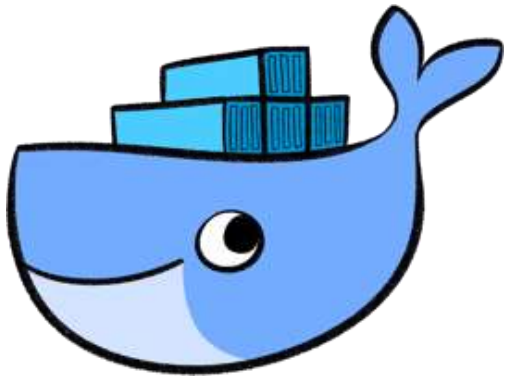


Docker / Kubernetes / Istio

Containers | Container Orchestration | Service Mesh

Araf Karsh Hamid : Co-Founder/CTO, MetaMagic Global Inc., NJ, USA

Agenda

1 Docker

- 12 Factor App Methodology
- Docker Concepts
- Images and Containers
- Anatomy of a Dockerfile
- Networking / Volume

2 Kubernetes

- Kubernetes Concepts
- Namespace / Pods / RelicaSet /
- Deployment / Service / Ingress
- Rollout and Undo / Autoscale

3 Kubernetes Networking

- Docker / Kubernetes Networking
- Pod to Pod Networking
- Pod to Service Networking
- Ingress and Egress – Internet
- Network Policies

4 Kubernetes Advanced Concepts

- Quotas / Limits / QoS
- Pod / Node Affinity
- Pod Disruption Budget
- Persistent Volume / Claims
- Secrets / Jobs / Cron
- Kubernetes Commands

5 Istio

- Istio Concepts
- Gateway / Virtual Service
- Destination Rule / Service Entry
- AB Testing using Canary
- Beta Testing using Canary
- Logging and Monitoring

6 Best Practices

- Docker Best Practices
- Kubernetes Best Practices

12 Factor App Methodology

	Factors	Description
1	Codebase	One Code base tracked in revision control
2	Dependencies	Explicitly declare dependencies
3	Configuration	Configuration driven Apps
4	Backing Services	Treat Backing services like DB, Cache as attached resources
5	Build, Release, Run	Separate Build and Run Stages
6	Process	Execute App as One or more Stateless Process
7	Port Binding	Export Services with Specific Port Binding
8	Concurrency	Scale out via the process Model
9	Disposability	Maximize robustness with fast startup and graceful exit
10	Dev / Prod Parity	Keep Development, Staging and Production as similar as possible
11	Logs	Treat logs as Event Streams
12	Admin Process	Run Admin Tasks as one of Process

Source: https://12factor.net/

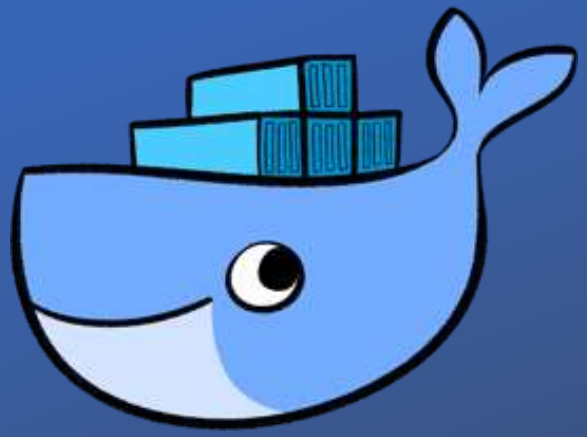
High Level Objectives

#19 Slide No's

From Creating a Docker Container to Deploying the Container in Production Kubernetes Cluster. All other activities revolves around these 8 points mentioned below.

1. Create Docker Images #19
2. Run Docker Containers for testing. #19
3. Push the Containers to registry #22
4. Docker image as part of your Code Pipeline Process.

1. Create Pods (Containers) with Deployments #40-46
2. Create Services #47
3. Create Traffic Rules (Ingress / Gateway / Virtual Service / Destination Rules) #49 #97-113
4. Create External Services



Docker Containers

Understanding Containers

Docker Images / Containers

Docker Networking

What's a Container?

