **vnfr iteration (mongodb)**

- **prometheus exporter**
  - VIM/N2VC metric plugins

- **mon-collector**
  - Iterates over vnfrs and collects metrics

- **mon-evaluator**
  - Iterates & evaluates alarms, triggering notifications to the bus

- **Prometheus TSDB**

© ETSI 2017
Performance Management
- OSM “MON” Component -
(1) MON collects VIM/VNF metrics defined at VNFD, from VNFs (through N2VC) and/or from NFVI (through VIMs)

(2) Prometheus TSDB stores metrics exposed by MON and exposes them at UI and its REST API via port 9091

(3) Analytics UI like Grafana can use existing plugins with well-known TSDB

OPTIONAL tools

Grafana
Main features

• Support for VIM metrics (related to VDUs)
  • OpenStack support ready since 5.0.0
  • vROps support ready in master (next 5.0.x point release)
  • AWS support pending
  • Supported metrics are cpu_utilization, average_memory_utilization, among others.

• Support for VNF-specific metrics.
  • Collection via proxy charms ‘juju metrics’ layer
  • Commands or API calls are executed from VCA to collect metrics every 5 minutes (fixed period)

• Monitoring happens on a per-VDU basis.
• VDU Metric Collection from VIM

vdu:
  id: apache_vdu
  ...
  monitoring-param:
    - id: "apache_cpu_util"
      nfvi-metric: "cpu_utilization"
  ...
  monitoring-param:
    - id: "apache_vnf_cpu_util"
      name: "apache_vnf_cpu_util"
      aggregation-type: AVERAGE
  vdu-monitoring-param:
    vdu-ref: "apache_vdu"
    vdu-monitoring-param-ref: "apache_cpu_util"

nfvi-metric corresponds to a established metric name at MON
VDU Metric Collection through VCA

```
vdu:
  - id: haproxy_vdu
    ...
    interface:
      - external-connection-point-ref: haproxy_mgmt
        mgmt-interface: true
    ...
  vdu-configuration:
    initial-config-primitive:
      ...
    juju:
      charm: testmetrics
      metrics:
        - name: load
      ...
    monitoring-param:
      - id: "haproxy_load"
        name: "haproxy_load"
        aggregation-type: AVERAGE
        vdu-metric:
          vdu-ref: "haproxy_vdu"
          vdu-metric-name-ref: "load"
```

**metrics “name”** corresponds to a predefined metric name at the proxy charm
• VNF Metric Collection through VCA

```
vnfd:
...
mgmt-interface:
  cp: haproxy_mgmt
vnf-configuration:
  initial-config-primitive:
    ...
  juju:
    charm: testmetrics
  metrics:
    - name: users
  ...
  monitoring-param:
    - id: "haproxy_users"
      name: "haproxy_users"
      aggregation-type: AVERAGE
      vnf-metric:
        vnf-metric-name-ref: "users"
```

metrics “name” corresponds to a predefined metric name at the proxy charm
Proxy Charm metrics layer

• Sample of ‘metrics.yaml’ file (root of charm folder)

```yaml
metrics:
  users:
    type: gauge
    description: "# of users"
    command: who|wc -l
  load:
    type: gauge
    description: "5 minute load average"
    command: cat /proc/loadavg |awk '{print $1}'
```
Metrics collection in action

Walkthrough Example (VIM Metrics)

1. Download and review descriptors from here:
   
   ```
   hackfest_autoscale_vimmetric_nsd
   hackfest_autoscale_vimmetric_vnfd
   ```

2. Onboard them!

3. Make sure the ‘public’ network maps to a network your browser can reach, and ‘mgmt’ network is not mapped to a VIM network. Your VIM should have Ceilometer/Gnocchi installed.

