

# ONAP IM discussion



Ericsson

# Two CNF onboarding and orchestration options



The Standards People

ETSI MANO NFV VNFD Sol001  
IFA 11 based  
Standardized in ETSI NFV

- ETSI started assuming infrastructure capability based upon VM capabilities.
- The VNFD was created to allow the VNFM to support the necessary lifecycle capabilities.
- Support for CNF was added on top but inheriting the structure from VM based deployments

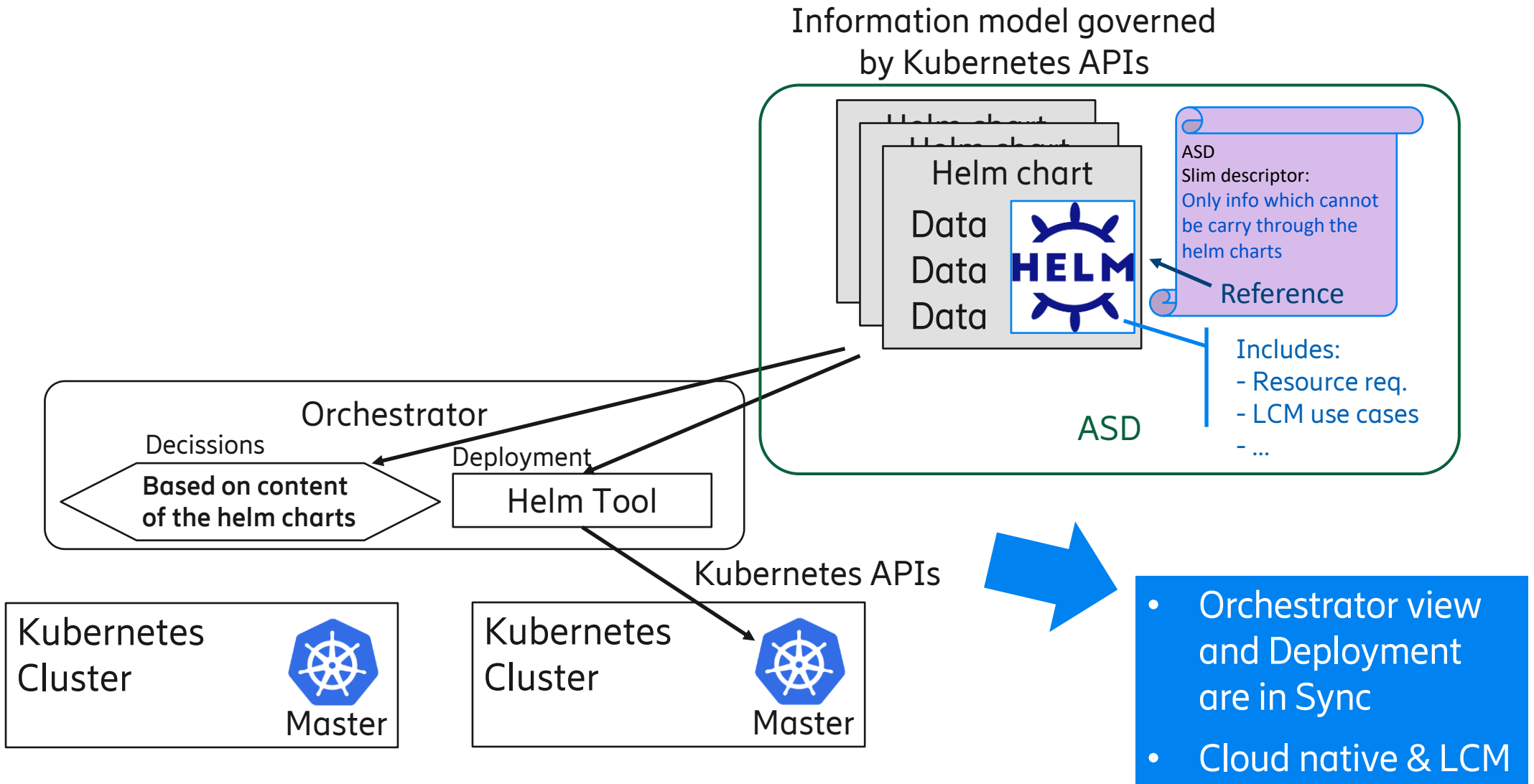


Application Service Descriptor  
**ASD**

An alternative proposed by the  
CNF taskforce in ONAP

- Since then, the infrastructure has become more capable and taken significant SW LCM functionality and it is captured in the Helm chart.
- The VNFD is not required.
- ASD builds upon cloud native approach and tooling.

