Current Architecture & Features

OSM Service Assurance
Service Assurance MDG

Main components

MON
- Covers the basic use cases, with a solid architecture to expand them easily.
- Opportunities to enhance usability.

POL
- Designed around the autoscaling use case.
- Starting to cover VNF alarms.
- Architecture needs a revisit based on expected use cases.

Prometheus
- OSM’s TSDB for metrics since REL5
- Opportunities to enhance multi-tenancy to match new RBAC capabilities.

Auxiliary/Optional Tools

Grafana
- Integrates seamlessly with Prometheus.
- Great tool for enhancing usability of the system’s Service Assurance → should be included by default?

ELK
- Proved seamless integration with OSM.
- Main use case remains at log processing where stack is used.
MON Architecture

Formal documentation: https://osm.etsi.org/gitlab/osm-architecture/osm-arch-doc/blob/master/04-mon.md
POL Architecture

Formal documentation: https://osm.etsi.org/gitlab/osm-architecture/osm-arch-doc/blob/master/05-pol.md
### Metrics Collection @ OSM

<table>
<thead>
<tr>
<th>Metric</th>
<th>Collection type</th>
<th>Behavior</th>
<th>KPI</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIM Status</td>
<td>Infrastructure</td>
<td>By default</td>
<td>status (up/down)</td>
<td>vim_id</td>
</tr>
<tr>
<td>SDNC Status</td>
<td></td>
<td>By default</td>
<td>status (up/down)</td>
<td>sdnc_id</td>
</tr>
<tr>
<td>VM Status</td>
<td></td>
<td>By default</td>
<td>status (up/down)</td>
<td></td>
</tr>
<tr>
<td>VDU CPU Utilization</td>
<td></td>
<td>Enabled by descriptor</td>
<td>utilization, rate, etc.</td>
<td>nsr_id, vnf_member_index, vdu_name</td>
</tr>
<tr>
<td>VDU Memory Utilization</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDU Packet forwarding</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VNF Metrics through Juju</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(to be deprecated)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• VDU Metric Collection from VIM

```json
vdun
  id: hackfest_basic-VM
  ...
  monitoring-param:
    - id: "cpu_util"
      nfvi-metric: "cpu_utilization"
    ...
  monitoring-param:
    - id: "vnf_cpu_util"
      name: "vnf_cpu_util"
      aggregation-type: AVERAGE
      vdu-monitoring-param:
        vdu-ref: "hackfest_basic-VM"
        vdu-monitoring-param-ref: "cpu_util"
```

`nfvi-metric` corresponds to a OSM metric name which maps to the corresponding metric in each supported VIM.
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.

```yaml
scaling-group-descriptor:
  - name: "vdu_autoscale"
    min-instance-count: 0
    max-instance-count: 10
    scaling-policy:
      - name: "cpu_util_above_threshold"
        scaling-type: "automatic"
        threshold-time: 10
        cooldown-time: 120
        scaling-criteria:
          - name: "cpu_util_above_threshold"
            scale-in-threshold: 10
            scale-in-relational-operation: "LT"
            scale-out-threshold: 60
            scale-out-relational-operation: "GT"
            vnf-monitoring-param-ref: "vnf_cpu_util"

  vdu:
    - vdu-id-ref: hackfest_basic_metrics-VM
      count: 1
```
• Alarms based on metric thresholds can be sent to webhooks

```json
  vdu:
  -  alarm:
      -  alarm-id: alarm-1
      operation: LT
      value: 20
      actions:
        alarm:
          -  url: https://webhook.site/1111
        ok:
          -  url: https://webhook.site/2222
        insufficient-data:
          -  url: https://webhook.site/3333
        vnf-monitoring-param-ref: vnf_cpu_util
```
New Proposals
OSM Service Assurance
New methods for VNF Indicator Collection

Objective: Evolve the way OSM collects VNF indicators to allow for more compatibility with VNFs, real-time collection and standards alignment.