4. Make sure you MON container matches the metrics granularity of the underlying VIM
   
   ```
   docker service update --env-add OS_DEFAULT_GRANULARITY=60 osm_mon
   ```

4. Launch the NS, you will have a LB (HA Proxy) and a Web server (Apache).

5. Visit the load balancer IP Address with your browser
Walkthrough Example (VIM Metrics)

6. After a couple of minutes, visit the Prometheus TSDB GUI at OSM’s IP address, port 9091.

7. Validate that MON exporter “target” is properly connected at Status/Targets

8. Back in Graph, type ‘osm_cpu_utilization’ or ‘osm_average_memory_utilization’ and see if metrics are already there.
Metrics collection in action

Walkthrough Example (VIM Metrics)

9. Metrics should appear like this:
Walkthrough Example (VIM Metrics)

10. Now let’s add the optional Grafana component to see metrics in a friendlier way

Installing Grafana

./install_osm.sh -o pm_stack
Metrics collection in action

Walkthrough Example (VIM Metrics)

11. You should be able to visit Grafana at the OSM IP address, port 3000 (admin/admin)

12. There’s a default sample dashboard at ‘Manage → Dashboards’ (to the left), that will show some predefined graphs connected to Prometheus TSDB
Walkthrough Example (VDU Metrics from VCA)

1. Download and review descriptors from here:
   
   hackfest_autoscale_vnfmet_nsd
   
   hackfest_autoscale_vnfmet_vnfd

2. Onboard them!

3. Make sure the ‘vim-network-name’ of the management network points to a “PUBLIC” network that your OSM instance can reach.
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

6. You can visit the ‘juju status’ to see if the ‘metrics proxy charm’ is being built:

```
ubuntu@osm:~$ juju status
Model     Controller  Cloud/Region    Version  SLA      Timestamp
default   osm         localhost/localhost 2.5.0      unsupported 09:55:28Z

App        Version  Status      Scale  Charm      Store  Rev  OS      Notes
ub-b-ubuntuvvdub-aa  2.5.0    maintenance   1      testmetrics local  0   ubuntu

Unit       Workload  Agent     Machine  Public address  Ports  Message
ub-b-ubuntuvvdub-aa/0* maintenance executing 0  10.37.214.142    (install) installing charm software

Machine  State  DNS        Inst id  Series  AZ  Message
0        started 10.37.214.142 juju-d98962-0 xenial  Running
```
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

7. After around five minutes, you will see metrics at `juju metrics <name-of-the-application>`

```
ubuntu@osm:~$ juju metrics ub-b-ubuntuvdub-aa
UNIT                TIMESTAMP       METRIC      VALUE  LABELS
ub-b-ubuntuvdub-aa/0 2019-02-07T09:56:26Z load        0.15
ub-b-ubuntuvdub-aa/0 2019-02-07T09:56:26Z load_pct  15
ub-b-ubuntuvdub-aa/0 2019-02-07T09:56:26Z users      1
```
Walkthrough Example (VDU Metrics from VCA)

8. Finally, visit the Prometheus TSDB GUI at OSM’s IP address, port 9091. In ‘Graph’, type 'osm_load' or 'osm_users' and see if metrics are already there.

You can also see the metrics at Grafana.
Walkthrough Example (VDU Metrics from VCA)

9. Access with SSH to the VNF (ubuntu/osm2018) and execute `yes > /dev/null &`. You should see users and load metrics changing in the next collection interval (5mins).
Fault Management
- Docker logging & ‘POL’ Component -
FM – What’s available in Release FIVE?

(1) client includes thresholds (and actions) at descriptor

(2) POL creates alarms through MON

(3) MON configures the alarm locally and starts its evaluation process (by default every 30 seconds)

(4) When a metric threshold is crossed, MON puts a notification in the bus

OPTIONAL tools

beats

elasticsearch

kibana

KAFKA BUS

OSM N2VC

openstack

vmware

amazon web services

© ETSI 2017
Main Features

• Logging
  • docker containers send their logs to stdout.
  • They can be checked on the fly using:
    • docker logs osm_mon.1…
    • docker logs osm_lcm.1…
  • They can also be found at: /var/lib/containers/[container-id]/[container-id].json.log
• VCA logs
  • Run ‘juju debug-log’ from the host
Main Features

• Alarming

  • As of Release FIVE, MON includes a new module called 'mon-evaluator'. The only use case supported today by this module is the configuration of local alarms and evaluation of thresholds related to metrics, for the Policy Manager module (POL) to take actions such as auto-scaling (next chapter)

  • Whenever a threshold is crossed and an alarm is triggered, the notification is generated by MON and put in the Kafka bus so other components can consume them. This event is today logged by both MON (generates notification) and POL (consumes notification, for its auto-scaling action)
We can enable a “EBK” stack to visualize logs and metrics (Elasticsearch, Beats, Kibana)