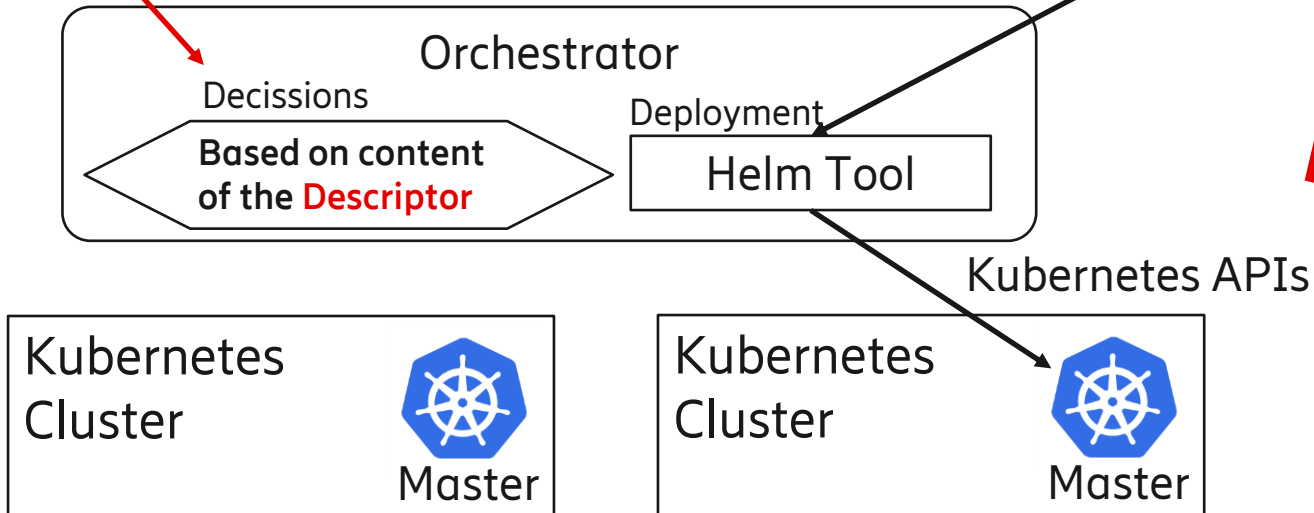
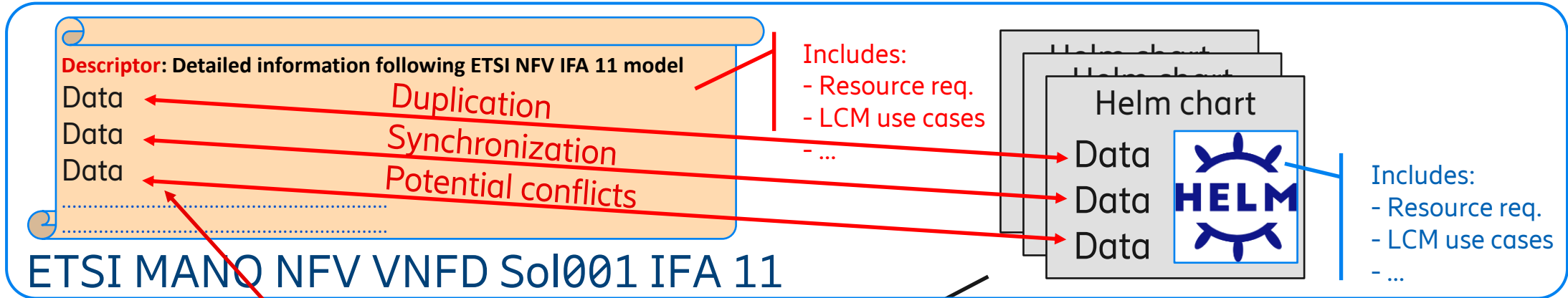
# ASD based modeling and deployment of CNFs



