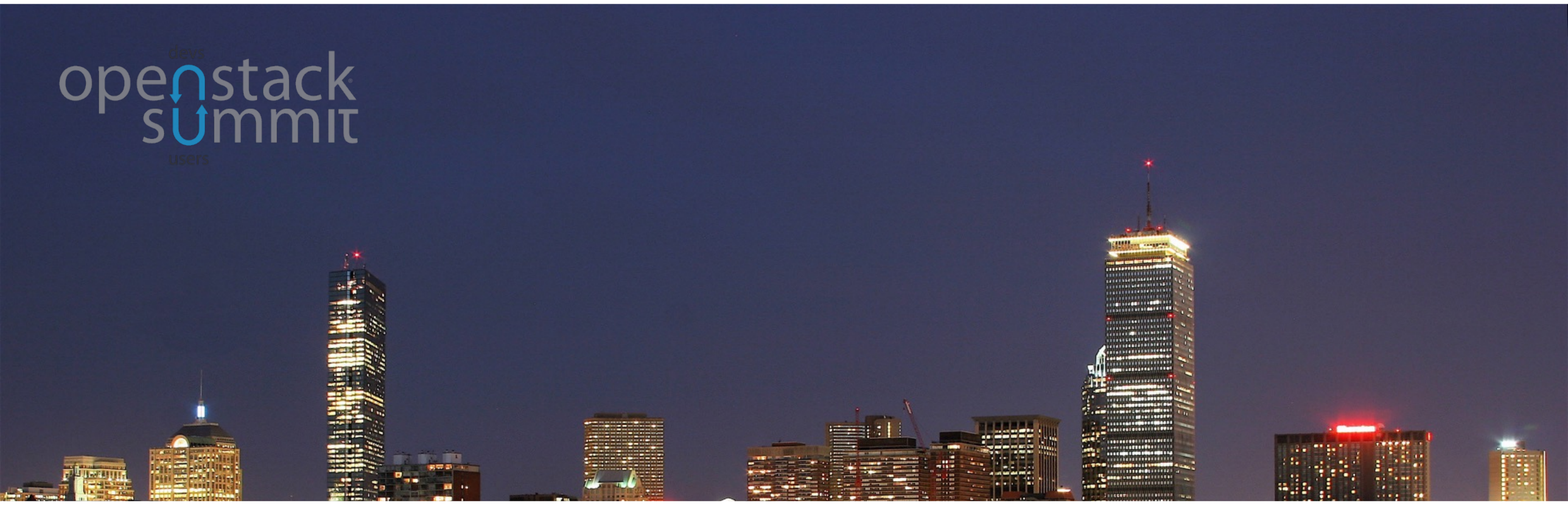


OPENSTACK + KUBERNETES + HYPERCONTAINER

The Container Platform for NFV

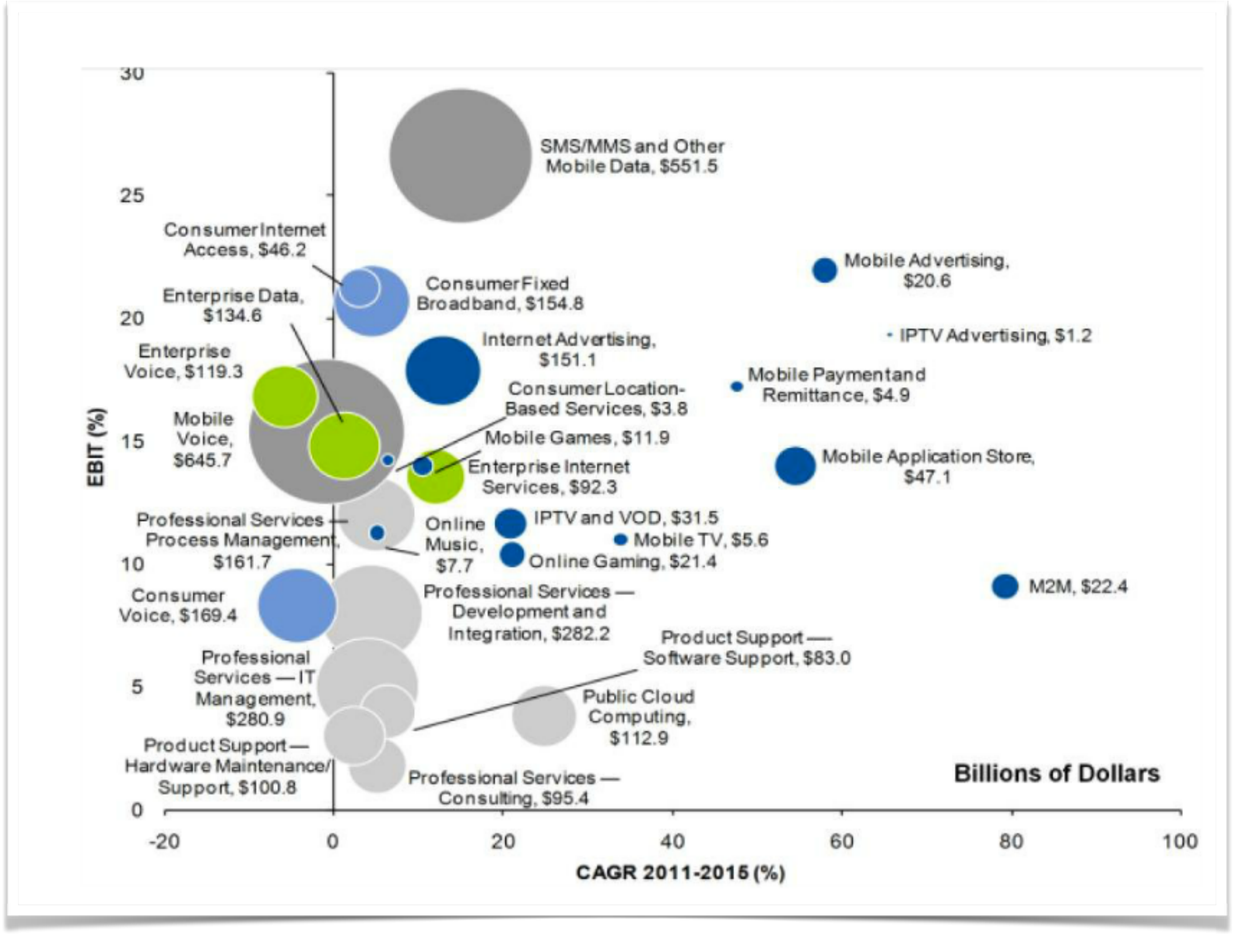
ABOUT ME

- Harry Zhang
  - ID: @resouer
- Coder, Author, Speaker …
- Member of Hyper
- Feature Maintainer & Project Manager of Kubernetes
  - sig-scheduling, sig-node
  - Also maintain: kubernetes/frakti (hypervisor runtime for k8s)

NFV

Network Functions Virtualization: why, and how?

TRENDS OF TELECOM OPERATORS

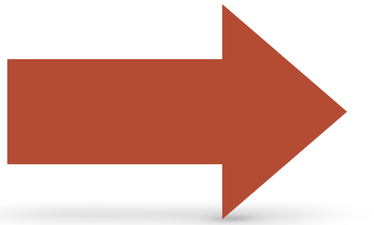


Source: The Gartner Scenario for Communications Service Providers

- Traditional businesses rarely grow
- Non-traditional businesses climb to 8.1% of the whole revenue, even 15%~20% in some operators
- The new four business models:
  - Entertainment & Media
  - M2M
  - Cloud computing
  - IT service

WHAT'S WRONG?

- Pain of telecom network
  - Specific equipments & devices
  - Strict protocol
  - Reliability & performance
  - High operation cost



Long deploy time cost

Complex operation processes

Multiple hardware devices co-exists

Close ecosystem

New business model requires new network functioning

NFV

- Replacing hardware network elements with
  - **software** running on **COTS** computers
  - that may be hosed in **datacenter**
- **Functionalities should be able to:**
  - **locate anywhere most effective or inexpensive**
  - **speedily combined, deployed, relocated, and upgraded**

