Open Source MANO

by ETSI

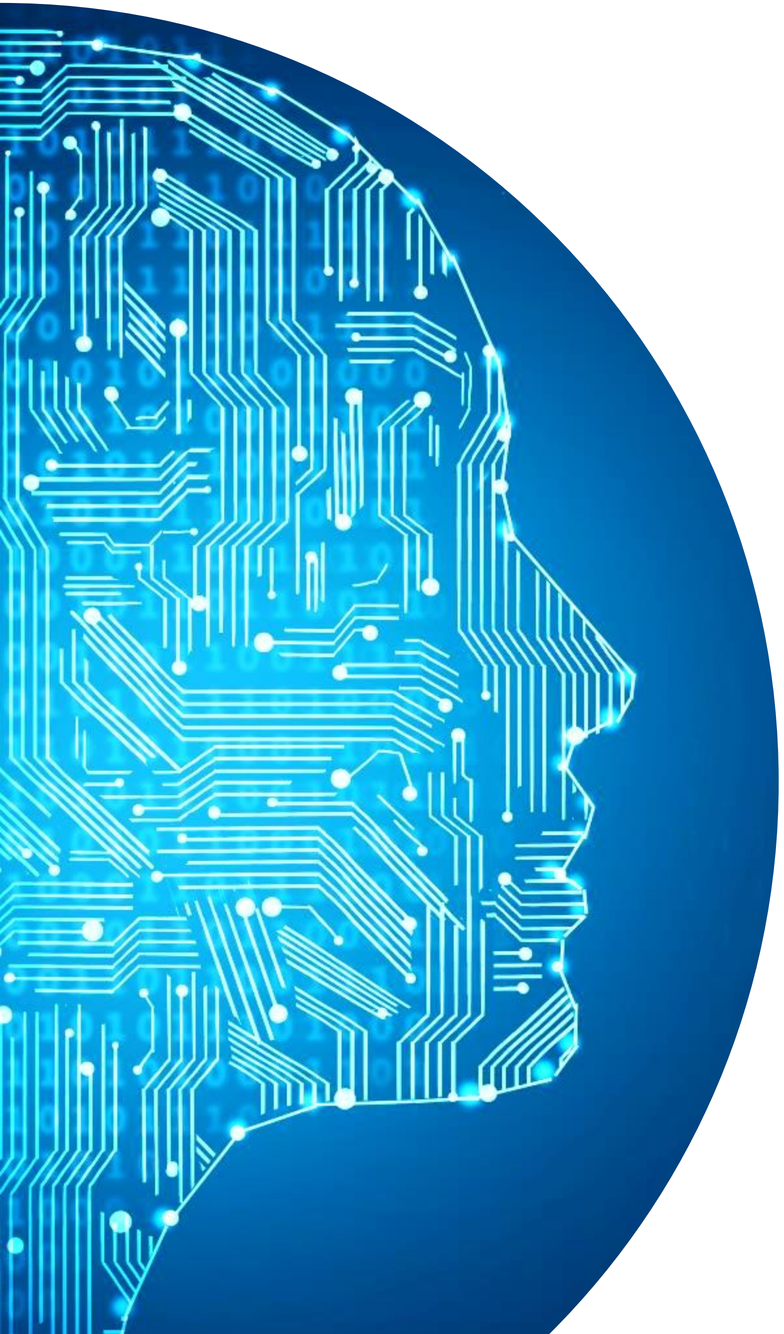
OSM usage

Onboarding

(descriptors, packages and process)

Gulsum Atici
(Canonical)

12/06/2023



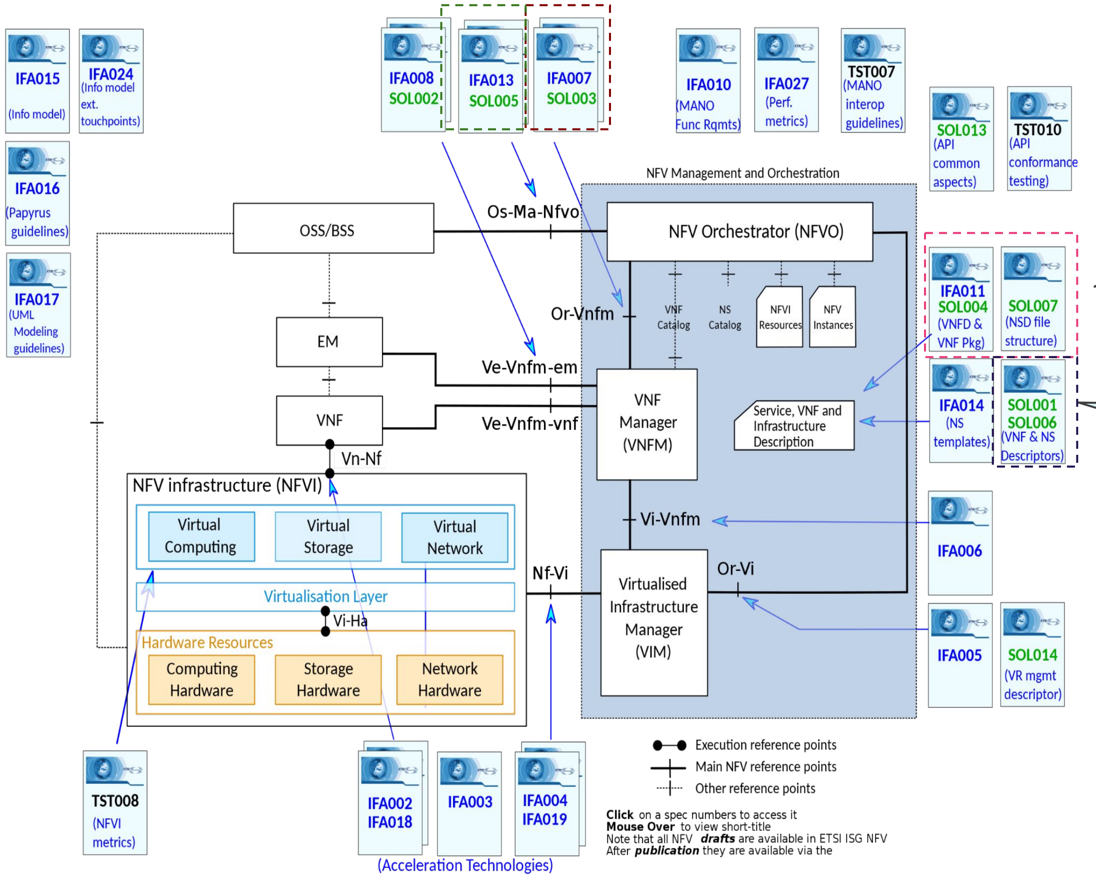
Agenda

- OSM Concepts
- NF Onboarding Process
  - What is NF onboarding ?
  - Onboarding Requirements
  - Onboarding Stages