# ETSI MANO NFV VNFD Sol001 IFA 11 approach



Information model governed by  
Descriptors content

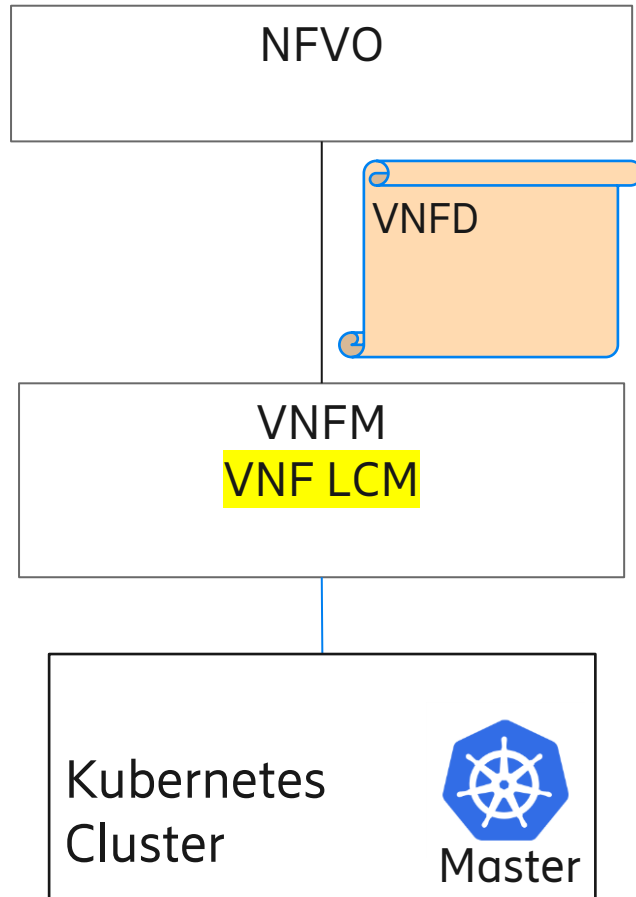


- Orchestrator view and Deployment potentially in conflict
- Potentially conflicting Data & LCM handling