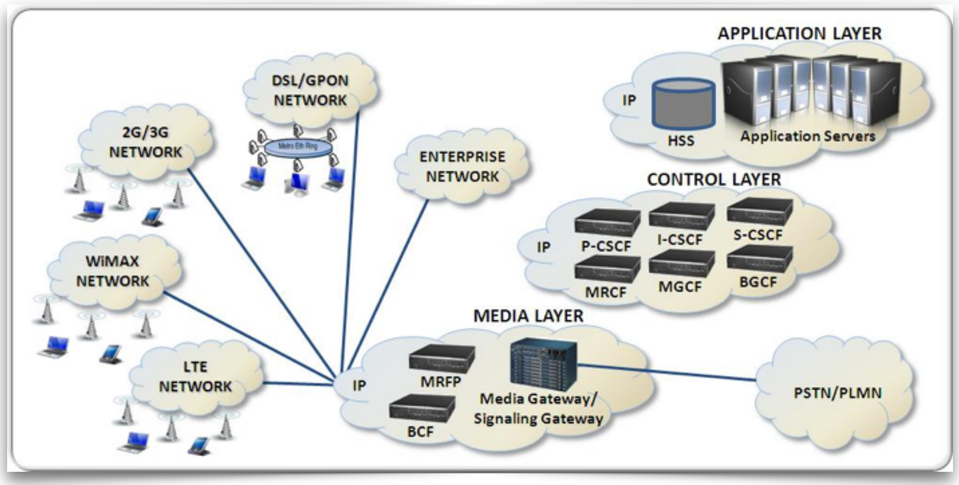
Speedup TTM

Save TCO

Encourage innovation

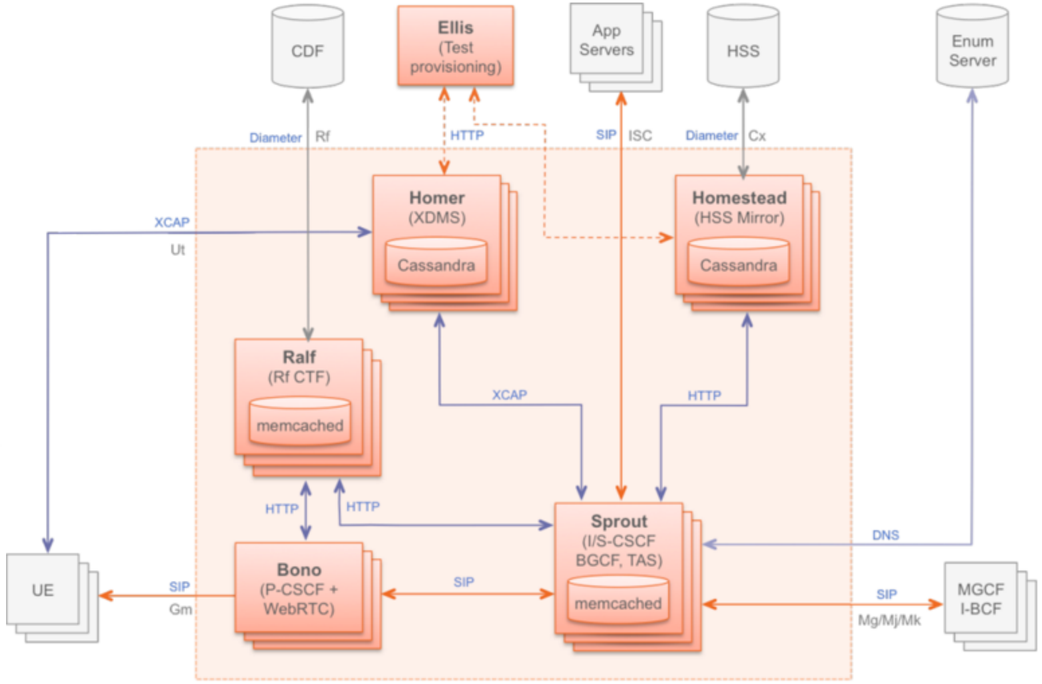
USE CASE

- Project Clearwater
  - Open source implementation of IMS (IP Multimedia Subsystem) for NFV deployment



Devices (physical equipments)

NFV

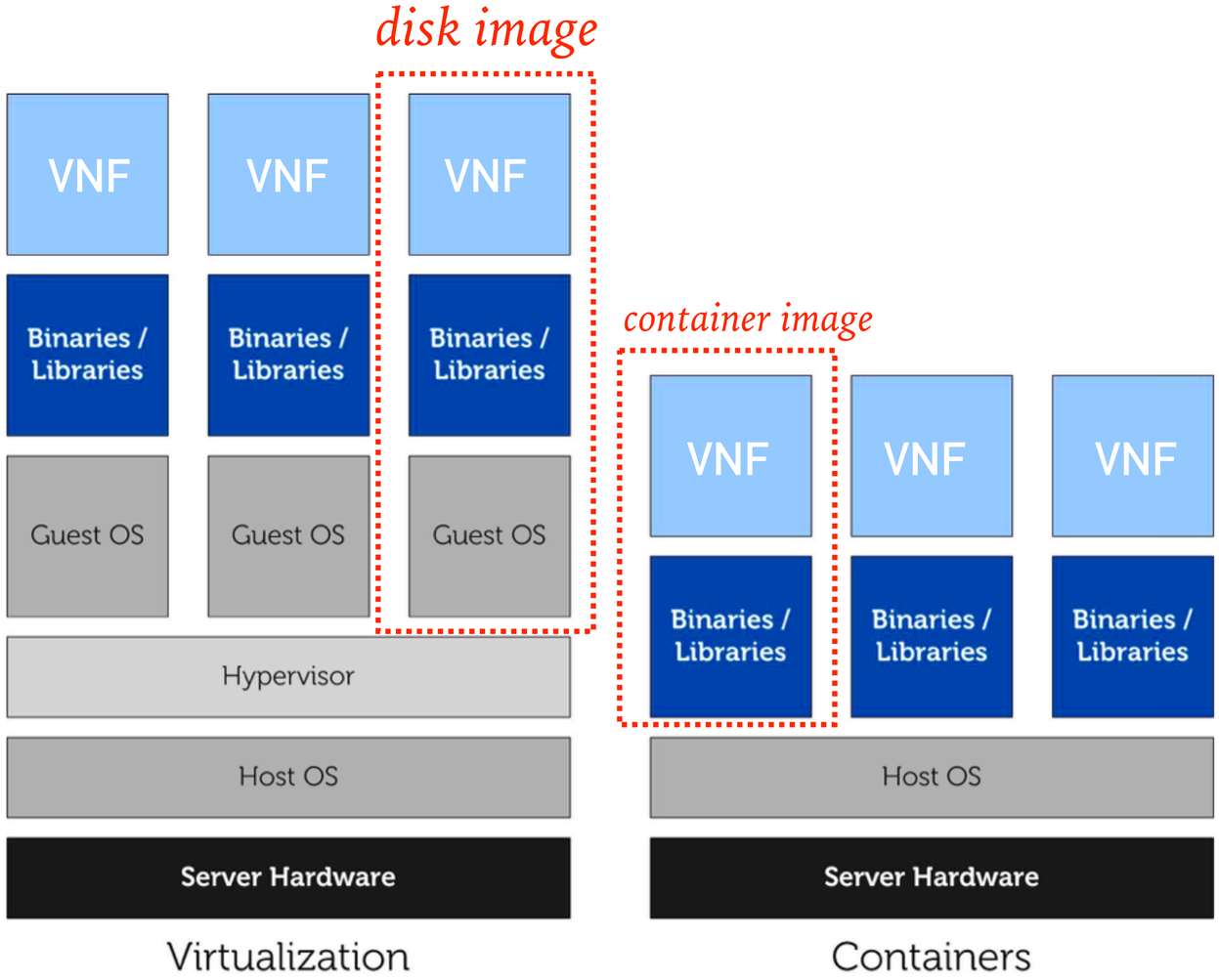


VNF (software)