OSM Concepts

- Information Model & Packages
- VNFd, NSd
- VNF/KNF/PNF/Network Service

OSM Concepts: Specs and Information Model



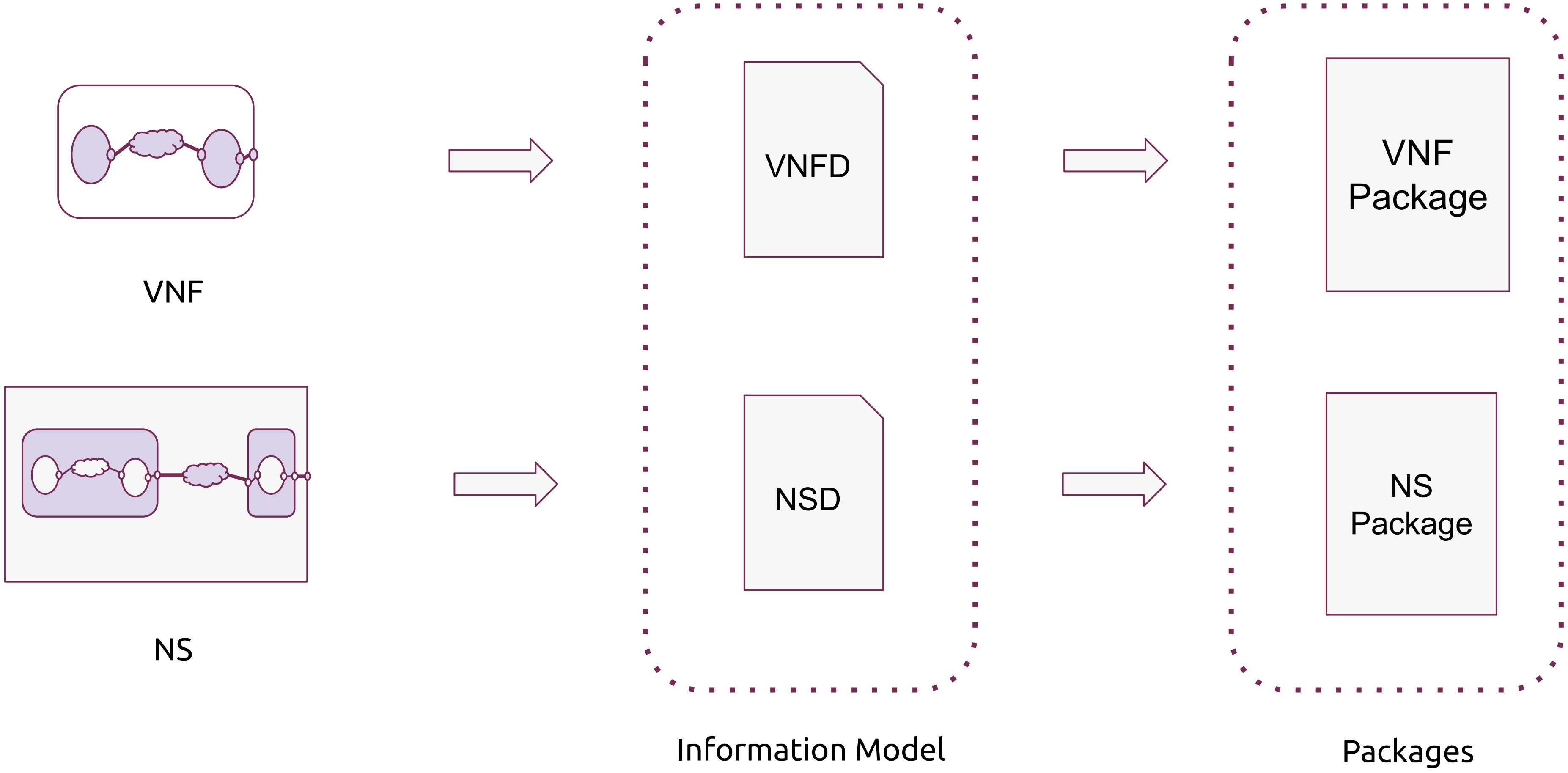
IFA015 (Info model)
IFA024 (Info model ext. touchpoints)
IFA016 (Papyrus guidelines)
IFA017 (UML Modeling guidelines)
IFA008 SOL002
IFA013 SOL005
IFA007 SOL003
IFA010 (MANO Func Rqmts)
IFA027 (Perf. metrics)
TST007 (MANO interop guidelines)
SOL013 (API common aspects)
TST010 (API conformance testing)
NFV Management and Orchestration
OSS/BSS
Os-Ma-Nfvo
NFV Orchestrator (NFVO)
Or-Vnfm
VNF Catalog
NS Catalog
NFVI Resources
NFV Instances
EM
Ve-Vnfm-em
VNF
Ve-Vnfm-vnf
VNF Manager (VNFM)
Service, VNF and Infrastructure Description
Vn-Nf
NFV infrastructure (NFVI)
Virtual Computing
Virtual Storage
Virtual Network
Virtualisation Layer
Vi-Ha
Hardware Resources
Computing Hardware
Storage Hardware
Network Hardware
Vi-Vnfm
Nf-Vi
Virtualised Infrastructure Manager (VIM)
Or-Vi
IFA011 SOL004 (VNFD & VNF Pkg)
SOL007 (NSD file structure)
IFA014 (NS templates)
SOL001 SOL006 (VNF & NS Descriptors)
IFA006
IFA005
SOL014 (VR mgmt descriptor)
TST008 (NFVI metrics)
IFA002 IFA018
IFA003
IFA004 IFA019
(Acceleration Technologies)
Execution reference points
Main NFV reference points
Other reference points
Click on a spec numbers to access it
Mouse Over to view short-title
Note that all NFV ***drafts*** are available in ETSI ISG NFV
After ***publication*** they are available via the

OSM NF and NS package formats are based on SOL004 and SOL007

OSM data model and descriptors are derived from SOL006

OSM IM reference link:
https://osm.etsi.org/docs/user-guide/latest/11-osm-im.html

OSM Concepts: IM & Packages



VNF

NS

VNFD

NSD

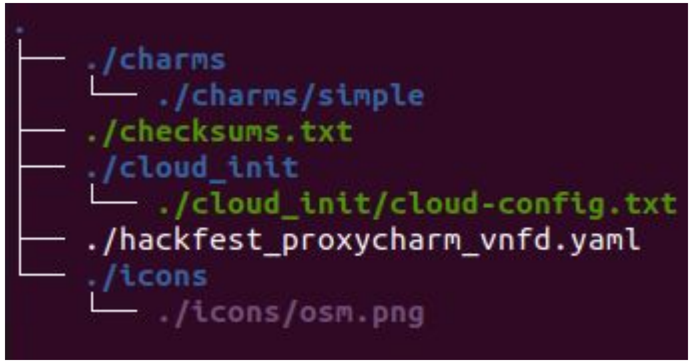
Information Model

VNF Package

NS Package

Packages

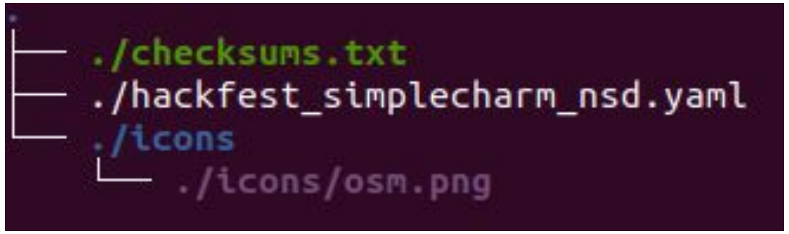
OSM Concepts: VNF/NS Packages



```
.
├── ./charms
│   └── ./charms/simple
├── ./checksums.txt
├── ./cloud_init
│   └── ./cloud_init/cloud-config.txt
├── ./hackfest_proxycharm_vnfd.yaml
└── ./icons
    └── ./icons/osm.png
```