A first approach is using additional “Prometheus exporters”
### Objective: OSM Operators can install OSM and immediately and permanently know the health of the system.

<table>
<thead>
<tr>
<th>Feature 7898</th>
<th>Feature 8132</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coverage</strong></td>
<td>OSM on Kubernetes</td>
</tr>
<tr>
<td>● Prometheus Operator Chart (New prometheus instance, Grafana and different exporters: node, cadvisor, etc.)</td>
<td>● Grafana promoted to OSM stack.</td>
</tr>
<tr>
<td>● Other charts: MongoDB, MySQL and Kafka exporters</td>
<td>● Node exporter</td>
</tr>
<tr>
<td><strong>Additional components</strong></td>
<td>● CAdvisor exporter</td>
</tr>
<tr>
<td><strong>Implements</strong></td>
<td>Multiple Grafana dashboards for a comprehensive health check of the system.</td>
</tr>
</tbody>
</table>

Check it out! (beta) → [http://172.21.248.14:3000/d/gDHmpHbWk/osm-system-metrics](http://172.21.248.14:3000/d/gDHmpHbWk/osm-system-metrics)
Project-scoped VIM/VNF Metrics

Objective: Follow RBAC structure for metric consumption.

- Prometheus does not support multi-tenancy, other projects need to be explored (e.g. Cortex)
- Short-term proposal is to add a label for `project_id` in all Prometheus metrics

Objective: adding to the previous feature, a new “MON Dashboarder” component will take care of dashboard “lifecycle”.

<table>
<thead>
<tr>
<th>Updates in...</th>
<th>...automates these dashboards...</th>
<th>...and these Grafana resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSM installation</td>
<td>System Metrics, Admin Project-scoped</td>
<td>Admin-privileges</td>
</tr>
<tr>
<td>OSM Projects</td>
<td>Project-scoped (Grafana “team” privileges)</td>
<td>Grafana “team” privileges</td>
</tr>
<tr>
<td>OSM Users</td>
<td>-</td>
<td>Grafana users to teams</td>
</tr>
<tr>
<td>OSM Network Services</td>
<td>NS-scoped</td>
<td>-</td>
</tr>
</tbody>
</table>

...Let’s play with the prototype!
Hands-on!

OSM Service Assurance
Let’s play with metrics and (auto)dashboards!

1. From your SSH console, download new descriptors and upload them to OSM

```bash
wget http://osm-download.etsi.org/ftp/osm-6.0-six/8th-hackfest/packages/hackfest_basic_metrics_vnfd.tar.gz
wget http://osm-download.etsi.org/ftp/osm-6.0-six/8th-hackfest/packages/hackfest_basic_metrics_nsd.tar.gz
osm vnfd-create hackfest_basic_metrics_vnfd.tar.gz
osm nsd-create hackfest_basic_metrics_nsd.tar.gz
```

2. Create your VIM & instantiate your NS

```bash
osm vim-create --name whitecloud_XX --user osm_hackfest_XX --password osm_hackfest_XX --auth_url http://172.21.247.1:5000/v3 --tenant osm_hackfest_XX --account_type openstack

osm ns-create --ns_name hfmetrics_XX --nsd_name hackfest_basic-ns-metrics --vim_account whitecloud_XX --config '{vld: [ {name: mgmtnet, vim-network-name: osm-ext} ] }'
```

3. Go and check how your own “project dashboard” starts to be populated.

Then, look for a new dashboard dedicated to your NS! → http://172.21.248.14:3000/dashboards

© ETSI 2019
Let’s play with autoscaling!

1. Access your VM and stress it out!

   osm vnf-list  # to find the IP address
   ssh ubuntu@172.21.248.93  # password: osm4u
   yes > /dev/null &  # 4 or 5 times!

2. Wait for a bit (5 to 10 minutes due to current collection period), and watch it scale!
Open Source MANO

Find us at:

osm.etsi.org
osm.etsi.org/wikipub