devs openstack summit users

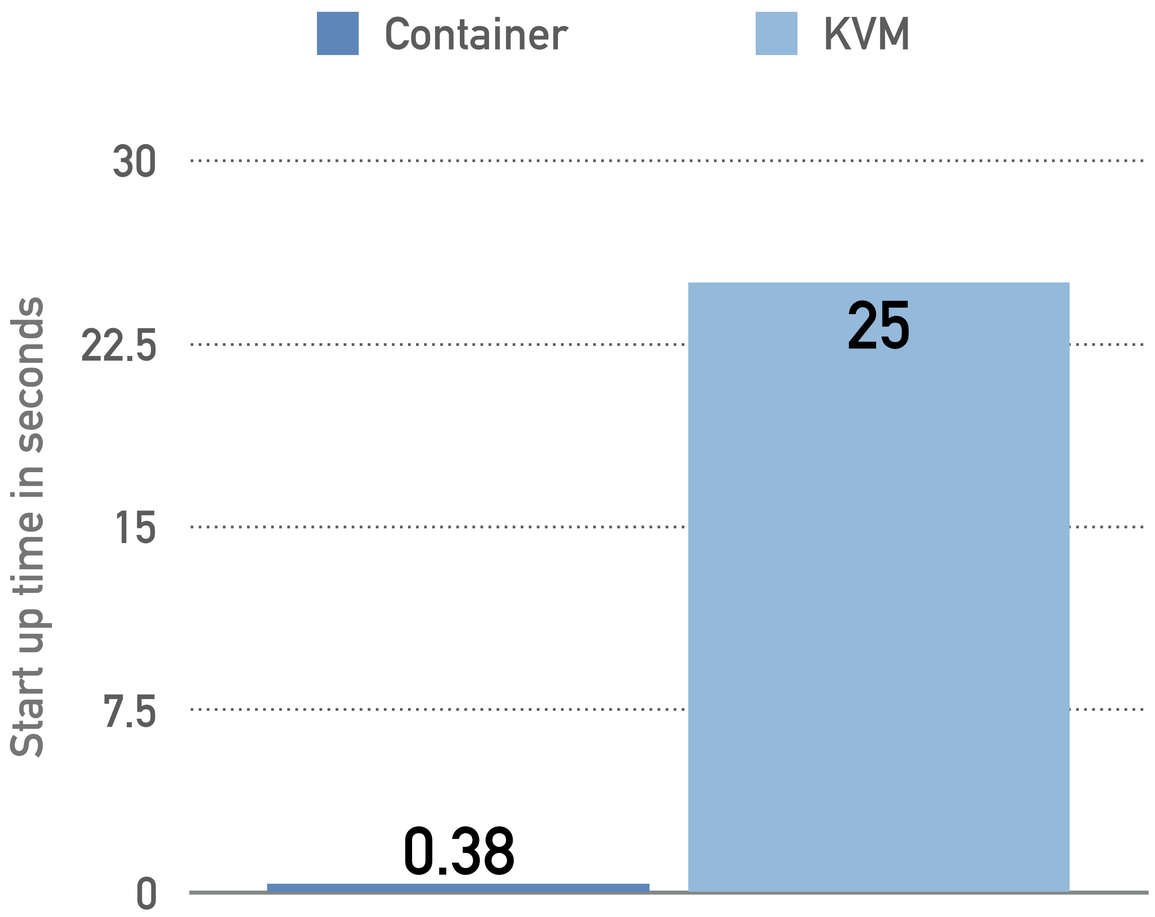
SHIP VNF TO CLOUD

Physical Equipments -> VNFs -> Cloud



VNF ❤ cloud

- Wait, what kind of cloud?
- Q: VM, or container?
- A: 6 dimensions analysis
  - Service agility
  - Network performance
  - Resource footprint & density
  - Portability & Resilience
  - Configurability
  - Security & Isolation



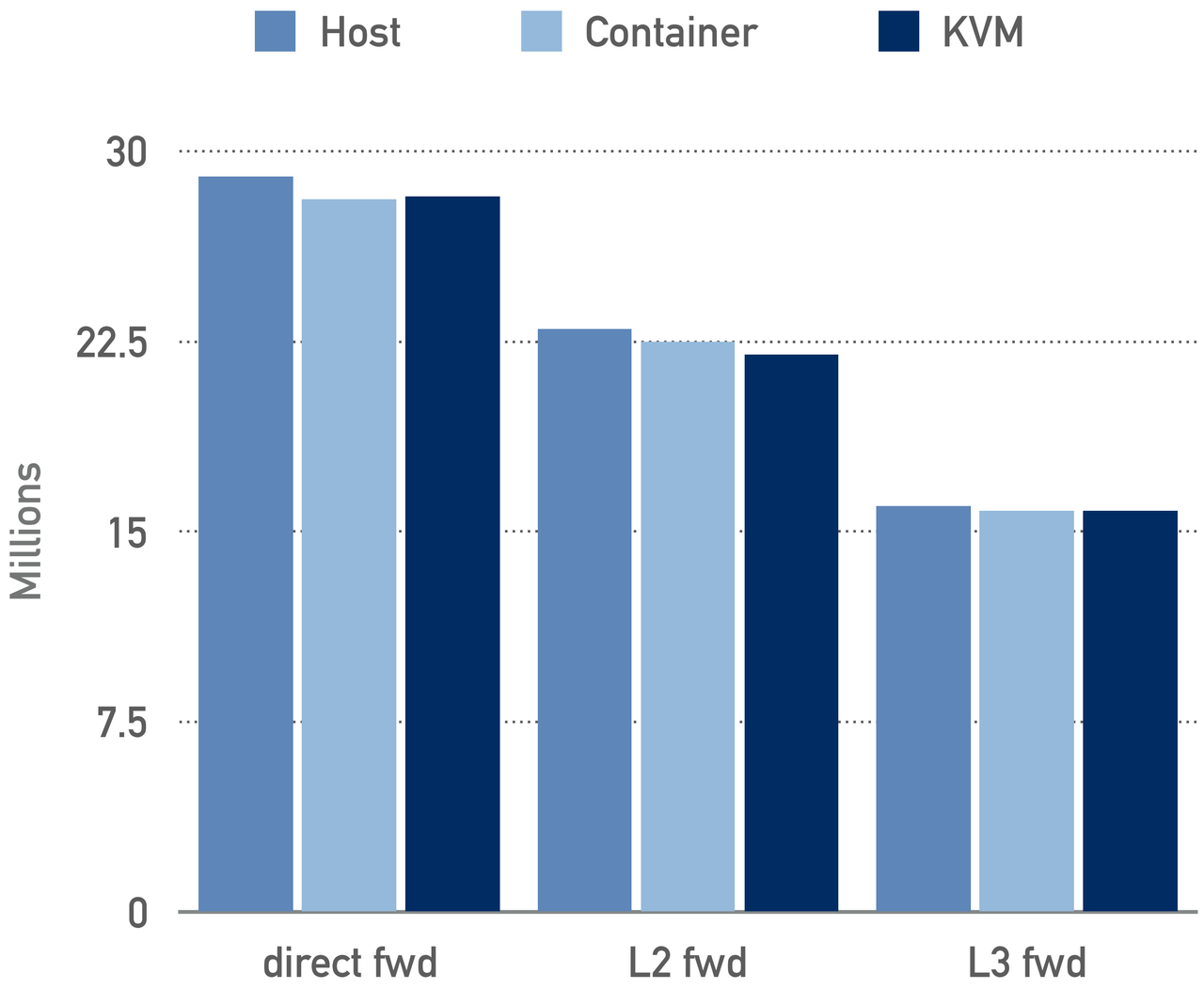
Average Startup Time (Seconds) Over Five Measurements

Data source: Intel white paper

SERVICE AGILITY

- Provision VM
  - hypervisor configuration
  - guest OS spin-up
  - align guest OS with VNFs
    - process mgmt service, startup scripts etc
- Provision container
  - start process in right namespaces and cgroups
  - no other overhead

NETWORK PERFORMANCE



- Throughput
  - "the resulting packets/sec that the VNF is able to push through the system **is stable and similar** in all three runtimes"

Packets per Second That a VNF Can Process in Different Environments

Data source: Intel white paper

NETWORK PERFORMANCE

Latency Introduced by Moving Packets from One NIC onto Another

Environment	Average Latency (µs)	Maximum Latency (µs)
Host	11.4375	53
Container	11.3955	56
KVM	11.1735	457

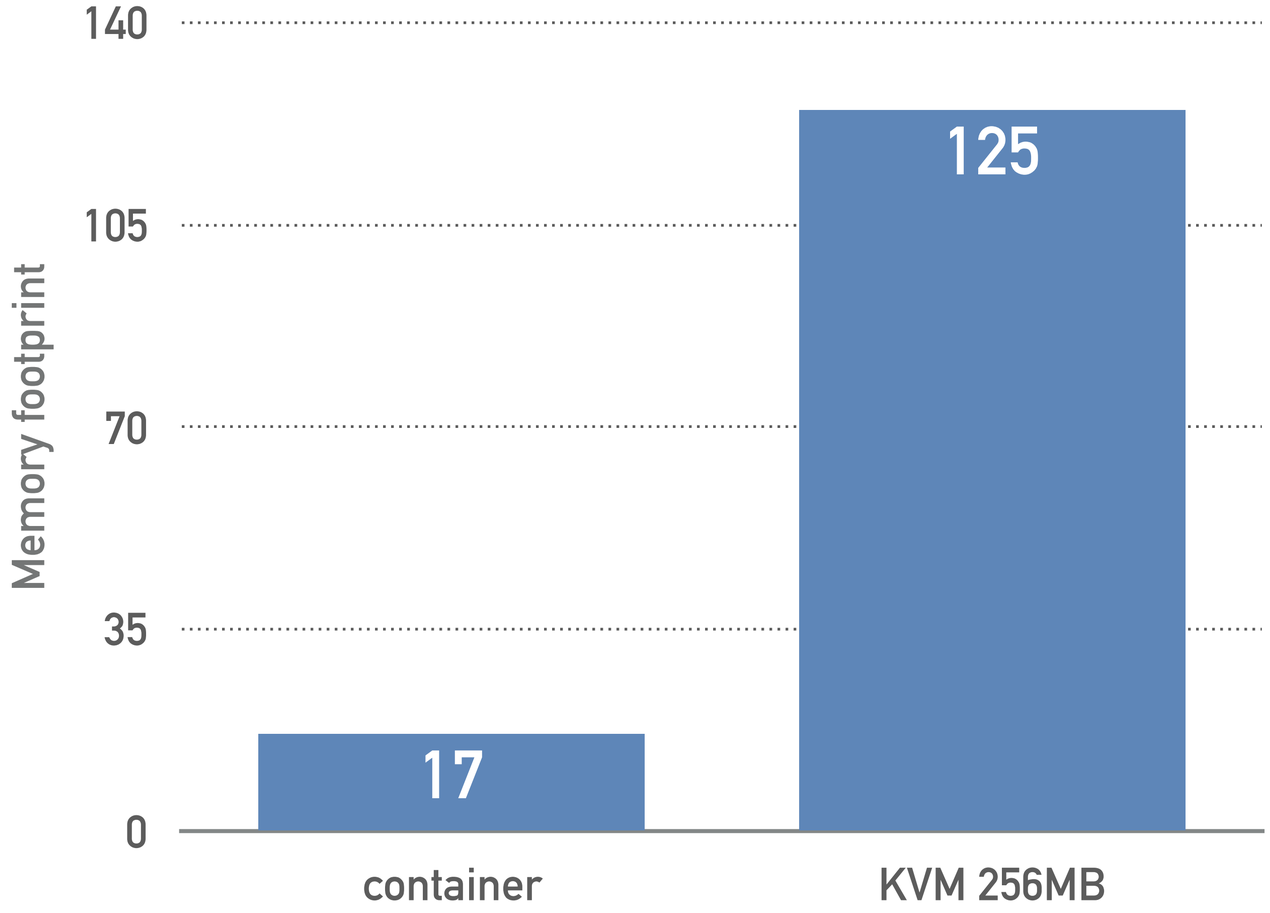
Latency Introduced by Layer 2 Forwarding