The VNF package may contain the following files and directories:

```
| - vnfd.yaml (VNF descriptor file in YAML format)
| - README (Contains README about this VNF package)
| - checksum.txt (Contains checksum for each file in the package)
| - images\ (Directory containing all the VM images for this VNF)
| - scripts\ (Directory containing custom scripts for lifecycle events
and configuration primitives)
| - icons\ (Directory containing the icons)
| - charms\ (Directory containing the configuration charms/plugins)
| - helm-charts\ (Directory containing the helm-charts)
```

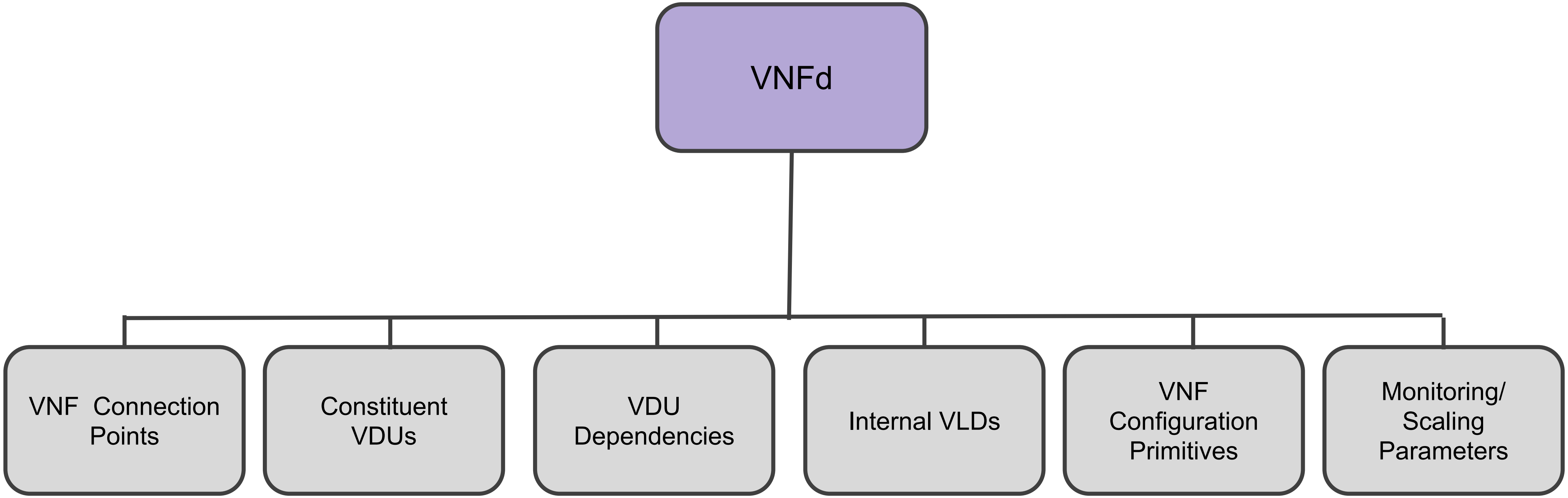


```
.
├── ./checksums.txt
├── ./hackfest_simplecharm_nsd.yaml
└── ./icons
    └── ./icons/osm.png
```

The NS package may contain the following files and directories:

```
| - nsd.yaml (NS descriptor in YAML format)
| - README (Information regarding the NS)
| - checksum.txt (Contains checksum for each file in package)
| - scripts\ (Directory containing the scripts NS lifecycle events and primitives)
| - icons\ (Directory containing icons and logs for the network wervice)
| - charms\ (Directory containing the scripts NS lifecycle events and primitives)
```

