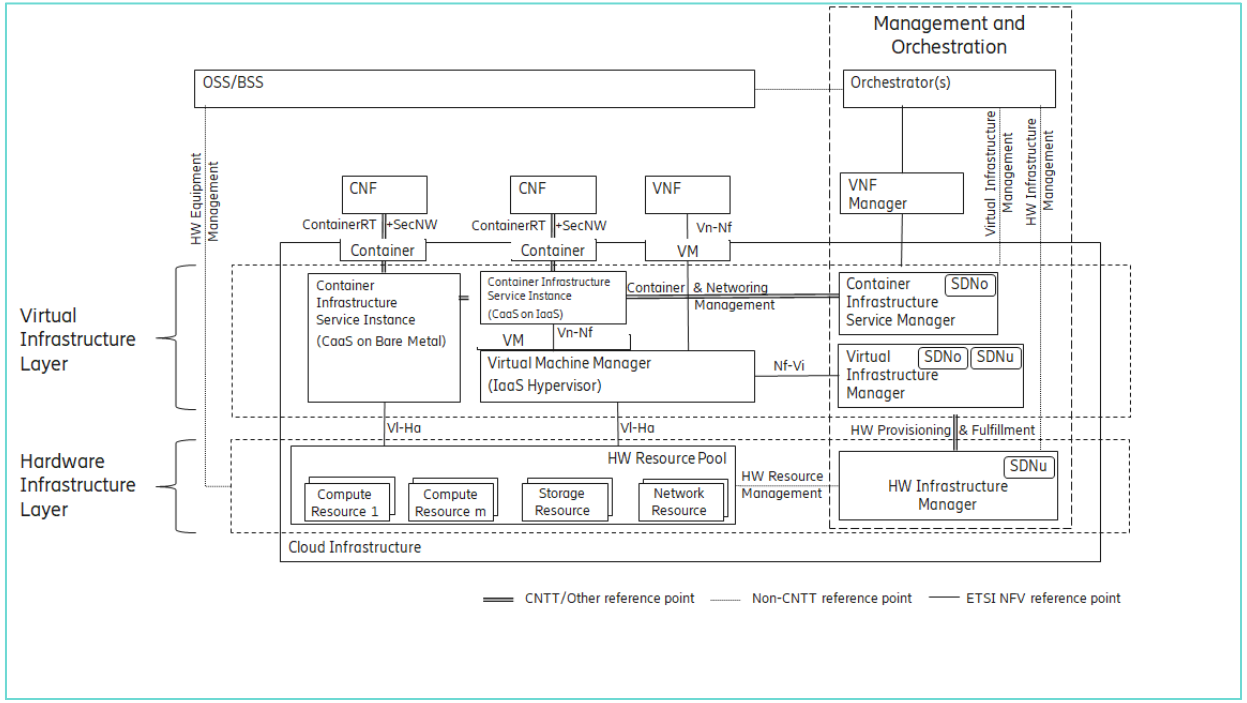
Load Balancing in the Infrastructure

Per Andersson (per@kaloom.com)

Load Balancing and the reference model

- The reference model encompasses
  - Virtual Machine Manager
  - Virtual Infrastructure Manager
  - Container Infrastructure Service Instances
  - Container Infrastructure Service Manager
- How does Load Balancing relate to these two different types of environments?



Management and Orchestration
OSS/BSS
Orchestrator(s)
HW Equipment Management
Virtual Infrastructure Management
HW Infrastructure Management
CNF
CNF
VNF
VNF Manager
ContainerRT
+SecNW
ContainerRT
+SecNW
Vn-Nf
Container
Container
VM
Virtual Infrastructure Layer
Container Infrastructure Service Instance (CaaS on Bare Metal)
Container Infrastructure Service Instance (CaaS on IaaS)
Container & Networing Management
Container Infrastructure Service Manager
SDNo
VM
Vn-Nf
Virtual Machine Manager (IaaS Hypervisor)
Nf-Vi
Virtual Infrastructure Manager
SDNo
SDNu
Vl-Ha
Vl-Ha
HW Provisioning & Fulfillment
Hardware Infrastructure Layer
HW Resource Pool
Compute Resource 1
Compute Resource m
Storage Resource
Network Resource
HW Resource Management
SDNu
HW Infrastructure Manager
Cloud Infrastructure
CNTT/Other reference point
Non-CNTT reference point
ETSI NFV reference point