Environment	Average Latency (µs)	Maximum Latency (µs)
Host	493.148	565.441
Container	530.065	599.067
KVM	518.032	974.435

Data source: Intel white paper

- Latency
  - Direct forwarding
    - no big difference
    - VM show unstable
      - caused by hypervisor time to process regular interrupts
  - L2 forwarding
    - no big difference
    - container even shows extra latency
      - extra kernel code execution in cgroups
    - VM show unstable
      - cased by same reason above

RESOURCE FOOTPRINT & DENSITY



- VM
  - KVM 256MB(without —mem-prealloc) using about 125MB when booted
- Container
  - only 17MB
    - amount of code loaded into memory is significantly less
- Deployment density
  - is limited by incompressible resource
    - Memory & Disk, while container does not need disk provision

OS Flavor	Disk Size	Container Image Size
Ubuntu 14.04	> 619MB	> 188.3MB
CentOS 7	> 680MB	> 229.6MB
Alpine	—	> 5 MB
Busybox	—	>2MB

Data source: Intel white paper

PORTABILITY & RESILIENCE

- VM disk image
  - a provisioned disk with full operating system
  - the final disk image size is often counted by GB
  - extra processes for porting VM
    - hypervisor re-configuration
    - process mgmt service
- Container image
  - share host kernel = smaller image size
  - can even be: “app binary size + 2~5MB” for deploy
    - docker multi-stage build (NEW FEATURE)

CONFIGURABILITY

- VM
  - no obvious method to pass configuration to application
  - alternative methods:
    - share folder, port mapping, ENV …
      - **no** easy or user friendly tool to help us
- Container
  - user friendly container control tool (dockerd etc)
    - volume
    - ENV
    - …

SECURITY & ISOLATION

- VM
  - hardware level virtualization
  - independent guest kernel
- Container
  - weak isolation level
  - share kernel of host machine
  - reinforcement
    - Capabilities
    - libseccomp
    - SELinux/APPArmor
    - while non of them can be easily applied
      - e.g. what CAP is needed/unneeded for a specific container?

No cloud provider allow user to run containers without wrapping them inside full blown VM!

“

Cloud Native vs Security?

Hyper

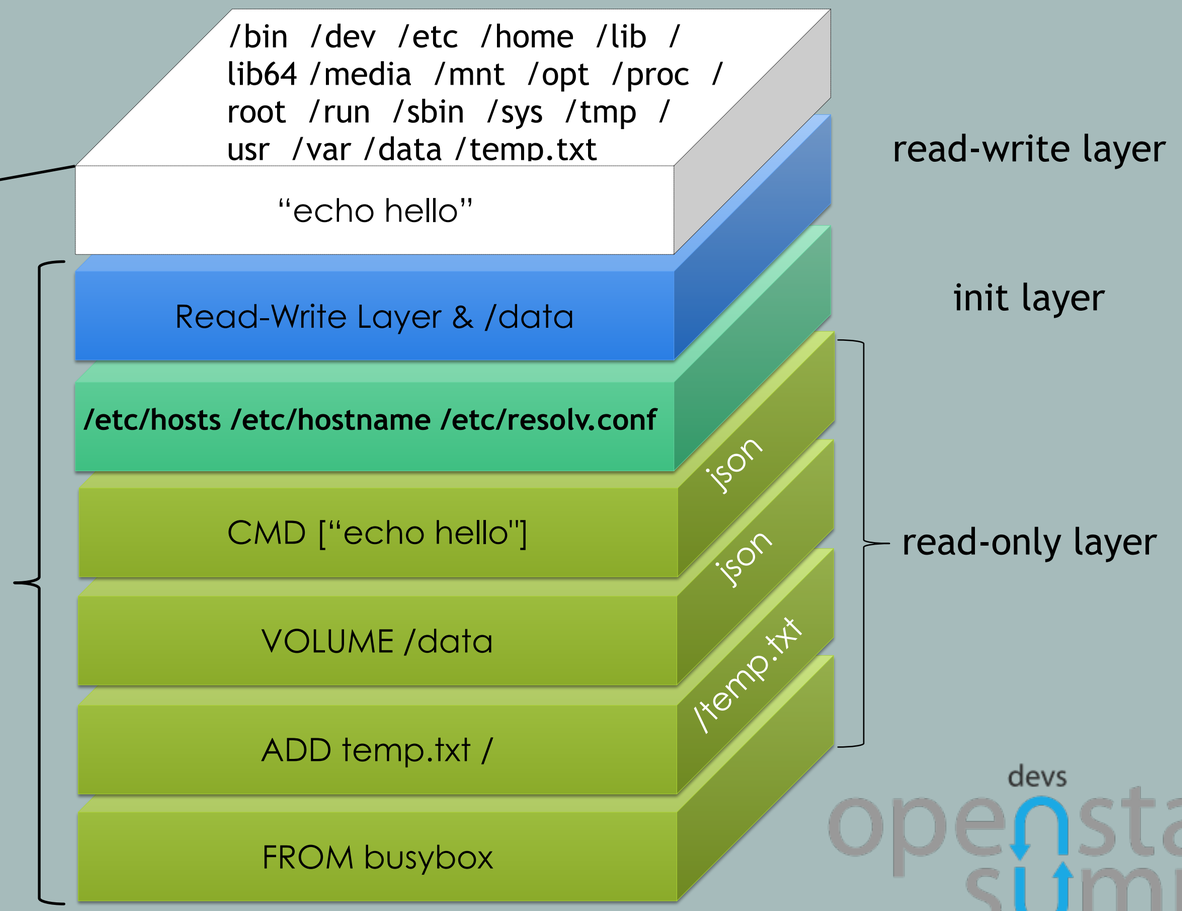
Let's make life easier

HYPERCONTAINER

- Secure, while keep Cloud Native
  - ~~Make container more like VM~~
  - Make VM more like container

REVISIT CONTAINER

```
FROM busybox
ADD temp.txt /
VOLUME /data
CMD ["echo hello"]
```



- **Container Runtime**
  - The dynamic view and boundary of your running process
- **Container Image**
  - The static view of your program, data, dependencies, files and directories

e.g. Docker Container

HYPERCONTAINER

- Container runtime: hypervisor
  - **RunV**
    - https://github.com/hyperhq/runv
    - The OCI compatible hypervisor based runtime implementation
  - **Control daemon**
    - hyperd: https://github.com/hyperhq/hyperd
  - **Init service (PID=1)**
    - hyperstart: https://github.com/hyperhq/hyperstart/
- Container image:
  - Docker image
  - OCI Image Spec

```
[root@localhost ~]# hyperctl pull ubuntu:latest
...
[root@localhost ~]# hyperctl run -t ubuntu
root@ubuntu-2994825143:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@ubuntu-2994825143:/# exit
exit
[root@localhost ~]# hyperctl run -d ubuntu
POD id is pod-aEafFYramp
Time to run a POD is 143 ms
[root@localhost ~]# hyperctl list
POD ID              POD Name            VM name             Status
pod-CMLStRNiKG      ubuntu-2994825143                       succeeded
pod-aEafFYramp      ubuntu-3972307775   vm-TeLKCtGBcF       running
[root@localhost ~]# virsh list
 Id    Name                           State
----------------------------------------------------
 104   vm-TeLKCtGBcF                  running
[root@localhost ~]#
```

Want to see a demo?