OSM Concepts: VNFd


VNFd

VNF Connection Points

Constituent VDUs

VDU Dependencies

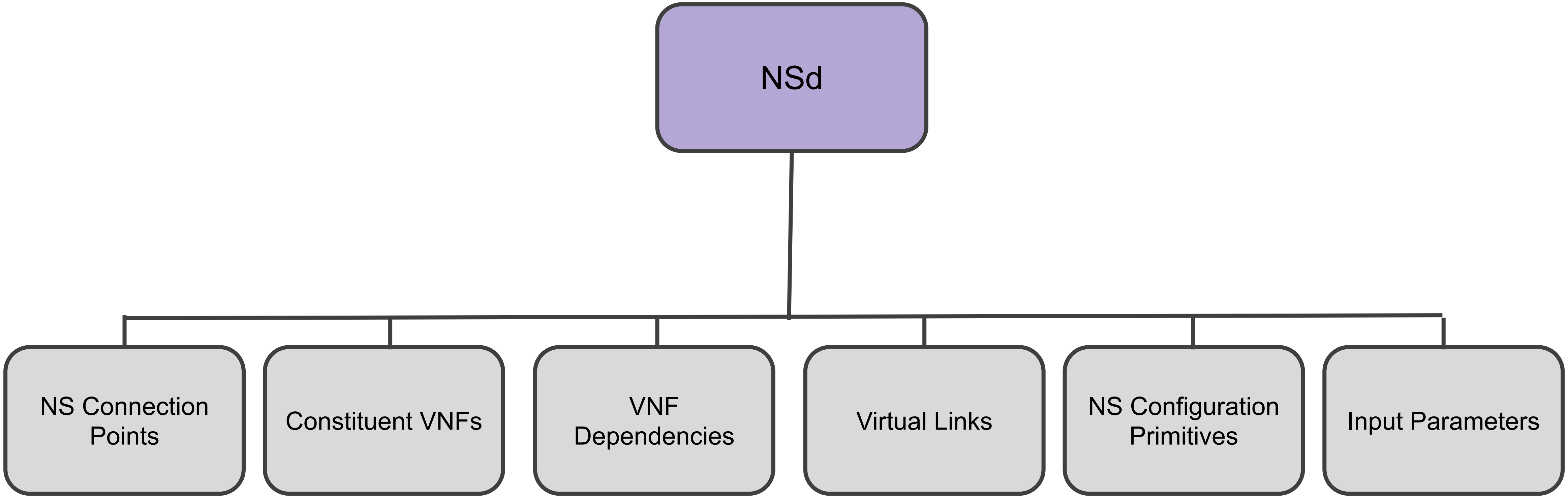
Internal VLDs

VNF Configuration Primitives

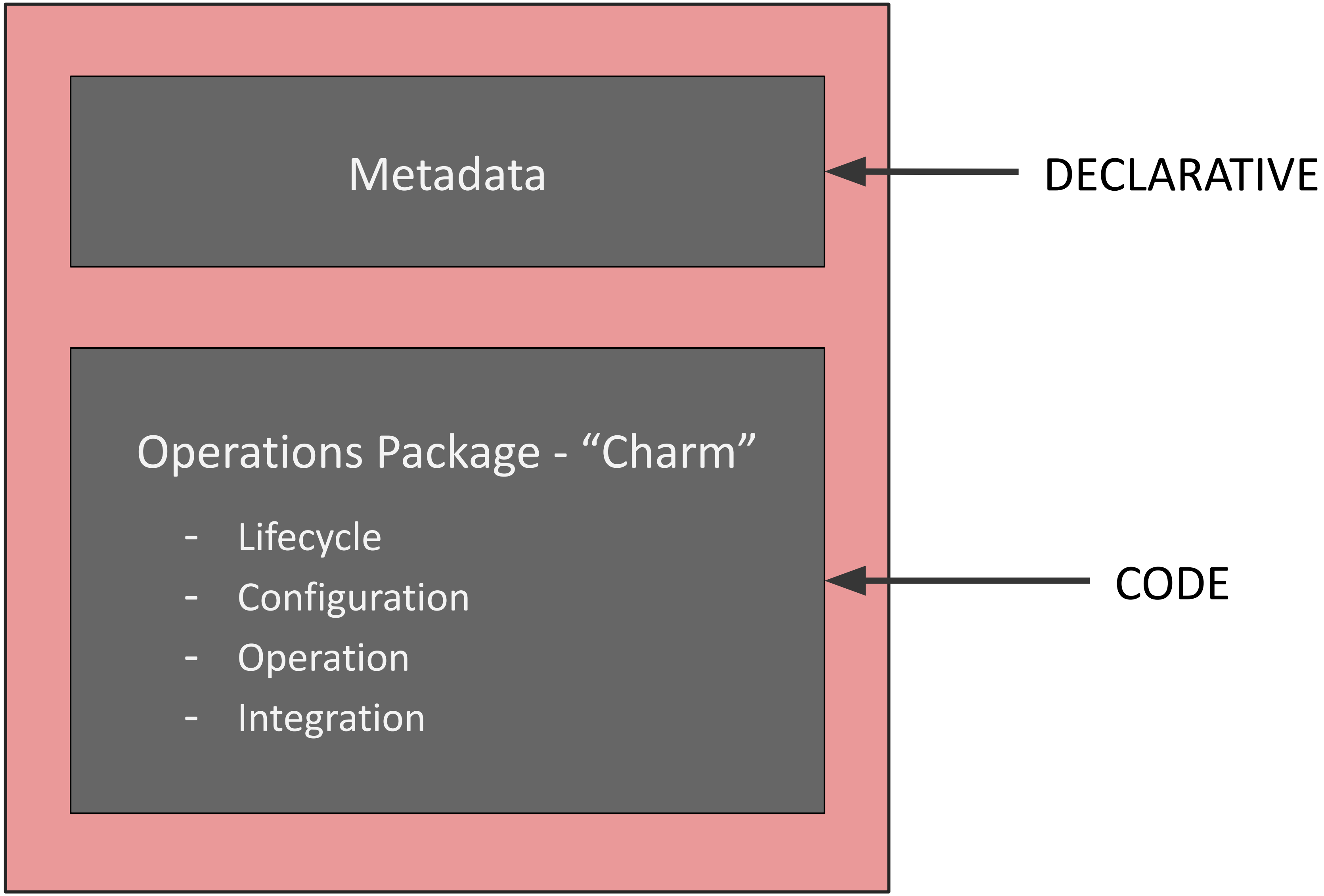
Monitoring/ Scaling Parameters

OSM Concepts: NSd

Open Source MANO



OSM Concepts: VNFd



Metadata

DECLARATIVE

Operations Package - “Charm”

- Lifecycle
- Configuration
- Operation
- Integration

CODE

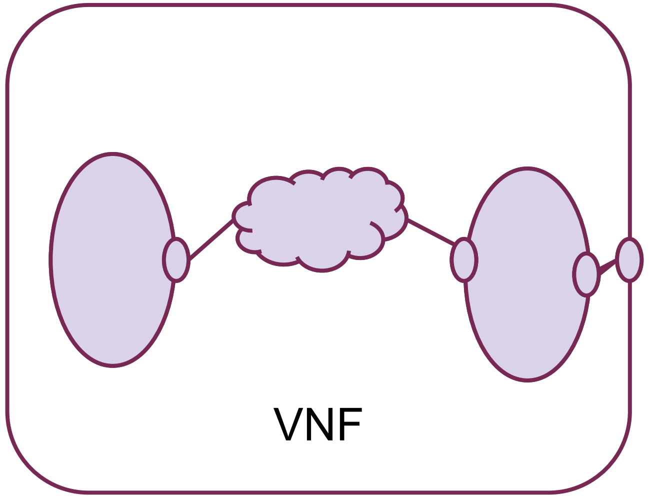
OSM Concepts: VNF

Virtual Network Function

- One or more Deployment Units
- Internal networks
- Internal connection points (interfaces)
- Mapping VDU connections to the networks
- External connection points

VDU: Virtual Deployment Unit

- Virtual Machines
- OSM models vCPUs, RAM, Storage, Interfaces, Performance Capabilities (SR-IOV, EPA)



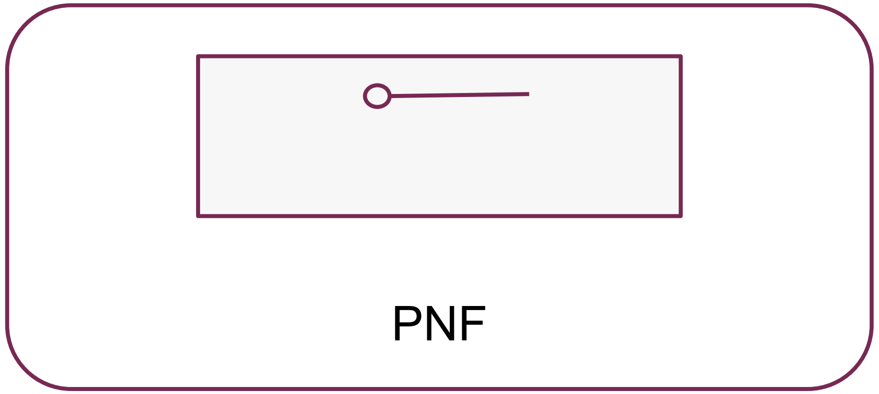
OSM Concepts: PNF

Physical Network Function

- Models an already existing physical application.
- It uses the same concepts as the VNF

PDU: Physical Deployment Unit

- Already existing application
- No control over the Lifecycle
- Perform operations on it



© ETSI

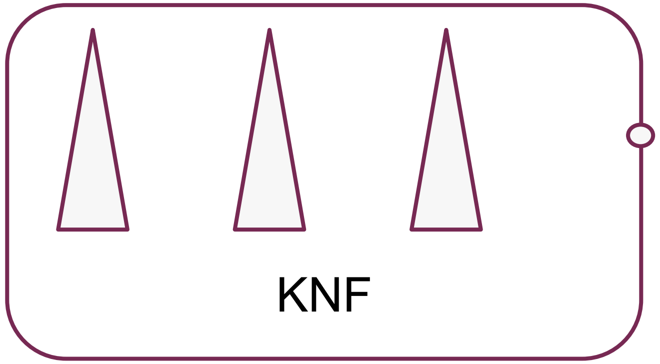
OSM Concepts: KNF

Kubernetes Network Function

- Composed of one or more KDUs and the connection points to communicate with other KNFs/VNFs/PNFs
- Specify the networks that need to be already present in the K8s cluster

