OSM Release SEVEN Webinar
Part 2: Running cloud-native Network Services with OSM
16 January 2020
### Overview of the webinar

#### DAY #1:
Overview of OSM Release SEVEN

1. **OSM introduction and Release SEVEN highlights**
   (Francisco-Javier Ramón, Telefónica) [10 minutes]

2. **Running real Network Services with OSM**
   (Gianpietro Lavado, WhiteStack) [40 minutes]

3. **Monitoring the Open Source MANO components**
   (Francisco Rodríguez, Indra) [15 minutes]

#### DAY #2:
Running cloud-native Network Services with OSM

1. **Deploying on Kubernetes with OSM**
   (Gerardo García, Telefónica) [20 minutes]

2. **Deploying on Kubernetes with OSM and Juju bundles**
   (Tytus Kurek, Canonical) [20 minutes]

3. **Automatic placement of Network Services**
   (Mats Eriksson, Arctos Labs) [20 minutes]
REMEMBER:
If you want to learn more...

... you can join us to our upcoming OSM Hackfest!

Register now for the Open Source MANO Hackfest in Madrid, Spain from 9th to 13th March 2020.
Deploying on Kubernetes with OSM
Gerardo García (TSC member, Telefónica)
Why K8s in OSM?

• Applications based in micro-services
  • OSM is, in fact, already running in K8s, both distros and community installer

• Upcoming NFV use cases: 5G Core, uCPE/SD-WAN...

• K8s apps and clusters are essential ingredients for many Edge use cases
Requirements of K8s-based apps

1. A K8s cluster

• The K8s cluster:
  • Can be created in different ways:
    • Standalone: Openshift, Charmed K8s, Ericsson CCD, etc.
    • As part of a VIM: Vmware Cloud PKS, AWS, etc.
  • Can run on Bare Metal or on VMs running in a VIM
  • Once created, each cluster provides a K8s API, irrespective of the way it was created.

• Specific versions of K8s or CNI plugins might be required
Requirements of K8s-based apps

2. K8s service objects

- K8s provides a huge number of high-level service objects, which are the core of its functionality:
  - Pod sets*: deployments (+replicasets), statefulsets
  - Services: clusterIP, NodePort, LoadBalancer
  - Storage: persistent volumes, persistent volume claims
  - ...

- TWO Ways to deploy a K8s app:
  - **Helm charts**: packaged format + indirect call to the K8s API via helm
  - **Juju charms and bundles**: packaged format + indirect call to the K8s API via Juju

(*) The concept “pod set” is not part of K8s terminology, but has been used here for convenience
Requirements of K8s-based apps

2. K8s service objects

- K8s provides a huge number of high-level service objects, which are the core of its functionality:
  - Pod sets*: deployments (+replicasets), statefulsets
  - Services: clusterIP, NodePort, LoadBalancer
  - Storage: persistent volumes, persistent volume claims
  - ...

- TWO Ways to deploy a K8s app:
  - **Helm charts**: packaged format + indirect call to the K8s API via helm
  - **Juju charms and bundles**: packaged format + indirect call to the K8s API via Juju

(*) The concept “pod set” is not part of K8s terminology, but has been used here for convenience
K8s support in OSM
From K8s apps to xNF
Model-driven (like everything in OSM)

- NF composition specified in the VNF descriptor
  - Deployment Units:
    - Virtual (VDU) = VM
    - Physical (PDU) = Physical Node
    - Kubernetes (KDU) = K8s app
  - Modelling in the VNF descriptor:
    - KDU based on helm charts or juju bundles
      ```
      ├── root
      │   └── kdu*[
      │       └── name [string]
      │         └── description [string]
      │             └── (kdu-model)
      │                 └── (helm-chart)
      │                     └── (juju-bundle)
      │                         └── juju-bundle [string]
      ```

- K8s cluster requirements:
  ```
  ├── root
  │   └── k8s-cluster
  │       └── version [string]
  │             └── cni [string]
  │                 └── nets [id]
  │                     └── id [string]
  │                           └── external-connection-point-ref [string]
  ```