STRENGTHS

- Service agility
  - startup time: **sub-second** (e.g. 500~ms)
- Network performance
  - same with VM & container
- Resource footprint
  - **small (e.g. 30MB)**
- Portability & Resilience
  - **use Docker image (i.e. MB)**
- Configurability
  - same as Docker
- Security & Isolation
  - **hardware virtualization & independent kernel**

DEMO

- hyperctl run -d ubuntu:trusty sleep 1000
  - small memory footprint
- hyperctl exec -t $POD /bin/bash
- fork bomb
  - **Do not test this in Docker** (without ulimit set)
    - unless you want to lose your host machine :)

WHERE TO RUN YOUR VNF?

	Container	VM	HyperContainer
Kernel features	No	Yes	Yes
Startup time	380ms	25s	500ms
Portable Image	Small	Large	Small
Memory footprint	Small	Large	Small
Configurability of app	Flexible	Complex	Flexible
Network Performance	Good	Good	Good
Backward Compatibility	No	Yes	Yes (bring your own kernel)
Security/Isolation	Weak	Strong	Strong

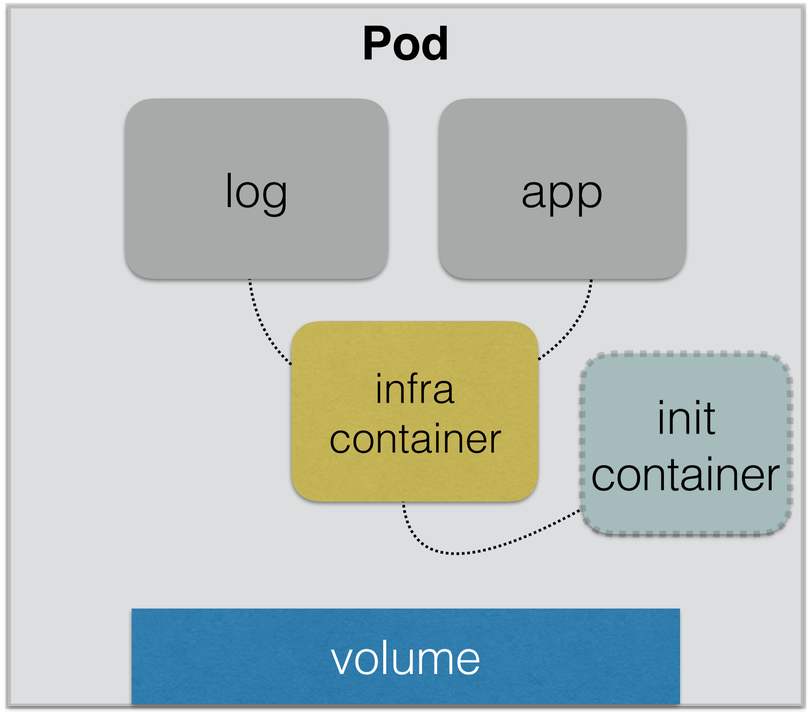
HYPERNETES

the cloud platform for NFV

HYPERNETES

- Hypernetes, also known as h8s is:
  - Kubernetes + HyperContainer
    - HyperContainer is now an official container runtime in k8s 1.6
      - integration is achieved thru kubernetes/frakti project
  - + OpenStack
    - Multi-tenant network and persistent volumes
      - standalone Keystone + Neutron + Cinder

1. CONTAINER RUNTIME

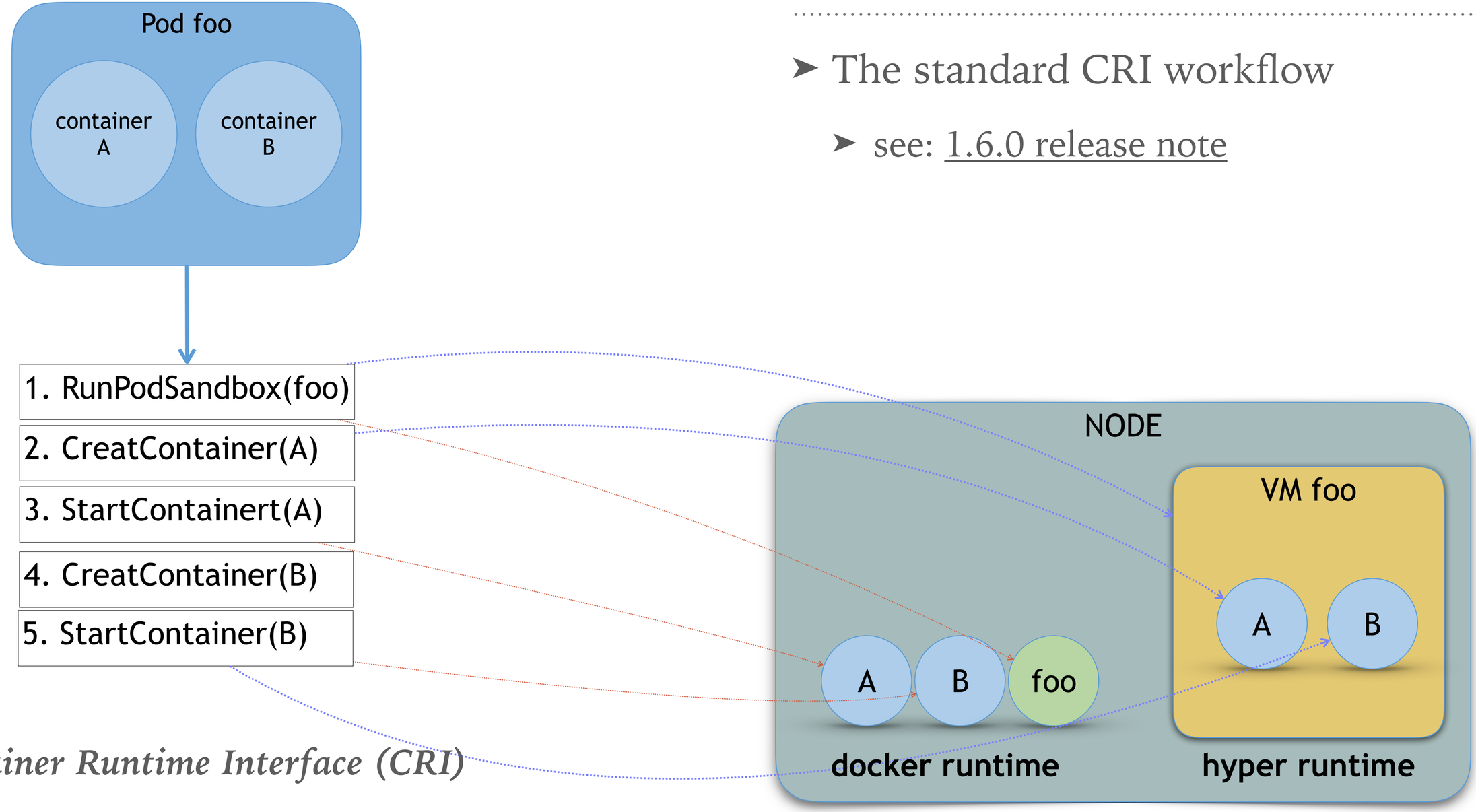


POD

- Why?
  - Fix some bad practices:
    - use supervised manage multi-apps in one container
    - try to ensure container order by hacky scripts
    - try to copy files from one container to another
    - try to connect to peer container across whole network stack
- So Pod is
  - The group of super-affinity containers
  - The atomic scheduling unit
  - The “process group” in container cloud
  - **Also how HyperContainer match to Kubernetes philosophy**

HYPERCONTAINER IN KUBERNETES

- The standard CRI workflow
  - see: 1.6.0 release note



Container Runtime Interface (CRI)

2. MULTI-TENANT NETWORK

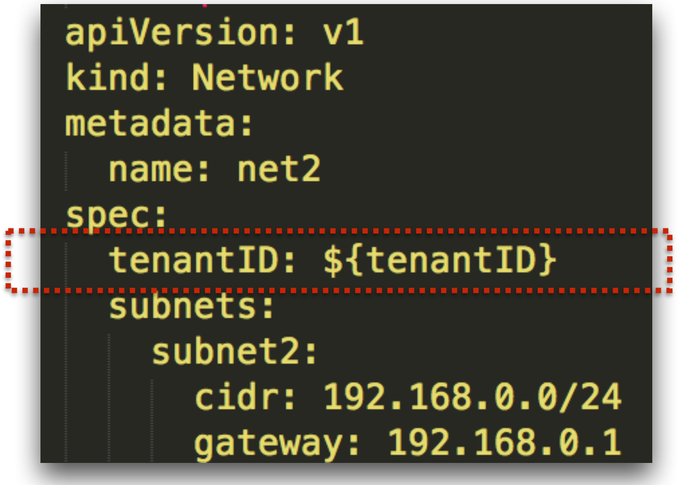
MULTI-TENANT NETWORK

- Goal:
  - leveraging tenant-aware Neutron network for Kubernetes
  - following the k8s network plugin workflow
- Non-goal:
  - break k8s network model

KUBERNETES NETWORK MODEL

- **Pod reach Pod**
  - all Pods can communicate with all other Pods without NAT
- **Node reach Pod**
  - all nodes can communicate with all Pods (and vice-versa) without NAT
- **IP addressing**
  - Pod in cluster can be addressed by its IP