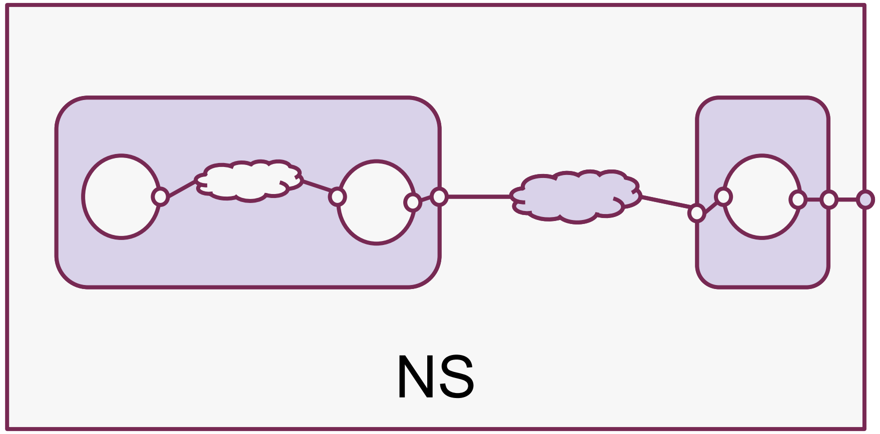
KDU: Kubernetes Deployment Unit

- Kubernetes applications
- A KDU represents a Helm Chart or a Juju Bundle



OSM Concepts: Network Service

- One or more xNFs
- Networks
- Mapping xNF connections to the networks
- Network Service level connection points



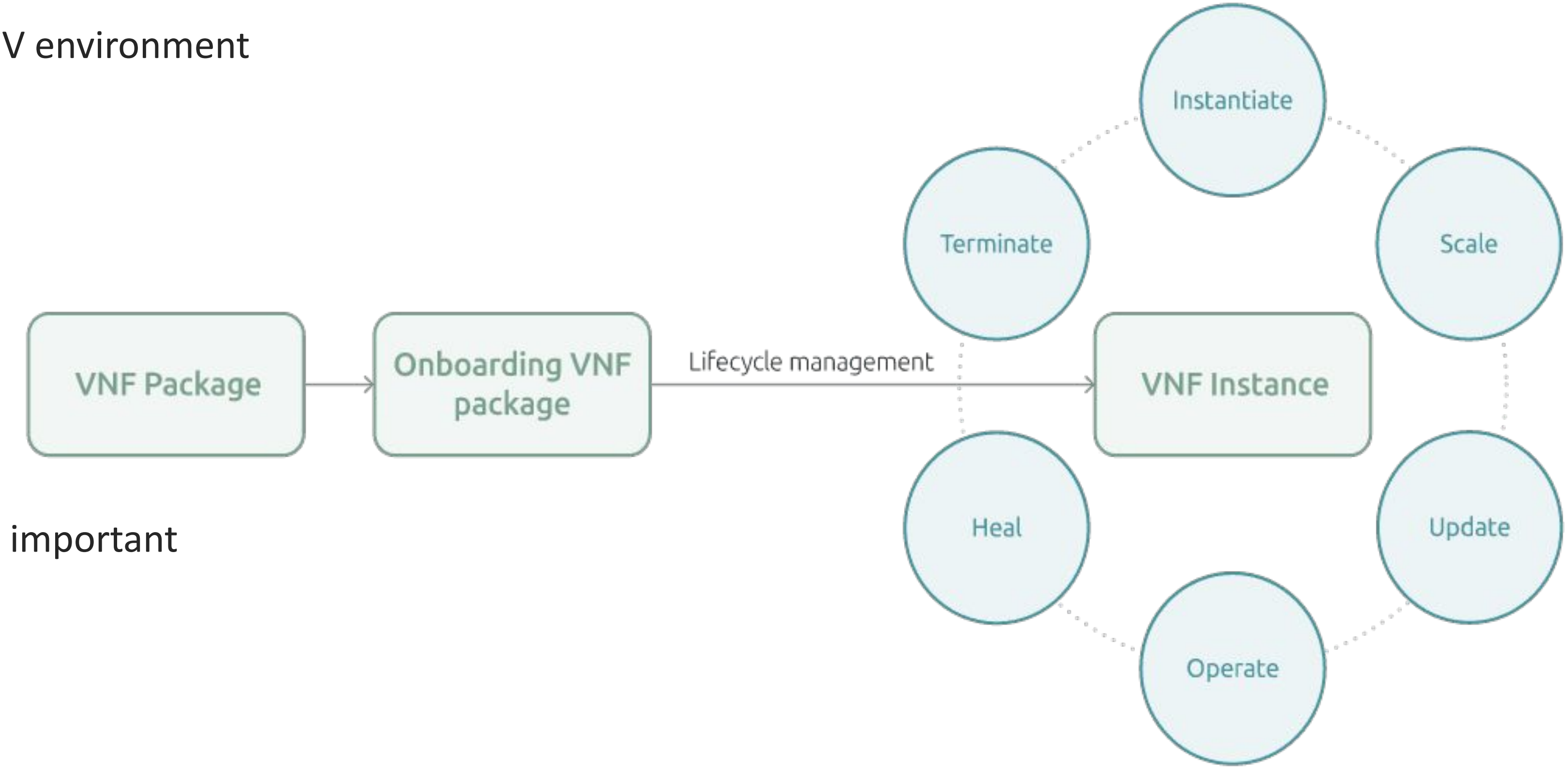
NF Onboarding Process

- What is NF Onboarding ?
- Onboarding Requiremets
- Onboarding Stages

What is Onboarding ?

Network Function Onboarding

Network function onboarding is an automated methodology for bringing new network functions into an operational NFV environment



Modeling the network function is very important

So NFs can be:

- instantiated
- scaled in and out
- fully utilized to deliver features

Network Function Onboarding

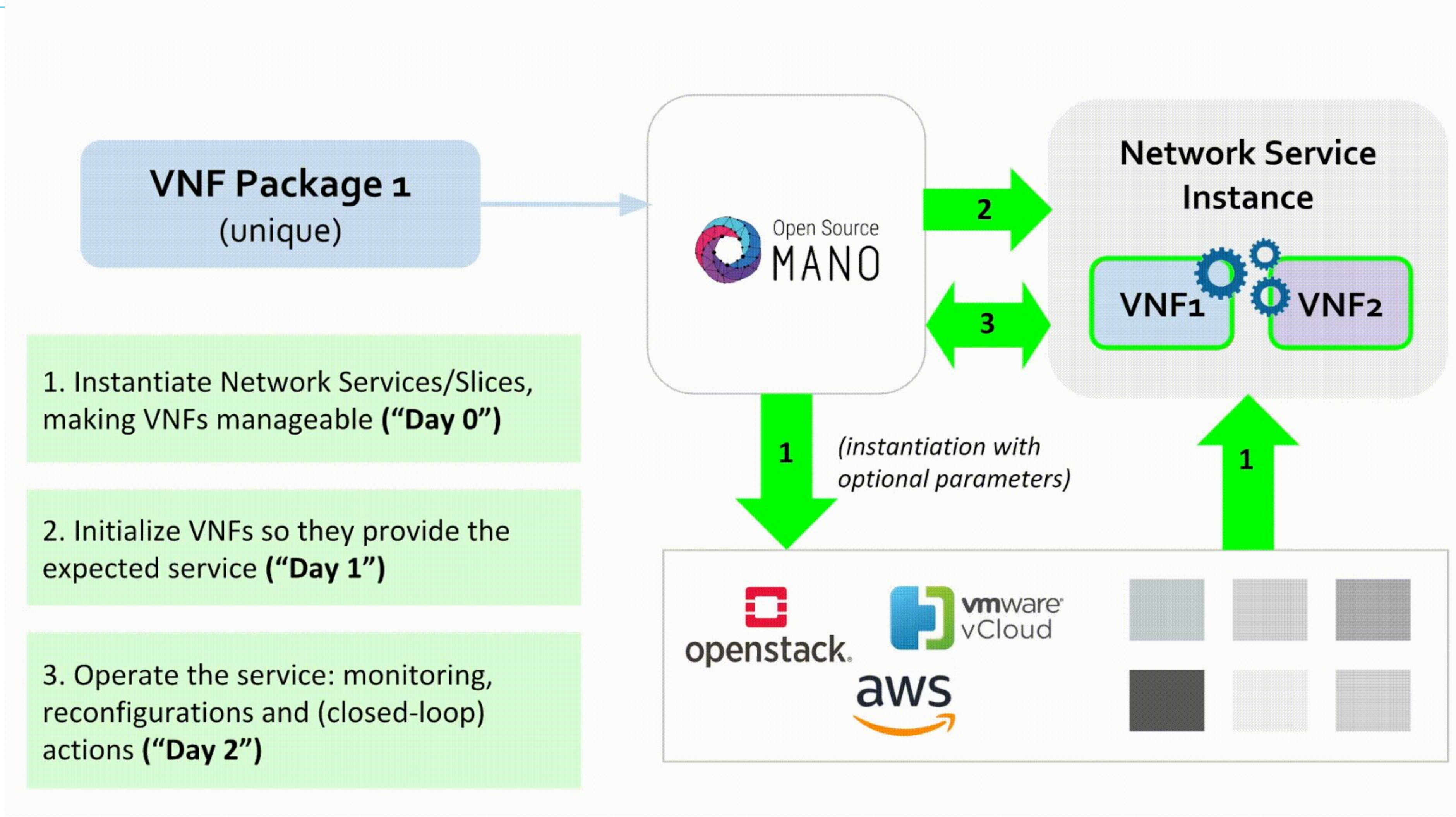
The complete onboarding process implies **producing a VNF Package** that will be part of the **OSM catalogue** for its inclusion in a Network Service.