© ETSI 2020
Two steps are considered in OSM

<table>
<thead>
<tr>
<th>STEP #1. CREATION OF THE K8S CLUSTER</th>
<th>STEP #2. USE OF THE K8S CLUSTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS:</td>
<td></td>
</tr>
<tr>
<td>1. By an external platform, static</td>
<td>• The full catalog of K8s objects is entirely incorporated in a future-proof manner:</td>
</tr>
<tr>
<td>• Cluster is then registered into OSM administratively</td>
<td>• Helm charts: +20,000 stable applications are already available for production</td>
</tr>
<tr>
<td>2. By using external standalone platform API</td>
<td>• Juju bundles: fairly powerful for inter-object configurations</td>
</tr>
<tr>
<td>• Covered by plugin model (Rel EIGHT)</td>
<td>• OSM also supports <strong>hybrid cases</strong>, which are required for real VNFs (e.g. 5G Core)</td>
</tr>
<tr>
<td>3. By using “enriched” APIs in some VIMs</td>
<td></td>
</tr>
<tr>
<td>• Covered by plugin model (Rel EIGHT)</td>
<td></td>
</tr>
<tr>
<td>4. Created by OSM as a regular NS</td>
<td></td>
</tr>
</tbody>
</table>
Life cycle management of KDU is managed through OSM NBI

Full K8s app lifecycle operations:
- install
- upgrade
- rollback
- delete
Open Source MANO

DEMO
Demo

• Demo 1. Cluster creation using OSM NS
• Demo 2. Deployment of simple K8s-based VNF
  • Status
  • Implicit day-2 operations: upgrade, rollback
• Demo 3. Deployment of a multi-KDU VNF
THANK YOU
Deploying container network functions (CNFs) on Kubernetes with OSM and K8s charms

Tytus Kurek (Product Manager, Canonical)
Types of network functions
Types of network functions

Virtual Machine  Virtual Machine  Virtual Machine

Virtual Network Functions
Types of network functions

Virtual Network Functions
- Virtual Machine
- Virtual Machine
- Virtual Machine

Physical Network Functions
- Bare Metal Machine
- Bare Metal Machine
- Bare Metal Machine
Types of network functions

Virtual Machine

Virtual Network Functions

Bare Metal Machine

Physical Network Functions

Container

Container Network Functions

© ETSI 2020
Types of network functions

- **Physical Network Functions**
  - Bare Metal Machine
  - Bare Metal Machine
  - Bare Metal Machine

- **Virtual Network Functions**
  - Virtual Machine
  - Virtual Machine
  - Virtual Machine

- **Hybrid Network Functions**
  - Bare Metal Machine
  - Virtual Machine
  - Container

- **Container Network Functions**
  - Container
  - Container
  - Container

© ETSI 2020
Why CNFs?

- Microservices architecture
- Better orchestration capabilities
- More complex network services
- Performance and density considerations
- Edge deployments
Kubernetes operators

**Operators** - software extensions to Kubernetes that make use of custom resources to manage applications and their components.

Operators abstract operational tasks in a form of primitives.
K8s charms

**Charms** - provide framework for implementing *operators*.

The entire logic required to perform day-1 and day-2 operations:

- connecting network functions inside of network services
- network functions primitives (e.g. install, scale-up, upgrade)
- configuration management capabilities

By using the concept of relations, hooks, actions and templates, charms provide a native framework for implementing operators.

**K8s charms allow deploying CNF workloads on Kubernetes!**
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms
Deploying CNFs on K8s with charms #2

vnfd-catalog:
  vnfd:
    - id: myvnf
      name: myvnf
      connection-point:
        - name: mgmtnet
      mgmt-interface:
        cp: mgmt
      kdu:
        - name: mykdu
          juju-bundle: cs:~aisrael/bundle/mediawiki-k8s-0
  k8s-cluster:
    nets:
      - id: mgmtnet
        external-connection-point-ref: mgmt
THANK YOU
Placement Optimization in OSM

Release Seven Webinar Jan 16th

Mats Eriksson, Lars-Göran Magnusson
Arctos Labs
Placement Optimization - Outline

- What do we mean by Placement?
- Introduction to Placement Optimization in OSM Release Seven
- Achieving different types of Optimization
- Use case examples for Placement Optimization
- Invoking Placement from NBI
- Demonstration
What do we mean by Placement Optimization?