DEFINE NETWORK



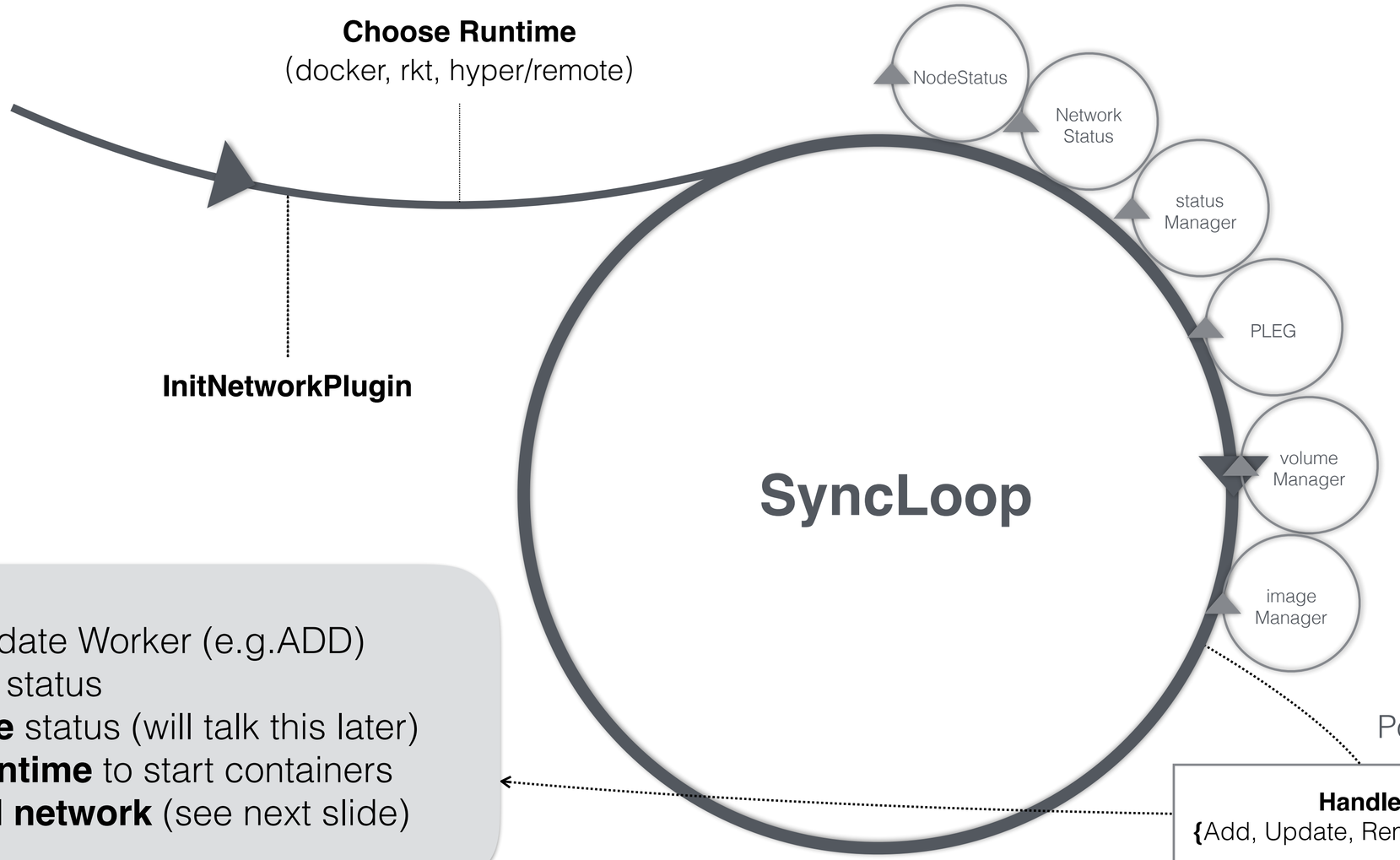
- Network
  - a top level API object
  - Network: Namespace = 1: N
  - each tenant (created by Keystone) has its own Network
- Network Controller is responsible for lifecycle of Network object
  - a control loop to create/delete **Neutron "net"** based on API object change

ASSIGN POD TO NETWORK

- Pods belonging to the same Network can reach each other directly through IP
  - a Pod's network mapping to Neutron "port"
  - kubelet is responsible for Pod network setup
    - let's see how kubelet works

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns2
spec:
  network: net2
```

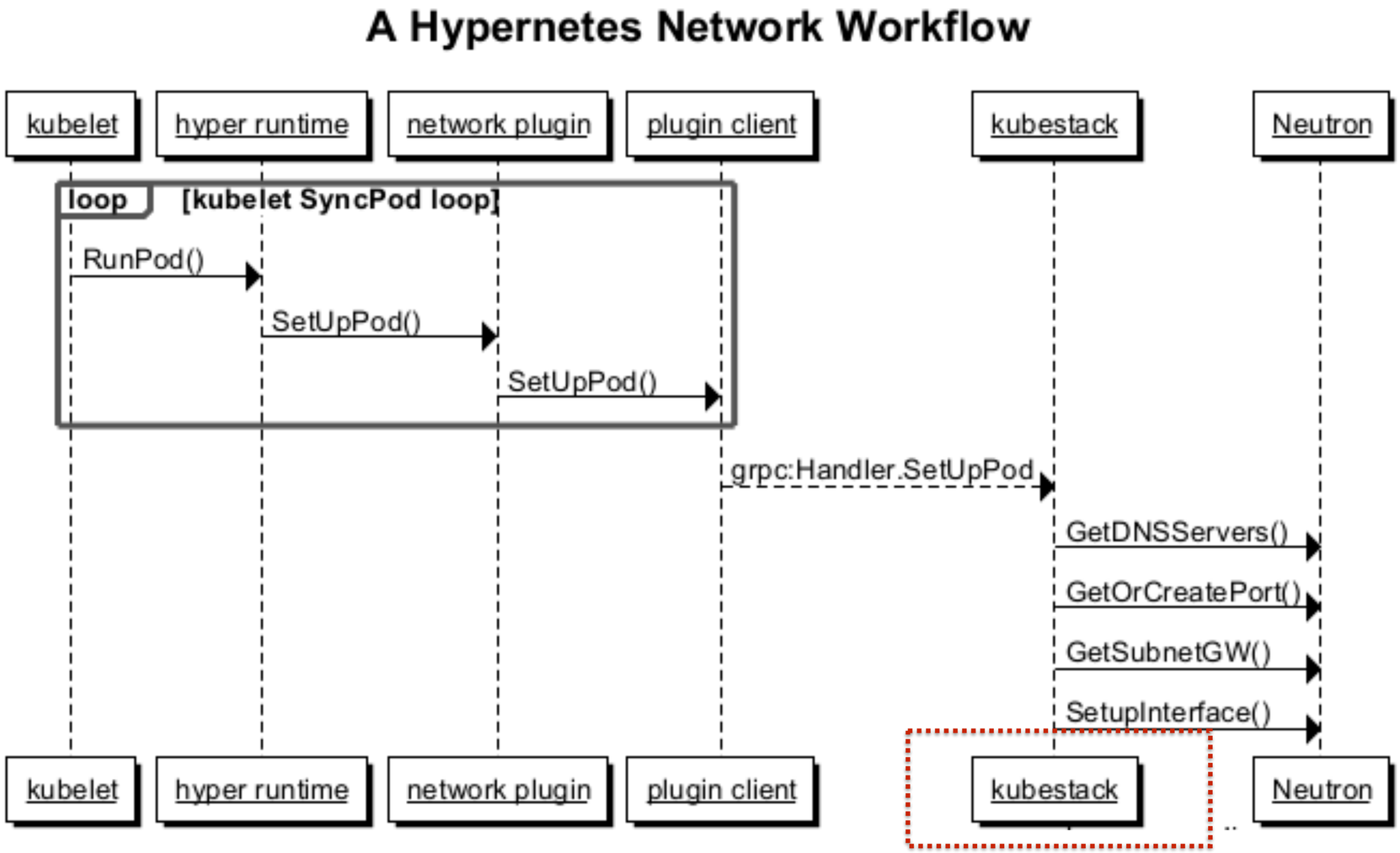
DESIGN OF KUBELET



Pod Update Worker (e.g.ADD)

- generale Pod status
- check **volume** status (will talk this later)
- **use hyper runtime** to start containers
  - **set up Pod network** (see next slide)

SET UP POD NETWORK



KUBESTACK

A standalone gRPC daemon

1. to “translate” the *SetUpPod* request to the Neutron network API
2. handling multi-tenant Service proxy

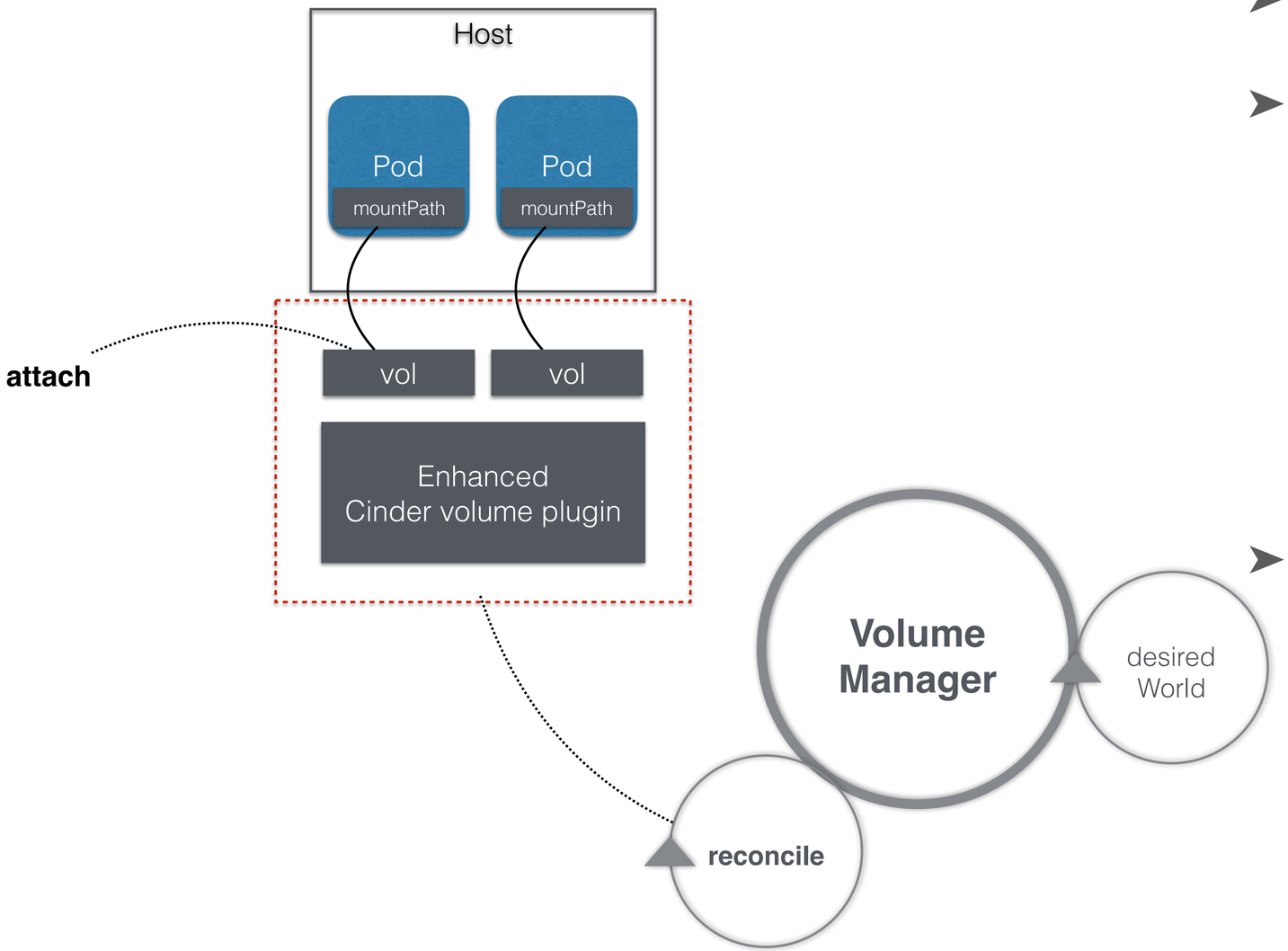
MULTI-TENANT SERVICE

- Default iptables-based kube-proxy is not tenant aware
  - Pods and Nodes are isolated into different networks
- Hypernetes uses a build-in **ipvs** as the Service LB
  - handle all Services in same namespace
  - follow *OnServiceUpdate* and *OnEndpointsUpdate* workflow
- ExternalProvider
  - a OpenStack LB will be created as Service
  - e.g. curl 58.215.33.98:8078

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: ns2
spec:
  type: NetworkProvider
  externalIPs:
  - 58.215.33.98
  ports:
  - port: 8078
    name: http
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
```