The onboarded VNF should aim to fulfill the lifecycle stages it requires to function properly.

Hence, the resulting package, should include **all the requirements, instructions and elements** to achieve the NS lifecycle stages, which are:

- Basic instantiation (Day-0)
- Service initialization (Day-1)
- Runtime operations (Day-2)

NF Onboarding Process



1. Instantiate Network Services/Slices, making VNFs manageable **("Day 0")**

2. Initialize VNFs so they provide the expected service **("Day 1")**

3. Operate the service: monitoring, reconfigurations and (closed-loop) actions **("Day 2")**

Onboarding Requirements

- Day-0 requirements
- Day-1 requirements
- Day-2 requirements

Onboarding Requirements

Each lifecycle stage targets specific configurations in the VNF. These are:

- **Management setup** during instantiation (Day-0)
- **Service initialization** right after instantiation (Day-1)
- **Re-configuration** during runtime (Day-2)

In order to provide a VNF with many capabilities for each lifecycle stage as possible, the related specific requirements should be adressed.

Day-0 requirements

During the Day-0 stage, the VNF is i**nstantiated and the management access is established** so that the VNF can be configured at a later stage.

The main requirements to achieve this are:

- **Description of each VNF component:**
  - The main function of every VNF component (**VDU**) should be clearly described
  - **Internal VLD** Network that interconnects VDUs within a VNF
  - **Internal CP** Element internal to a VNF, maps VDU interfaces to internal VLDs
- **Description of each NS component:**
  - **External VLD** Network that interconnects different VNFs within a NS
  - **External CP** Element exposed externally by a VNF, maps VDU interfaces to external VLDs

Day-1 requirements

The aim of Day-1 stage is to **configure the VNF so it starts providing the expected service.**

The main requirements are:

- **Identifying dependencies between components**
  - Identify instantiation parameters or special timing requirements
  - Components needing parameters from other components or from the infrastructure
  - Components depending on others for their configuration to be initialized
- **Defining the required configuration for service initialization**

  This initial configuration should activate the service delivered by the VNF
- **Identifying the need for instantiation parameters**

  The VNF Day-1 configuration may require some parameters passed at instantiation time in order to fulfill the needs of the particular environment or of other VNFs in the Network Service.

Day-2 requirements

The main objetive of Day-2 are to be able to **re-configure the VNF** so its behavior can be modified during runtime.

The main requirements are:

- **Identifying dependencies between components**

  Identify if a VNF component requires a parameter coming from other component for fulfilling runtime operations
- **Defining all possible configurations for runtime operations**

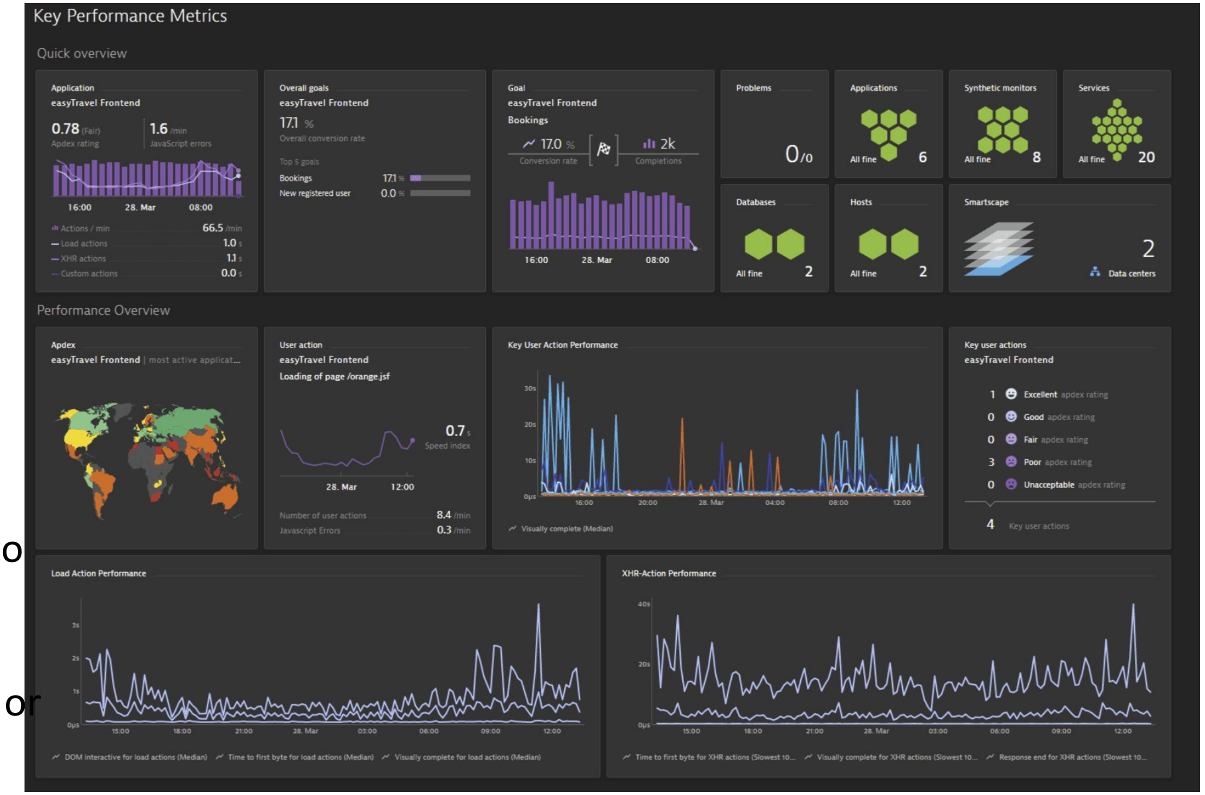
  The set of configurations should be available to be triggered from the orchestrator, either manually by the operator

  That set of configurations need to be incorporated by the mechanism that the generic VNF Manager implements. The set of configurations can be provided by **Python scripts, Ansible playbooks, VNF-specific commands that run over SSH, REST API calls, etc.**

Day-2 requirements

- **Defining key performance indicators**

  The metrics that are relevant to the VNF should be specified, either if they are supposed to be collected from the **infrastructure** or directly from the **VNF**



- **Defining closed-loop operations**

  Closed-loop operations are actions triggered by the status of a particular metric. The main use cases include:

  - **Auto-scaling**: a VNF component scales horizontally (out/in) to match the current demand.
  - **Auto-healing:** a VNF component is re-instantiated, reloaded or reconfigured based on a service status.