- Placement in context of OSM is the process of deciding which VNF that goes into which VIM (PoP)

- Optimal is subject to:
  - Cost of compute in PoP’s
  - Cost of links for NS interworking
  - Constraints in NS interworking (Latency, Jitter) – if there are any

- Placement feature makes this process **Automatic & Optimal**

Example:
1 Central PoP
1 Regional PoP
2 Local PoP’s
The Optimization Process

- **Placement function**
  - Will consider all VIM’s available to the user
  - Will make sure constraints are met – if there are any
  - Will optimize Cost (the Criteria)

- I.e. select the option that fulfills constraints at the lowest possible cost
Introduction to Placement in OSM Release Seven

• Automatic placement is optional – i.e. has to be specifically requested
• Placement is in-flow when requested – i.e. is native capability. New OSM module introduced
• Uses specified cost, latency & jitter
• Will consider placement over the VIMs available to the user
• Constraints are given in the instantiation request (more about that later)
Placement optimization examples

1. **Cost optimization only**
   - Constraint: **Nothing**
   - Criteria: **Placement**, **Cost**

2. **Cost optimization with Latency constraint**
   - Constraint: **Latency / Jitter**
   - Criteria: **Placement**, **Cost**

3. **Utilization optimization with Latency constraint**
   - Constraint: **Latency**
   - Criteria: **Placement**, **Utilization**

4. **Cost optimization with Capability constraint**
   - Constraint: **Capability** (e.g., CPU / GPU etc.)
   - Criteria: **Placement**, **Cost**

© ETSI 2020
Examples of Use-cases

**UPF supporting Low-latency**
Placement of UPF close to customer to achieve latency constraint

**Transport optimization (cost) for Application components**
Placement of Application components close to the source of data to reduce transport cost / load

**Compute cost optimization for slicing**
Placement of CN VNF’s at most cost effective compute

- **UPF**
- **IoT data proc**
- **VIM**
- **VIM**
- **VIM**
- **MME**
- **RAN**
- **S-GW**
- **P-GW**
- **APN**

Deploy as close as it has to be
Deploy as far away as it can be
Invoking Placement through the NBI

1. Request Placement Cost Optimization
   --config '{ additionalParamsForNs: {placementEngine: PLA} }'

2. Request Placement Cost Optimization with pinning of specified VNF
   --config '{additionalParamsForNs: {placementEngine: PLA, vnf: [{member-vnf-index: "3", vim_account: OpenStack_3}]} }'

3. Request Placement Cost Optimization with VLD Constraints
   --config '{additionalParamsForNs: {placementEngine: PLA, constraints: {vld-constraints: [{id:vld_1, link-constraints: {latency: 120, jitter: 20}}, {id:vld_2, link-constraints: {jitter: 20}}]}}}'

4. Combo of 2 and 3

Note: GUI is also supported, with or without YAML file
Use of VNF Pinning

- Ability to “pin” a VNF to e.g.:
  - the VIM with a specific VNF (e.g. P-GW)
  - the VIM with connectivity to a PNF
  - a CPE (customer location)

Auto implies there is no VIM specified, thus placement is subject to placement optimization
We want to assist...

• We would love to assist You in applying placement in your next project
• We would like to get feedback to evolve the feature

mats.eriksson@arctoslabs.com
lars-goran.magnusson@arctoslabs.com
Live Demo

Thank you!
REMEMBER:
If you want to learn even more...

... you can join us to our
upcoming OSM Hackfest!

© ETSI 2020

Registrator
OSM Hackfest
Madrid, Spain 9 - 13 March 2020