3. PERSISTENT VOLUME

PERSISTENT VOLUME IN HYPERNETES



- Enhanced Cinder volume plugin
- **Linux container:**
  1. query Nova to find node
  2. attach Cinder volume to host path
  3. bind mount host path to Pod containers
- **HyperContainer:**
  - **directly attach** block devices to Pod
  - no extra time to query Nova
  - no need to install full OpenStack

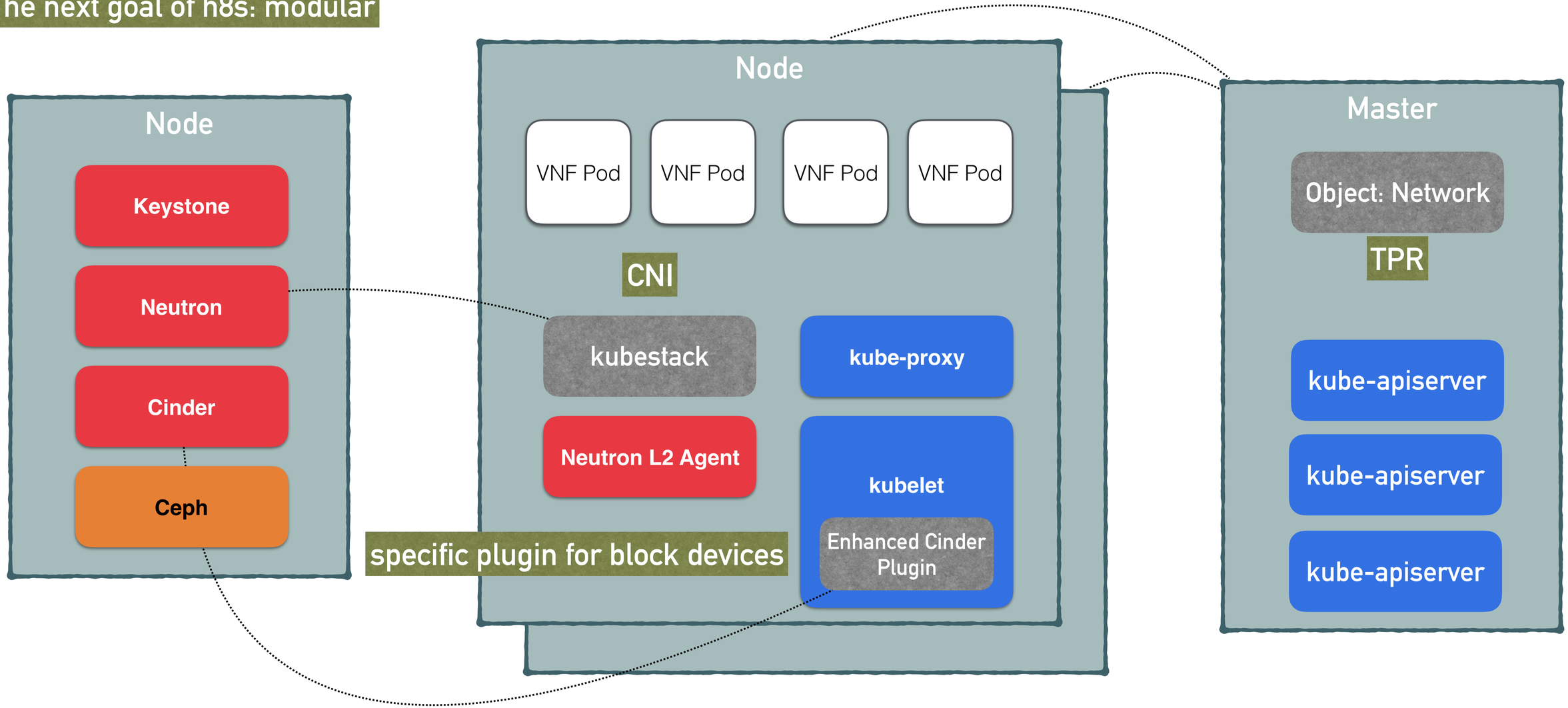
PV EXAMPLE

```
  cinder create --name volume 1
  volId=$(cinder show volume | awk '/ id /{print $4}')
  cat | kubectl create -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: web
  namespace: ns2
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: nginx-persistent-storage
      mountPath: /var/lib/nginx
  volumes:
  - name: nginx-persistent-storage
    cinder:
      volumeID: ${volId}
      fsType: ext4
EOF
```