Onboarding Stages

- Day 0: VNF Instantiation & Management setup
- Day 1: VNF Services Initialization
- Day 2: VNF Runtime Operations

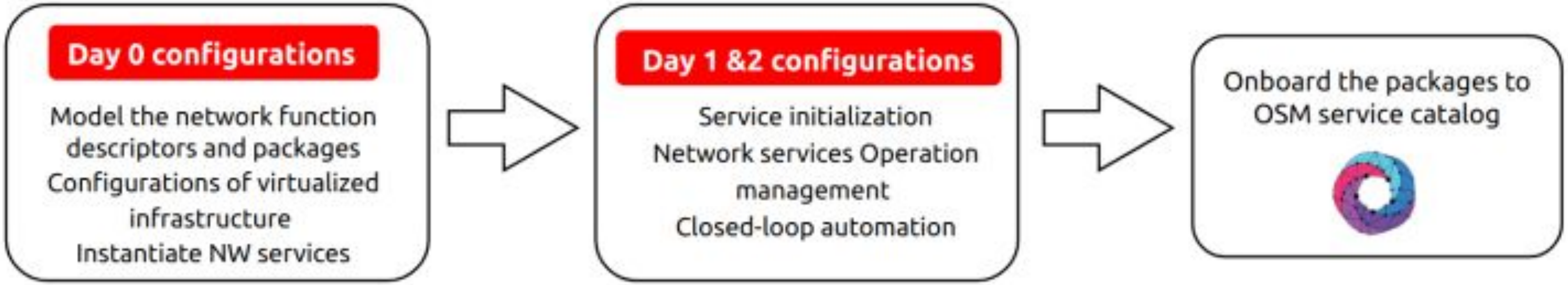
Onboarding Stages

OSM (Open Source MANO) supports the onboarding of all types of network functions whether it is

- Virtualized
- Containerized
- Physical in nature

The stages for onboarding NFs (network functions) in OSM include

- Creating the NF package that specifies Day-0 (basic instantiation)
- Day-1 (service initialization)
- Day-2 (runtime operations) configurations



Day 0 configurations

Model the network function descriptors and packages
Configurations of virtualized infrastructure
Instantiate NW services

Day 1 &2 configurations

Service initialization
Network services Operation management
Closed-loop automation

Onboard the packages to OSM service catalog

Day 0: VNF Instantiation & Management setup

The Day-0 configuration includes following steps:

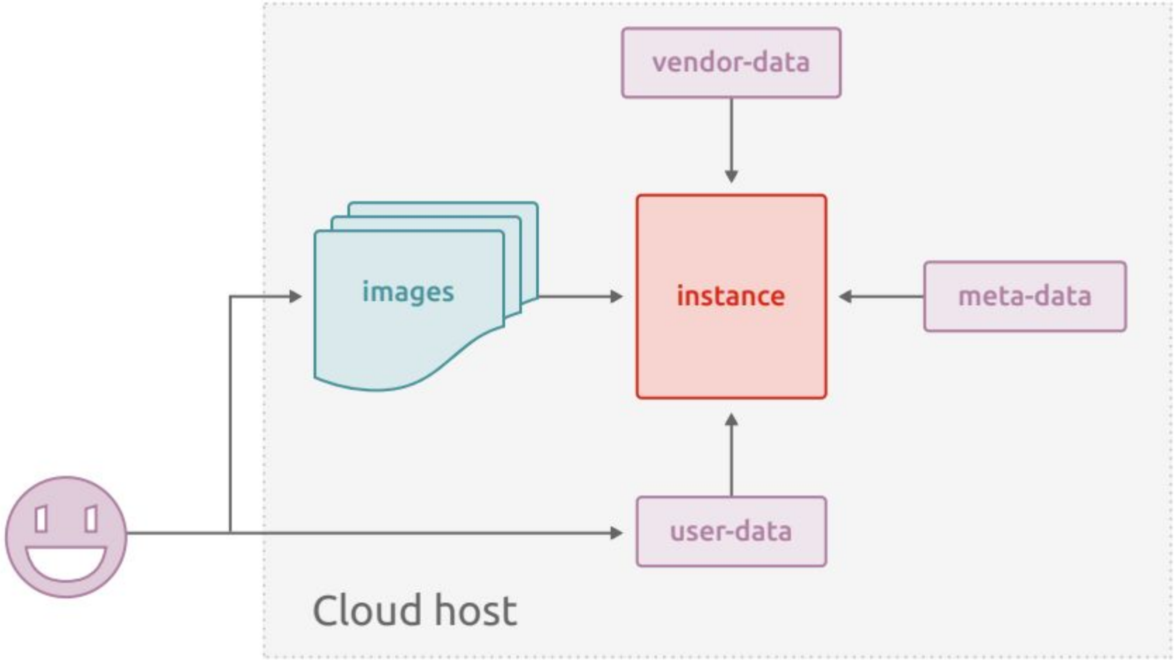
- Build an initial package by using network service descriptors (VNFD/NSD).
- Include the basic configurations for the interconnections of network components.
- Integrate the **cloud-init scripts** in the descriptors for primary configurations needed to build up a network service like the OS boot requirements, **setting up a hostname, adding SSH keys, configuring network devices and users.**
- Configure the virtualized infrastructure

  Enable the EPA capabilities in the descriptors like **Hugepages, CPU pinning, SR-IOV**, or any other data accelerated features depending on the functional requirements.

Day 0: VNF Instantiation & Management setup

Sample Day-0 operations like:

- Building and adding cloud-init scripts
- Setting a default locale
- Setting an instance hostname
- Generating instance SSH private keys or defining passwords
- Adding SSH keys to a user’s .ssh/authorized_keys
- Setting up ephemeral mount points
- Configuring network devices
- Adding users and groups
- Adding files



Day 0: VNF Instantiation & Management setup

Prepare the descriptor so that it accurately details the VNF requirements, **prepare cloud-init scripts** (if needed), and **identify parameters** that may have to be provided at later stage.

Building the initial package

Build a VNF package from scratch by using OSM client is a requirement to perform this stage

# For the VNF Package

```
osm package-create --base-directory /home/ubuntu --image myVNF.qcow2 --vcpu 1 --memory 4096
--storage 50 --interfaces 2 --vendor OSM vnf vLB
```

# For the NS Package

```
osm package-create --base-directory /home/ubuntu --vendor OSM ns vLB
```

Day 0: VNF Instantiation & Management setup

Testing Instantiation of the VNF Package

- Instantiating the VNF with all the required VDUs, images, initial state and NFVI requirements
- Making the VNF manageable from OSM
- Once the VNF Descriptor has been updated with all the Day-0 requirements, its folder needs to be repackaged by using the OSM CLI which validates and uploads the package to the catalogue

  ```
  osm vnfpkg-create [VNF Folder]
  ```

To test this package, the NS can be launched using the OSM client:

```
osm ns-create --ns_name [ns name] --nsd_name [nsd name] --vim_account [vim name] \
--ssh_keys [list of public key files]
```

At launch time, extra instantiation parameters can be passed so that the VNF.