- **Filebeats** collects logs from all docker containers
- **Metricbeats** collects metrics from the host, containers and applications, through modules.
- **Elasticsearch** organizes information and provides a way to filter and further process it.
- **Kibana** provides a way for visualizing information and building dashboards.
FM Experimental Features

• You can enable the EBK stack by using:

  ./install_osm.sh -o elk_stack

• After it’s up, visit it with your browser with the OSM IP, port 5601

• Import sample dashboards using this file:
  https://osm-download.etsi.org/ftp/osm-4.0-four/4th-hackfest/other/osm_elastic_dashboards.json
  (Management → Saved objects → Import)

• Go to ‘Discover’ and you will be asked to define one of the ‘beats’ as default ‘index pattern’, do so by selecting ‘filebeat-*’ and clicking
• All metrics and logging activity will appear at Kibana.

• Navigate the sample OSM dashboards and provide feedback!
Policy Management
- ‘POL’ Component -
(1) client includes thresholds and SCALING actions at VNF descriptor

(2) POL creates alarms through MON

(3) MON configures the alarm locally and starts its evaluation process (by default every 30 seconds)

(4) When a metric threshold is crossed, MON puts a notification in the bus

(5) SCALING actions are triggered based on the received notification

(6) LCM receives the scaling request and proceeds with instantiation

PM – What’s available in Release FIVE?
Main Features

• Autoscaling

  • Scaling descriptors can be included and be tied to automatic reaction to VIM/VNF metric thresholds.

  • An internal alarm manager is supported, so that both VIM and VNF metrics can trigger threshold-violation alarms and scaling actions.
VNF Scaling Descriptor (automatic, based on metrics)

```
scaling-group-descriptor:
  - name: "apache_vdu_autoscale"
    min-instance-count: 0
    max-instance-count: 10
    scaling-policy:
      - name: "apache_cpu_util_above_threshold"
        scaling-type: "automatic"
        threshold-time: 10
        cooldown-time: 120
        scaling-criteria:
          - name: "apache_cpu_util_above_threshold"
            scale-in-threshold: 20
            scale-in-relational-operation: "LT"
            scale-out-threshold: 80
            scale-out-relational-operation: "GT"
            vnf-monitoring-param-ref: "apache_vnf_cpu_util"
```

`vnf-monitoring-param-ref` corresponds to a predefined ‘monitoring param’
Model review - Sample VNFD

• Please note that scaling actions can also be triggered manually as long as there is a scaling descriptor of type ‘manual’

• The VNFD would look like this:

```json
scaling-group-descriptor:
  - name: "apache_vdu_manualscale"
    min-instance-count: 0
    max-instance-count: 10
    scaling-policy:
      - name: "apache_cpu_util_manual"
        scaling-type: "manual"
        threshold-time: 10
        cooldown-time: 120
```
The API call for that is:

- URL: POST to ns lcm/v1/ns_instances/{{nsInstanceId}}/scale
- Body

```json
{
  "scaleType": "SCALE_VNF",
  "scaleVnfData":
   {
     "scaleVnfType": "SCALE_OUT",
     "scaleByStepData": {
       "scaling-group descriptor": "apache_vdu_manualscale",
       "member-vnf-index": "1"
     }
   }
}
```
Walkthrough Example

1. Launch a ubuntu machine with a m1-small flavor to use it as a client for stressing our HAProxy+Apache VNF locally. Instantiate it at the PUBLIC network.

Make sure you will be able to access it, either by using your ssh-key or the following configuration script:

```
#cloud-config
hostname: ubuntu_client
password: osm2018
chpasswd: { expire: False }
ssh_pwauth: True
```

2. Install Apache-Bench: `sudo apt-install apache2-utils`
Walkthrough Example

2. From this client, run a stress test towards your load balancer’s IP address:

   ab -n 5000000 -c 2 http://[HA-Proxy-IP]/test.php

3. Watch the policy manager logs to detect for autoscaling instructions. CPU should start going up in a minute, validate that at the Grafana Dashboard.
Walkthrough Example

4. Instances of Apache Web Server should start appearing (up to 2 or 3 before it starts load balancing traffic accordingly), validate this at the OpenStack Network Topology and visiting the HAProxy IP address.

5. Finally, test scale-in by stopping the traffic and waiting for a couple of minutes.