Using Microservices Architecture and Patterns to address application requirements on MEC

Presented by Cloud Architects: Prem Sankar Gopannan (E///) and Prakash Ramchandran (DellEMC)
Agenda – Part 1 – Prem Sankar G, Ericsson

• MEC Application requirements
• Microservices architecture and patterns
• Walkthrough – Sample app
Application Types

- Offline applications
  - Fat Client
  - Processing done locally at UE
  - Data feed from cloud

- Online Applications
  - Presentation layer
  - All processing done online
  - Requires connectivity
Key requirements for Applications

• Bandwidth
• Latency, Jitter and Delay
• Location and Presence
MEC Architecture
1. UE APP requests AF to create compute for it (Eg. AR/VR, Gaming offload). UE provides information such as ‘coordinates’, ‘max cost’, ‘max latency’ etc...

2. AF requests ONAP to create VNFs in edge that is closer to UE that satisfies UE request (ONAP figures out the best region out of thousands of clouds)

3. ONAP brings up VNFs on the Edge Cloud using VIM API

4. AF informs NEF/SMF to create traffic rule (to enable redirection of UE APP traffic to new VNFs created).

5. SMF informs UPF in the edge cloud (Programs UE classifier of UPF).

6. When the traffic comes from UE APP, right application in the edge cloud receives the traffic.

Reference – ONAP Edge working group – Srinivas Addepalli, Intel
Sample application – Pothole fixer

- **Dashboard app** – Control app for the equipment that connects with edge network/Dispatcher app
- **Dispatcher app** – Dispatches jobs to equipment
- **Potholefix app** – App that is in core cloud has info about pothole, location and other information
Sample application – Pothole fixer

• System functions
  1. AF Request
  2. Select VIM and triggers VNF request
  3. Bring up VNF on the selected edge and establish transport path
  4. Redirect job to payload

• Application functions
  1. Require Fix – Pothole Image, City Authority (for budget)
  2. Response to bobcat dashboard application
  3. Terminate application
Microservice Architecture – Quick Intro

Microservices Design Patterns

Reference – Chris Richardson, http://microservices.io
SAGA Patterns

- Database per Service
- Replaces 2 Phase Commit
- Uses Event streams to co-ordinate between the distributed database
- Co-ordination via two ways
  - Choreography - each local transaction publishes domain events that trigger local transactions in other services
  - Orchestration - an orchestrator (object) tells the participants what local transactions to execute
Agenda – Part 2 by R.Prakash (Openstack)

• MEC – Architecture
• Cloudlet and Statelet
• Statelet Design and Flow
• Statelet proposed APIs
  • Admin API for Persistent Volume (PV)
  • User API for Persistent Volume (PV)
• Deployment options – OpenStack Zun micro services for PV
MEC reference architecture in a NFV environment

- UE uses **Cloudlet(C)** API over Mx2: UE App to User/App Proxy (Ambassador Pattern)
- MEPM manages MEP over Mm5
- MEP manages **(A) App VNF** over Mp1
- **Statelet(S)** is a state management service to MEP & ME App VNFM LCM over Mm6/Vi-Vnfm
Mobile Edge Computing Standard V1 to V2

Enable MEC development in NFV environments.

Expedite the development of innovative applications; ensure a low entry barrier.

Strengthen collaboration with other organization.

Cloudlet

Application Statelet

Openstack Zun LFN (CNI)

Management of the ME platform as a VNF
State Management for Edge Cloud on Server Side

UE talks to Cloudlet VM or container through Proxy for running Application

MEP Manager manages MEP
MEP manages Application

Statelet Support Cloudlet and Application to manage Application State using Shared Persistent Volume pools with mount and unmount calls.
Persistent Volume for state management at Edge

1. Create PV
2. Request PV Claim
3. Grant PV Claim
4. Mount PVC to VM or Container
State Management using Persistent Volume for Edge Cloud

• A Persistent Volume (PV) is a network attached storage in the cluster, which is provisioned by the administrator and user claims them from the Pools to mount them in PODs or VMs and finally used by the applications contained in them.