Day 1: VNF Services Initialization

Expose the services inside the VNF to be automatically initialized right after the VNF instantiation

To achieve this in OSM is to **build a Charm and include it in the descriptor**.

The operations code is called “Charm”, and it can handle the **lifecycle, configuration, integration, and actions/primitives** in your workloads.

There are two kinds of Charms:

- Proxy Charms: If you are using a fixed image for your workload, which CANNOT be modified
- Native Charms: if the the workload CAN be modified, then the code can live in the same workload

Besides charms, there is another way of configuring network functions, powered by **helm-based execution environments** launched as PODs.

Day 1: VNF Services Initialization

The Day1 operations should be specified in the VNF descriptor.

Day1 parameters are defined at two different levels:

- VDU-level: used when a VDU needs configuration, **which is different than the VDU used for managing the VNF**
- VNF-level: for the “management VDU”, used when the **configuration applies to the VDU exposing a interface for managing the whole VNF**.

Juju-based or Helm-based execution environments could be used to run those parameters.

Day 2: VNF Runtime Operations

Typical operations like following requires additional configuration in the VNF packages:

- Reconfigurations needed after the service is running
- Monitoring of the specific metrics for the infrastructure
- Scaling on the basis of monitoring analysis
- Operations to enable closed-loop automation
- Different orchestration capabilities **like auto-scaling, auto-healing, self-monitoring**, etc.

To achieve reconfiguration in OSM Day-2 primitives needs be added to the descriptor

Day-2 primitives are actions invoked on demand, which is defined under the **config-primitive** block at the VNF or VDU level

Day 1/2 by using Helm based EEs

Since OSM version 8, NF configurations can also be done with Helm-based execution environments, which **deploy a pair of extra pods in the K8s namespace**.These PODs follow the VNF lifecycle (as charms do) to run day-1 and day-2 primitives

The EE interacts with the managed NF (e.g. via SSH) to manage NFs by OSM.

OSM communicates with its EE to trigger actions via **gRPC** calls.

In order to ease the NF onboarding tasks, there is already a helm chart template available named **eechart** to be included under helm-charts subdirectory of the VNF package to run actions easily

```
helm-charts
└── eechart
    ├── Chart.yaml
    ├── charts
    ├── source
    │   ├── install.sh
    │   ├── install_nginx.sh
    │   ├── mylib.py
    │   ├── playbook.yaml
    │   ├── run_ssh.sh
    │   └── vnf_ee.py
    ├── templates
    └── values.yaml
```

Day 1/2 EEs and config primitives with Helm

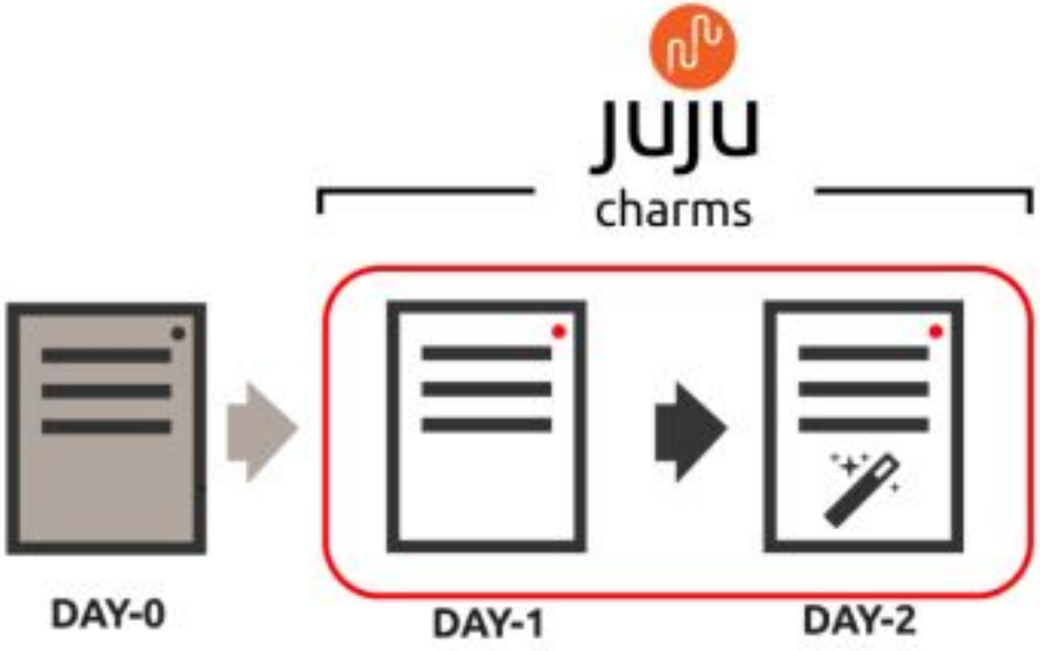
```
vnfd:
...
  lcm-operations-configuration:
   operate-vnf-op-config:
    day1-2:
    - config-primitive:
     config-access:
      ssh-access:
       default-user: ubuntu
       required: true
     execution-environment-list:
     - external-connection-point-ref: vnf-mgmt-ext
      helm-chart: eechart
      id: sample_ee
     id: sample_ee-vnf
     initial-config-primitive:
     - execution-environment-ref: sample_ee
      name: config
      parameter:
      - name: ssh-hostname
       value: <rw_mgmt_ip>
      - name: ssh-username
       value: ubuntu
      seq: 1
```

Day 1/2 operations by using Charms

OSM component VCA (VNF configuration and abstraction) is responsible for the day-1 and day-2 operations

Using charms for Day1/2 operation process:

- Creating a charm as an operator for the network function.
- Defining the primitives/actions in charms that need to be performed after the instantiation of service i.e. **Initial charm configuration, authentication, and actions** required during the service is up and running, etc.
- Integrating the configured charm in the OSM descriptors



Day 1/2 EEs and config primitives with Juju Charms

```
vnfd:
...
  lcm-operations-configuration:
   operate-vnf-op-config:
    day1-2:
     - id: ha_proxy_charm-vnf
      execution-environment-list:
      - id: simple-ee
       juju:
        charm: simple
      config-access:
       ssh-access:
        default-user: ubuntu
        required: true
      config-primitive:
  …
```

```
vnfd:
   ...
     config-primitive:
     - name: touch
      execution-environment-ref: simple-ee
      parameter:
      - data-type: STRING
       default-value: /home/ubuntu/touched
       name: filename
     initial-config-primitive:
     - name: config
      execution-environment-ref: simple-ee
      parameter:
      - name: ssh-hostname
       value: <rw_mgmt_ip>
      - name: ssh-password
       value: osm4u
      seq: 1
     - name: touch
      execution-environment-ref: simple-ee
      parameter:
      - data-type: STRING
       name: filename
       value: /home/ubuntu/first-touch
      seq: 2
```

Open Source MANO by ETSI

Thank You!