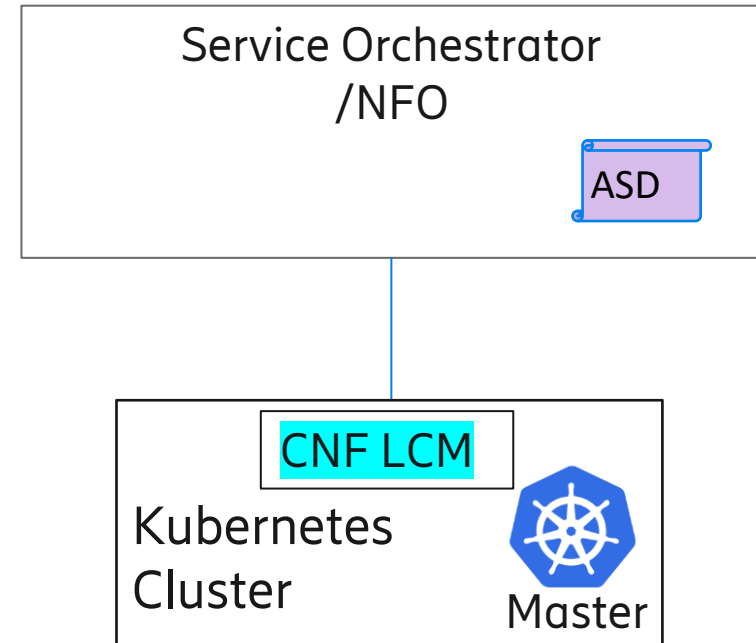
# Comparing LCM handling



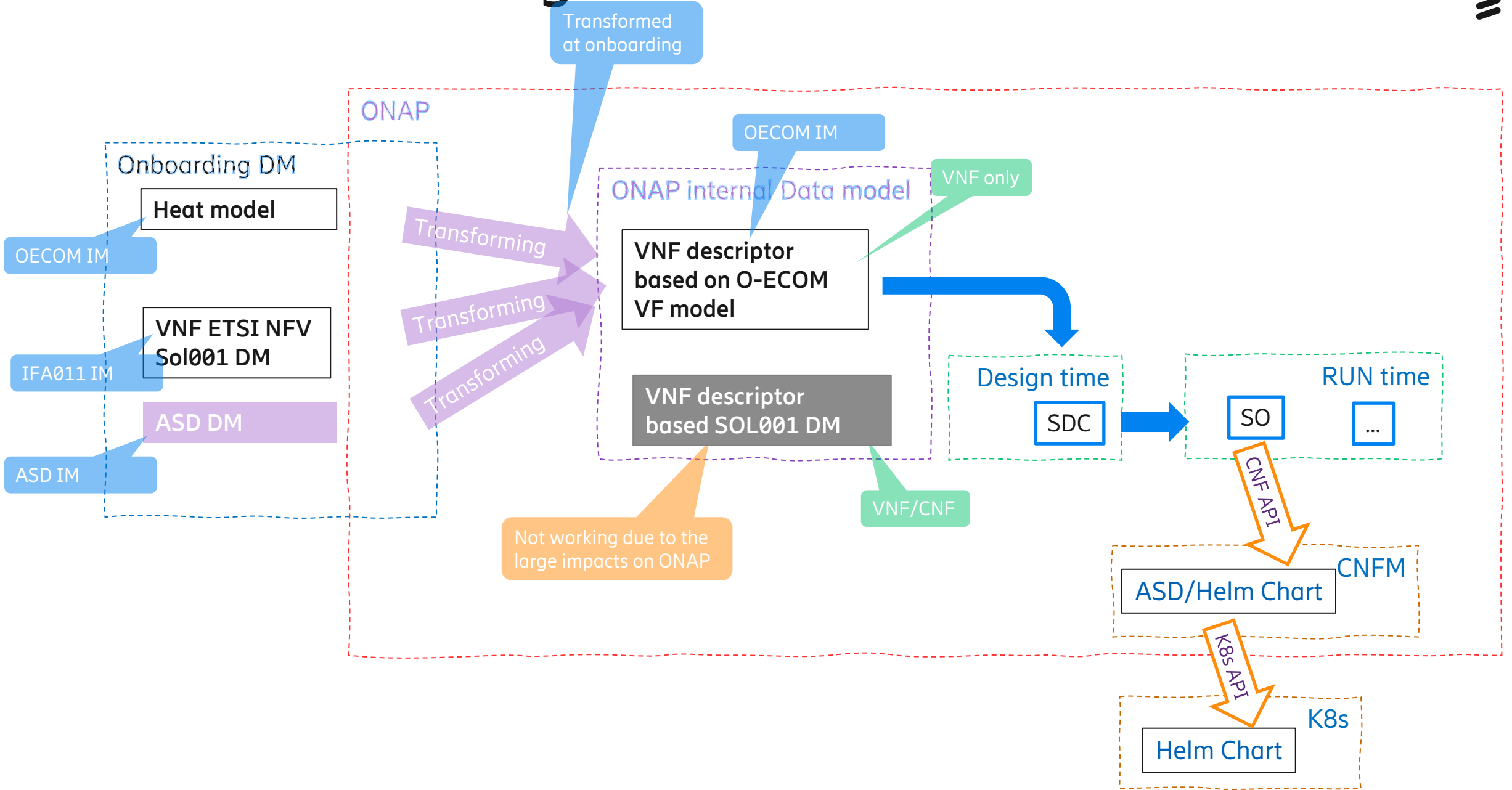
ETSI MANO NFV VNFD Sol001 IFA 11 approach



ASD based approach



# ONAP xNF modelling with ASD



# Response to comments



- CNF resource IM is specified by particular cloud native specifications, e.g., K8s.
- For instance, this [K8s document](#) contains all the information listed in the right side. All the info is managed by K8s, not SO.
- Proposal: adding a reference under *Deployment Item Information Element*, that “all cloud native CNF resource IM is specified by particular cloud native specifications. [K8s document](#) is an example.”

