OSM Hackfest – Session 7
Performance & Fault Management
Benjamin Diaz (Whitestack)
MON Architecture

Performance Management
- OSM “MON” Component -
PM – What’s available at OSM?

(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
Main features

- Support for VDU VIM metrics
  - Openstack and VMWare (VIO/VCD) plugins available
  - Pending support for AWS
  - Supported metrics are cpu_utilization, average_memory_utilization, among others.

- Support for VNF-specific metrics
  - Collection via proxy charms ‘juju metrics’ layer
  - 5 minute interval

- 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 OSM metric name which maps to the corresponding metric in each supported VIM.
• VDU Metric Collection through VCA

vdud:
  - 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

```json
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}'
```
Walkthrough Example (VIM Metrics)

1. Download and review descriptors from here:
   a. webserver_autoscale_vimmetric_nsd
   b. webserver_autoscale_vimmetric_vnfd
2. Onboard them!
3. Make sure you have a ‘vnf-mgmt’ network created at your vim and that your OSM is connected to it (do not confuse it with the public mgmt network)
4. Launch the NS, you will have a LB (HA Proxy) and a Web server (Apache)
   a. --config '{vld: [ {name: public_vld, vim-network-name: osm-ext} ] }'
5. Visit the load balancer IP Address with your browser
6. 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)
Metrics collection in action

Installing Grafana

- Now let’s add the optional Grafana component to see metrics in a friendlier way
- ./install_osm.sh -o pm_stack
Metrics collection in action

Walkthrough Example (VIM Metrics)

- You should be able to visit Grafana at the OSM IP address, port 3000 (admin/admin)
- There’s a default sample dashboard at ‘Manage → Dashboards’ (to the left), that will show some predefined graphs connected to Prometheus TSDB
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

- Download and review descriptors from here:
  - `ubuntuvm_vnfmetric_autoscale_nsd`
  - `ubuntuvm_vnfmetric_autoscale_vnfd`
- Onboard them!
- Deploy the NS:
  - `--config '{vld: [ {name: mgmt, vim-network-name: osm-ext} ] }'`
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

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

```
$ juju status
```

<table>
<thead>
<tr>
<th>App</th>
<th>Version</th>
<th>Status</th>
<th>Scale</th>
<th>Charm</th>
<th>Store</th>
<th>Rev</th>
<th>OS</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ub-b-ubuntuvdb-a</td>
<td>2.5.0</td>
<td>maintenance</td>
<td>1</td>
<td>testmetrics</td>
<td>local</td>
<td></td>
<td>ubuntu</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit</th>
<th>Workload</th>
<th>Agent</th>
<th>Machine</th>
<th>Public address</th>
<th>Ports</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ub-b-ubuntuvdb-a/0*</td>
<td></td>
<td></td>
<td>maintenance</td>
<td>10.37.214.142</td>
<td></td>
<td>(install) installing charm software</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine</th>
<th>State</th>
<th>DNS</th>
<th>Inst id</th>
<th>Series</th>
<th>AZ</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>started</td>
<td>10.37.214.142</td>
<td>juju-d98962-0</td>
<td>xenial</td>
<td></td>
<td>Running</td>
</tr>
</tbody>
</table>
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

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

```
<table>
<thead>
<tr>
<th>UNIT</th>
<th>TIMESTAMP</th>
<th>METRIC</th>
<th>VALUE</th>
<th>LABELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ub-b-ubuntuv-db-aa/0</td>
<td>2019-02-07T09:56:26Z</td>
<td>load</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>ub-b-ubuntuv-db-aa/0</td>
<td>2019-02-07T09:56:26Z</td>
<td>load_pct</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>ub-b-ubuntuv-db-aa/0</td>
<td>2019-02-07T09:56:26Z</td>
<td>users</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

© ETSI 2019
Walkthrough Example (VDU Metrics from VCA)

- 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.
Metrics collection in action

Walkthrough Example (VDU Metrics from VCA)

• 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
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)
FM Experimental Features

• 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.
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 -
PM – What’s available in Release FIVE?

1. **NBI**
   - Client includes thresholds and SCALING actions at VNF descriptor

2. **POL**
   - POL creates alarms through MON

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

4. **KAFKA BUS**
   - When a metric threshold is crossed, MON puts a notification in the bus

5. **KAFKA BUS**
   - SCALING actions are triggered based on the received notification

6. **LCM**
   - LCM receives the scaling request and proceeds with instantiation

---

(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
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)

```json
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’
• 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:

```yaml
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
```
Model review - Sample VNFD

• The API call for that is:
  • URL: POST to ns lcm/v1/ns_instances/{{nsInstanceId}}/scale
  • Body

  {"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:

```bash
#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:
   \[\text{ab -n 5000000 -c 2 http://[HA-Proxy-IP]/test.php}\]

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.
Autoscaling in action

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.