• PVs based on the Storage Class resource. A StorageClass contains pre-defined provisioners and parameters to create a Persistent Volume.

• Volume Types that support managing storage using Persistent Volumes are several but we will limit it to hostPath, CephFS or Cinder as in Openstack for Cloudlet. For minikube demo we use hostPath. Note for VMs use Cinder/CephS.

• So the Object PV will have C,RC,GC,MT(attach), UMT(detach) REST Calls for Cloudlets to support State mangement for both Openstack VMs and Containers.
## Admin APIs

**PV Pool, PV/PVExtend**

<table>
<thead>
<tr>
<th>Object</th>
<th>Action</th>
<th>Type</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PV_Pool</td>
<td>Create</td>
<td>Default</td>
<td>PV of Type hostPath</td>
</tr>
<tr>
<td>PV_Pool</td>
<td>Delete</td>
<td>Default</td>
<td>PV of Type hostPath</td>
</tr>
<tr>
<td>PV</td>
<td>Create</td>
<td>hostPath/Default</td>
<td>Adapt from k8s</td>
</tr>
<tr>
<td>PV</td>
<td>Create</td>
<td>Cinder/BlockStore</td>
<td>Adapt from Openstack</td>
</tr>
<tr>
<td>PV</td>
<td>Create</td>
<td>CephFS/FileStore</td>
<td>Adapt from Ceph</td>
</tr>
<tr>
<td>PVExtend</td>
<td>Create</td>
<td>Host Volume</td>
<td>k8s Worker Node</td>
</tr>
<tr>
<td>PVExtend</td>
<td>Create</td>
<td>Host Volume</td>
<td>Openstack compute Node</td>
</tr>
<tr>
<td>PV</td>
<td>Delete</td>
<td>PV Name</td>
<td>PV_Pool</td>
</tr>
<tr>
<td>PV_Pool</td>
<td>Delete</td>
<td>PV_Pool Name</td>
<td>PV list</td>
</tr>
</tbody>
</table>
## User APIs

**PV Claim, Grant, Mount, UnMount**

<table>
<thead>
<tr>
<th>Object</th>
<th>Action</th>
<th>Type</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PV</td>
<td>Claim</td>
<td>PV_Pool</td>
<td>Capacity</td>
</tr>
<tr>
<td>PV</td>
<td>Grant</td>
<td>PV_Pool</td>
<td>Capacity</td>
</tr>
<tr>
<td>PV</td>
<td>Mount</td>
<td>PV Name</td>
<td>Mount Device</td>
</tr>
<tr>
<td>PV</td>
<td>UnMount</td>
<td>Mount Device</td>
<td>PV Name</td>
</tr>
<tr>
<td>PV</td>
<td>Assign</td>
<td>PV Name</td>
<td>PV_Pool</td>
</tr>
<tr>
<td>PV_Pool</td>
<td>Assign</td>
<td>PV_Pool Name</td>
<td>PV list</td>
</tr>
</tbody>
</table>
MEC: Cloudlet, Statelet, ME Applications

- http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing
- http://www.etsi.org/deliver/etsi_gr/MEC/001_099/017/01.01.01_60/gr_MEC017v010101p.pdf
- https://www.cncf.io/blog/2017/05/23/cncf-hosts-container-networking-interface-cni/
- https://docs.openstack.org/queens/api/
- Refer APIs for Zun, Senlin, manila to define and design Statelet
- First add Statelet API for PV (Admin/User) to Zun for Containers
- Use Senlin to form container clusters using Zun and use Senlin Profile and Policy for placement
- Use Shared File Systems service (manila) if you use CephFS
- https://www.openstack.org/edge-computing/
- https://www.openstack.org/assets/edge/slides/2017-09-07-a-satya-opendev-keynote.pdf
- https://docs.openstack.org/kolla/latest/
- https://wiki.opnfv.org
- https://wiki.onap.org
- http://microservices.io/
- Contacts: Prem Sanka Gopannan (premsankar@gmail.com)/ Prakash Ramchandran (cloud24x7@ieee.org)