Load Balancing in Openstack

- Two ways to do it
  - Use the included LBaaS
    - Octavia
  - Bring you own
    - Instantiate a VM with the flavor of LB you wan to use
- The “user” is responsible for configuration and orchestration of the needed components and services
  - Networks
  - VIP
  - LB instance
    - LBaaS
    - VM
  - Endpoints and their addresses
  - Adding/removing endpoints to/from LB endpoint set
- Load Balancing is explicitly managed and controlled by the user
  - Networking is simple and straightforward to set up using the Neutron APIs

Load Balancing in “Standard” Kubernetes

- Traditional L4 Load Balancing as in an IaaS type of system does not exist in Kubernetes.
- K8s is built around concepts like
  - POD
  - Workload
    - Deploymnet
    - ReplicaSet
    - StatefulSet
    - DaemonSet
    - ....
  - Controller
  - Service
- Load Balancing/scaling is intrinsic to Kubernetes and is built in.
  - Controlled by creating a service
- A typical user never has to care about networking resources and network infrastructure,

Define a set of PODs in Kubernetes

- Example workload
  - Deployment with 4 replicas of the POD my-app
- There are no ip addresses defined in the specification!
- Late binding
  - A pod instance’s ip address is not known until the instance has been started and the CNI plugin has assigned an address to it
- A new ip address is typically assigned to each new “incarnation” of a pod instance

- Define a deployment

```
apiVersion: apps/v1
kind: Deployment
metadata: name: my-app-deployment
spec:
  selector:
      matchLabels:
          app: my-app
      replicas: 4
      template:
          metadata:
            labels:
              app: my-app
      spec:
          containers:
            - name: my-app
             image: my-app-1.2.3
             ports:
              - containerPort: 123
```

Define a Service in Kubernetes

- Example Service
  - A service linked to the deployment with 4 replicas of the pods matching the selector “my-app”
- There are no ip addresses defined in the specification!
- Late binding
  - A service is assigned it’s ip address once the specification is “consumed” by the system
- The ip address remains the same for the lifetime of the service
- A and AAAA records are typically added to the internal Kubernetes DNS service that maps the service name to the ip address(es) of the service.
- Internal Load Balancing is automatically set up
  - Automatic mapping between the ip address of the service towards the current set of ip addresses used by pods that match the selector “my-app”
  - New session request towards service address is load balanced over the set of pods

- Define a service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  spec:
    selector:
      app: my-app
    ports:
      - protocol: TCP
        port: 231
        targetPort: 123
```

Kubernetes and multi networking

- It is possible to add extra networks to Kubernetes and attach NW interfaces in the pod's network namespace using Kubernetes extensions
- The problem is that these extensions is not interacting well with the overall semantics of the Kubernetes networking principles
- Neither is there good support for "standard" network operations and orchestration that you have in an IaaS type of setup
  - It is not possible to add an interface to a running pod
  - It is not possible to use the built-in load balancing mechanism towards these networks and interfaces
  - It is not possible to use the L3 network policies to restrict communication between pods over these networks
  - It is not trivial to add/remove a network to Kubernetes
- It can be nontrivial to separate traffic towards Kubernetes services and the service provider's network services from services reachable from the added networks
  - Example: How are two default routes to different destinations over two different interfaces and networks managed?

Is it possible to design and instantiate a CNF like a virtual router or load balancer in Kubernetes?

- It is not easy to answer
- You can do this if you have complete control over the Kubernetes installation, basically a “service provider” can provide a Kubernetes system that has these types of functions built in and has added nonstandard network functionality
  - We have done this at Kaloom
- You can today, not do this in a good way as a “normal” user, there is no support for the network plumbing needed
  - It is possible for a “service provider” to extend Kubernetes with functionality that provides these capabilities though

Should Openstack and Kubernetes have the same network semantics, services and APIs in ?

- There are different opinions reading this
- I am personally against it; I don’t believe that it makes sense
  - Let the two type of systems develop in the way that is best for each system
  - Openstack’s network service and APIs have been developed during many years and are stable, do not touch them
  - Invest in development efforts to find the best possible way for Kubernetes to support advanced network services and orchestration

Where to go?

- What is needed to support a typical container/Kubernetes based Networking Function?
- What is the best way to support these types of functions in Kubernetes
  - Add an IaaS inspired network orchestrion functionality, services and APIs to Kubernetes?
  - Extend the existing Kubernetes semantics
    - Support for pods that have interfaces attaches to more than one networks
    - Services with "VIP" addresses that can be used not only for the cluster network
    - Network Policies that work across all the networks
    - Support for ip addresses that are not set by "CNI"
      - DHCP
      - IPv6 autoconfigured addresses
      - Addresses configured by the pod itself
    - …
- This is the problem we must solve
  - The challenge is how find ONE way to do this that is acceptable to the overall Kubernetes community

Anuket