---

Fred’s comments

---

Missing resource (CPU, Memory, Storage, ...) requirement information

---

Missing L2/L3 Protocol and Address information

---

Missing Software Image information

---

Missing Security Rules information

---

Missing Affinity/Anti-Affinity information

---

Missing Scaling Information

---

Missing Healing information

---

Missing Service/VIP mapping information

---

Missing configuration information

---

Missing monitoring information

---

Missing upgrade/downgrade information

# Kubernetes: Resource requirement (CPU, Memory, Storage, ...) requirement information [LINK](#)



The screenshot shows the Kubernetes documentation page for configuring CPU resources. The main heading is "Specify a CPU request and a CPU limit". Below the heading, there is a paragraph explaining that to specify a CPU request, the `resources.requests` field should be used, and to specify a CPU limit, the `resources.limits` field should be used. The page then provides a configuration file for a Pod with a container that requests 0.5 CPU and has a limit of 1 CPU. The configuration file is shown in a code block, and a red arrow points to a summary of its fields.

```
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo
  namespace: cpu-example
spec:
  containers:
  - name: cpu-demo-ctr
    image: vish/stress
    resources:
      limits:
        cpu: "1"
      requests:
        cpu: "0.5"
  args:
  - -cpus
  - "2"
```

The summary of the configuration file fields is as follows:

- apiVersion: v1
- kind: Pod
- metadata:
  - name: cpu-demo
  - namespace: cpu-example
- spec:
  - containers:
    - name: cpu-demo-ctr
    - image: vish/stress
    - resources:
      - limits:
        - cpu: "1"
      - requests:
        - cpu: "0.5"
  - args:
    - -cpus
    - "2"



# Kubernetes: Scaling Information [LINK](#)



**Horizontal Pod Autoscaling**

In Kubernetes, a *HorizontalPodAutoscaler* automatically updates a workload resource (such as a *Deployment* or *StatefulSet*), with the aim of automatically scaling the workload to match demand.

Horizontal scaling means that the response to increased load is to deploy more Pods. This is different from *vertical* scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods that are already running for the workload.

If the load decreases, and the number of Pods is above the configured minimum, the *HorizontalPodAutoscaler* instructs the workload resource (the *Deployment*, *StatefulSet*, or other similar resource) to scale back down.

Horizontal pod autoscaling does not apply to objects that can't be scaled (for example: a *DaemonSet*.)

The *HorizontalPodAutoscaler* is implemented as a Kubernetes API resource and a *controller*. The resource determines the behavior of the controller. The horizontal pod autoscaling controller, running within the Kubernetes control plane, periodically adjusts the desired scale of its target (for example, a *Deployment*) to match observed metrics such as average CPU utilization, average memory utilization, or any other custom metric you specify.

There is [walkthrough example](#) of using horizontal pod autoscaling.

### How does a HorizontalPodAutoscaler work?

```
graph TD; HPA[Horizontal Pod Autoscaler] --> RC[RC / Deployment]; subgraph RC; direction TB; Scale[Scale]; end; RC --> Pod1[Pod 1]; RC --> Pod2[Pod 2]; RC --> PodN[Pod N];
```

**behavior:**

- scaleDown:**
  - stabilizationWindowSeconds: 300
- policies:**
  - type: Percent
  - value: 100
  - periodSeconds: 15
- scaleUp:**
  - stabilizationWindowSeconds: 0
- policies:**
  - type: Percent
  - value: 100
  - periodSeconds: 15
  - type: Pods
  - value: 4
  - periodSeconds: 15
  - selectPolicy: Max

# Kubernetes: Healing information [LINK](#)



The screenshot shows the Kubernetes documentation page for 'Specifying a Disruption Budget for your Application'. The page includes a search bar, a navigation menu on the left, and the main content area. The main content area has a breadcrumb trail: 'Kubernetes Documentation / Tasks / Run Applications / Specifying a Disruption Budget for your Application'. The title is 'Specifying a Disruption Budget for your Application'. Below the title, it says 'FEATURE STATE: Kubernetes v1.21 [stable]'. The text explains that the page shows how to limit the number of concurrent disruptions. There are sections for 'Before you begin' and 'Protecting an Application with a PodDisruptionBudget'. The 'Before you begin' section lists requirements for the Kubernetes server version and user permissions. The 'Protecting an Application with a PodDisruptionBudget' section lists four steps: 1. Identify what application you want to protect with a PodDisruptionBudget (PDB). 2. Think about how your application reacts to disruptions. 3. Create a PDB definition as a YAML file. 4. Create the PDB object from the YAML file. The 'Identify an Application to Protect' section lists the most common use cases: Deployment, ReplicationController, ReplicaSet, and StatefulSet.

Besides fundamental Kubernetes capability to restore crashed Pods automatically, there is an advanced handling for keeping critical replicas always running. This is very useful when executing e.g. rolling upgrade.

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: zookeeper
```

# Q&A



Did we answer your all questions?