- Create a Cinder volume
- Claim volume by reference its *volumeID*

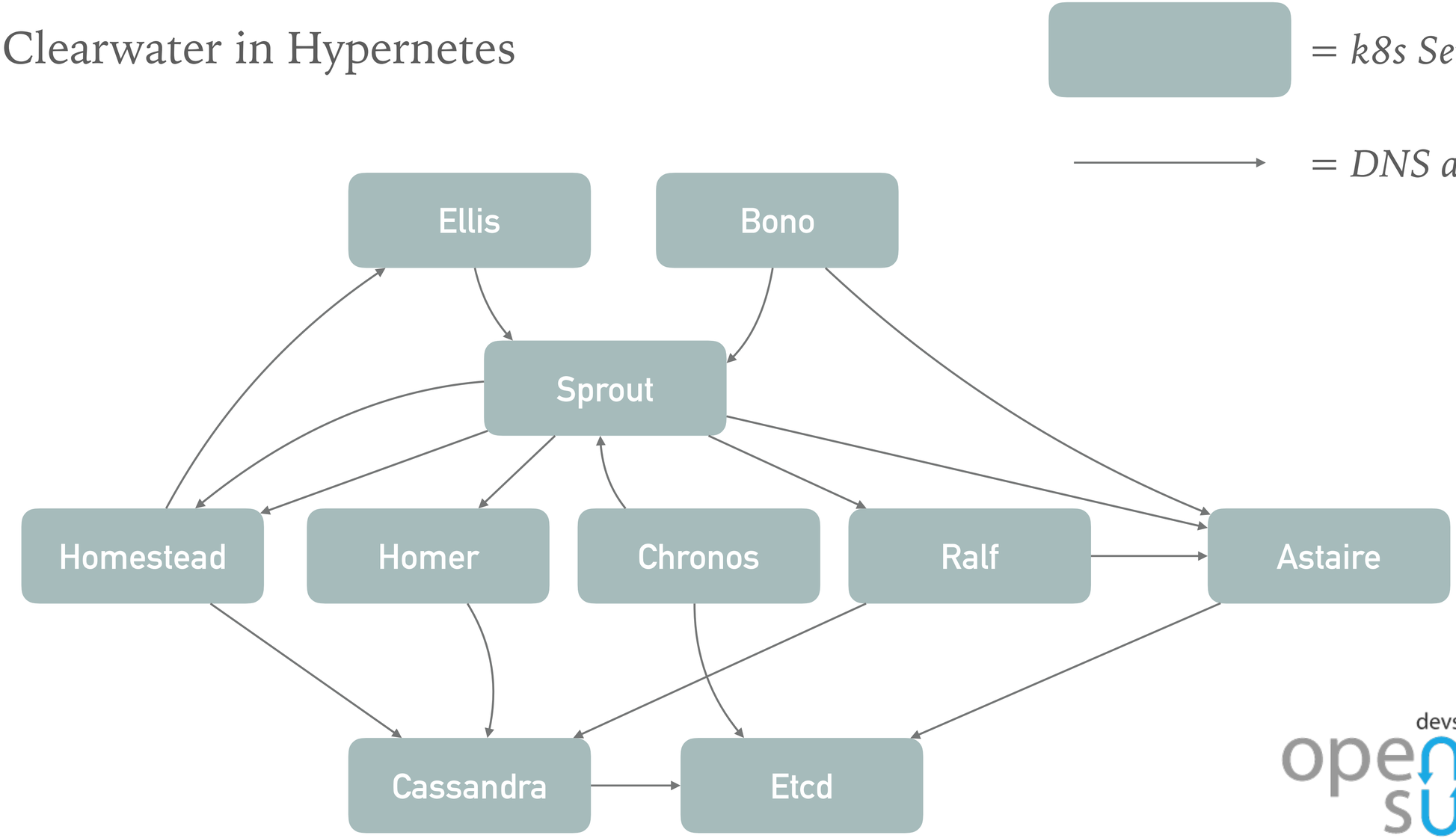
HYPERNETES TOPOLOGY

The next goal of h8s: modular

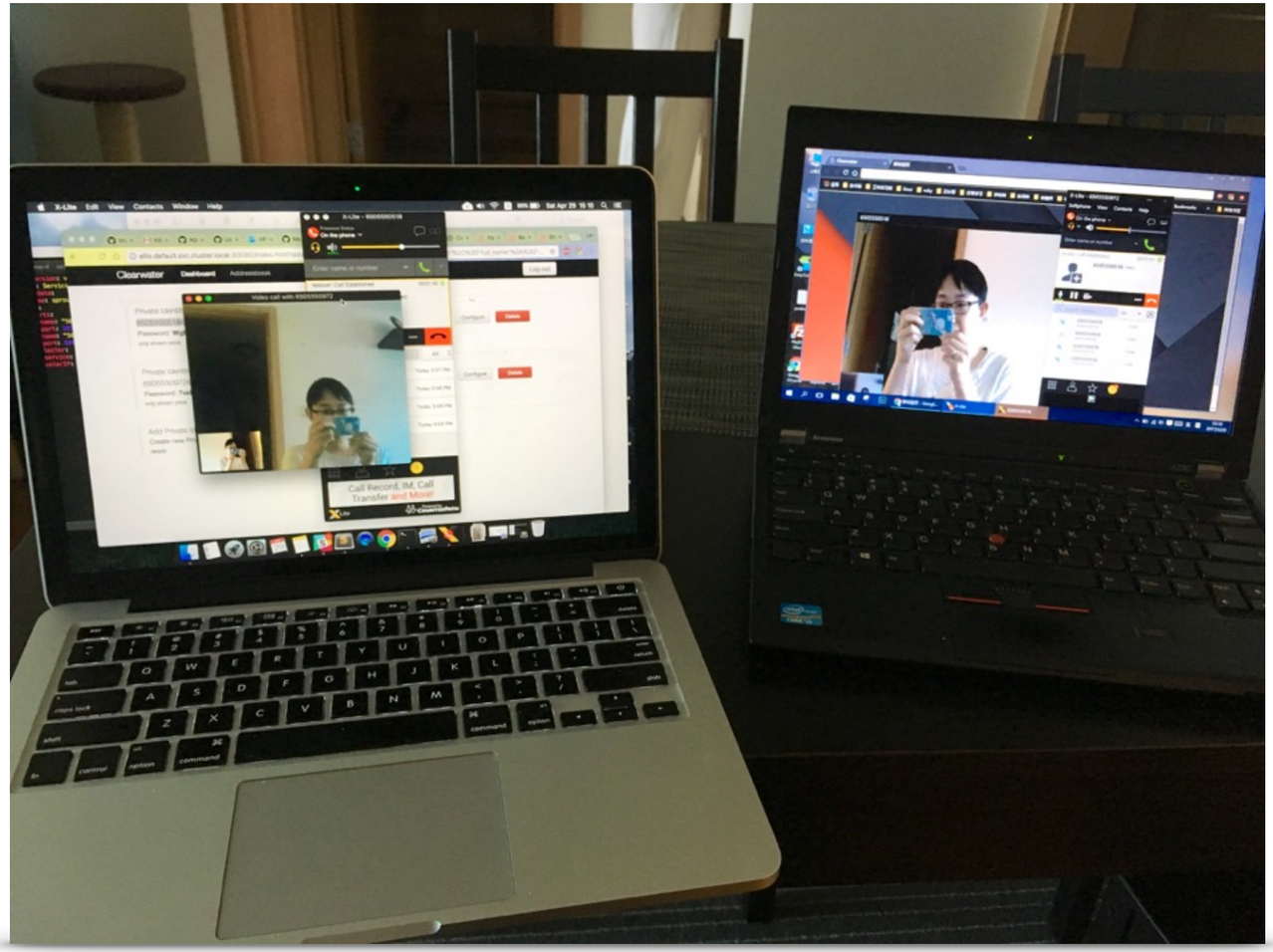


BACK TO THE REAL-WORLD DEMO

- Run Clearwater in Hypernetes



DEMO



- **One command** to deploy all
- All scripts and yamls can be found here:
  - https://github.com/hyperhq/hypernetes
  - https://github.com/Metaswitch/clearwater-docker

```
$ kubectl create -f clearwater-docker/kubernetes/
```

LESSONS LEARNED

- Do not use supervisord to manage processes
  - use Pod + initContainer
- Do not abuse DNS name
  - e.g. **scscf.sprout** is not a valid DNS name, see PR#441
- Liveness & Readiness check are useful

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: homestead
spec:
  ...
    spec:
      initContainers:
      - image: "clearwater-infra:latest"
        name: "clearwater-infra"
        ...
      - image: "clearwater-snmp:latest"
        name: "clearwater-snmp"
        ...
      containers:
      - image: "resouer/homestead:latest"
        name: homestead
        ...
      - image: "resouer/homestead-prov:latest"
        name: homestead-prov
        ...
      - image: "nginx:latest"
        name: nginx
        ...
```

```
root@h
clearw
clearw
clearw
clearw
snmpd
```

```yaml
livenessProbe:
  exec:
    command: ["/bin/bash", "/usr/share/kubernetes/liveness.sh", "8888 8889"]
  initialDelaySeconds: 60
readinessProbe:
  exec:
    command: ["/bin/bash", "/usr/share/kubernetes/liveness.sh", "8888 8889"]
```

```
00:02
:11
:11
```

THE END

*NEWS: **Stackube,** a new OpenStack project originated from h8s*