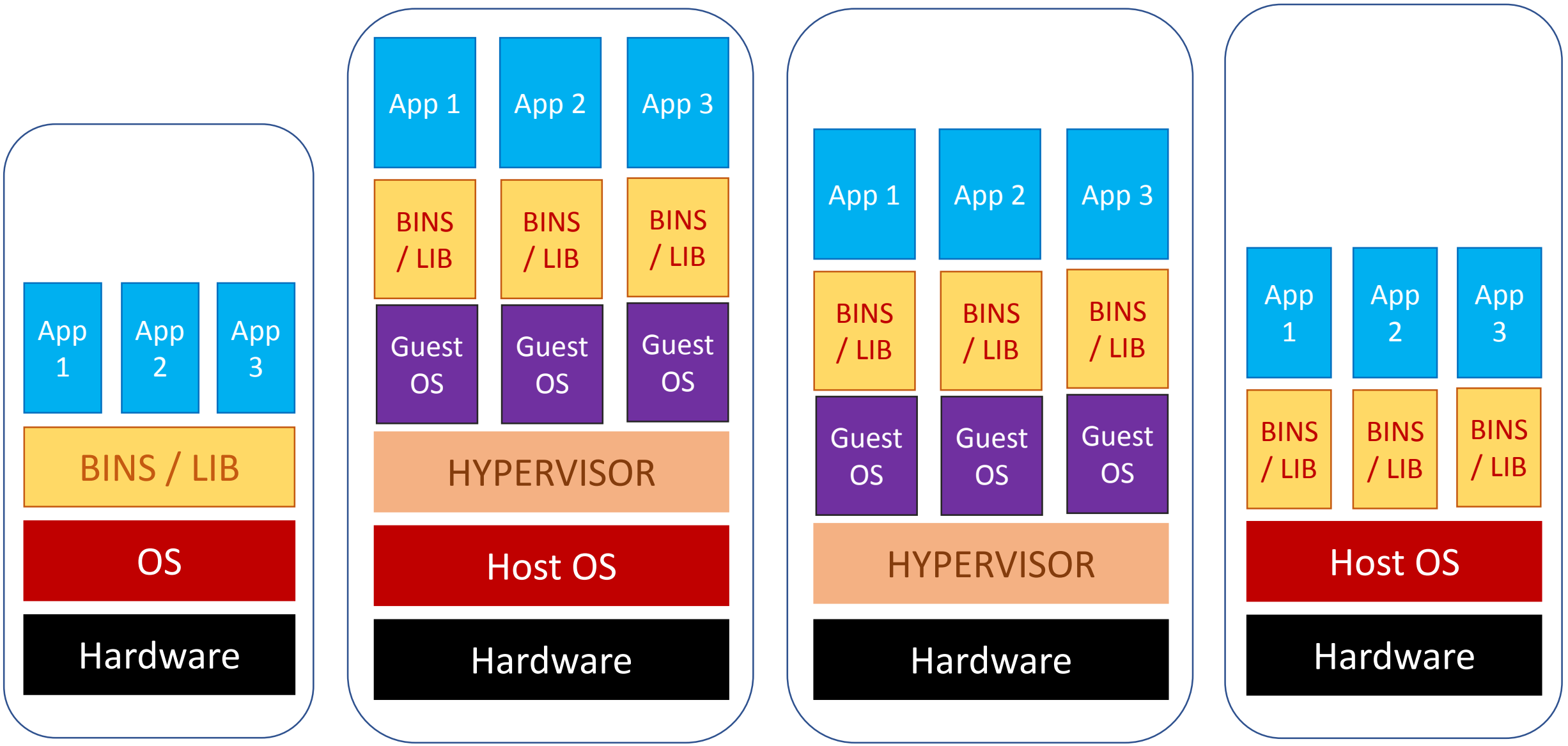
Looks like a Virtual

Walks like a

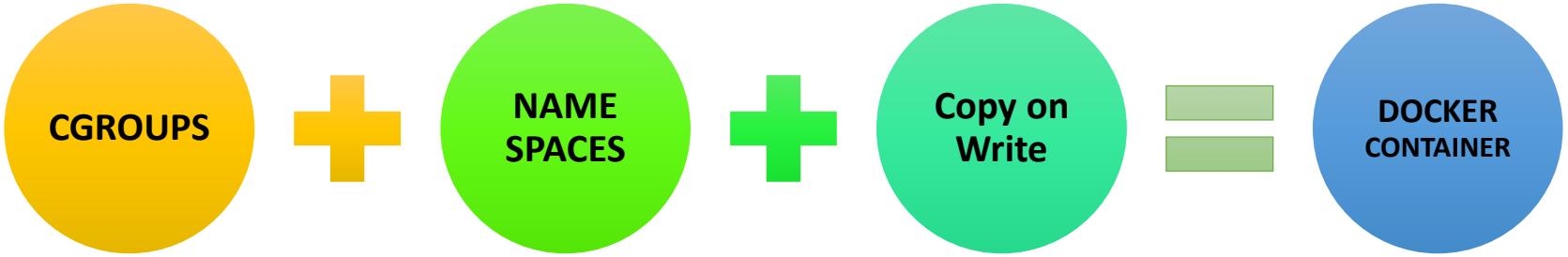
Runs like a Machine

Containers are a Sandbox inside Linux Kernel sharing the kernel with separate Network Stack, Process Stack, IPC Stack etc.

Servers / Virtual Machines / Containers



Docker containers are Linux Containers



- Kernel Feature
- Groups Processes
- Control Resource Allocation
  - CPU, CPU Sets
  - Memory
  - Disk
  - Block I/O

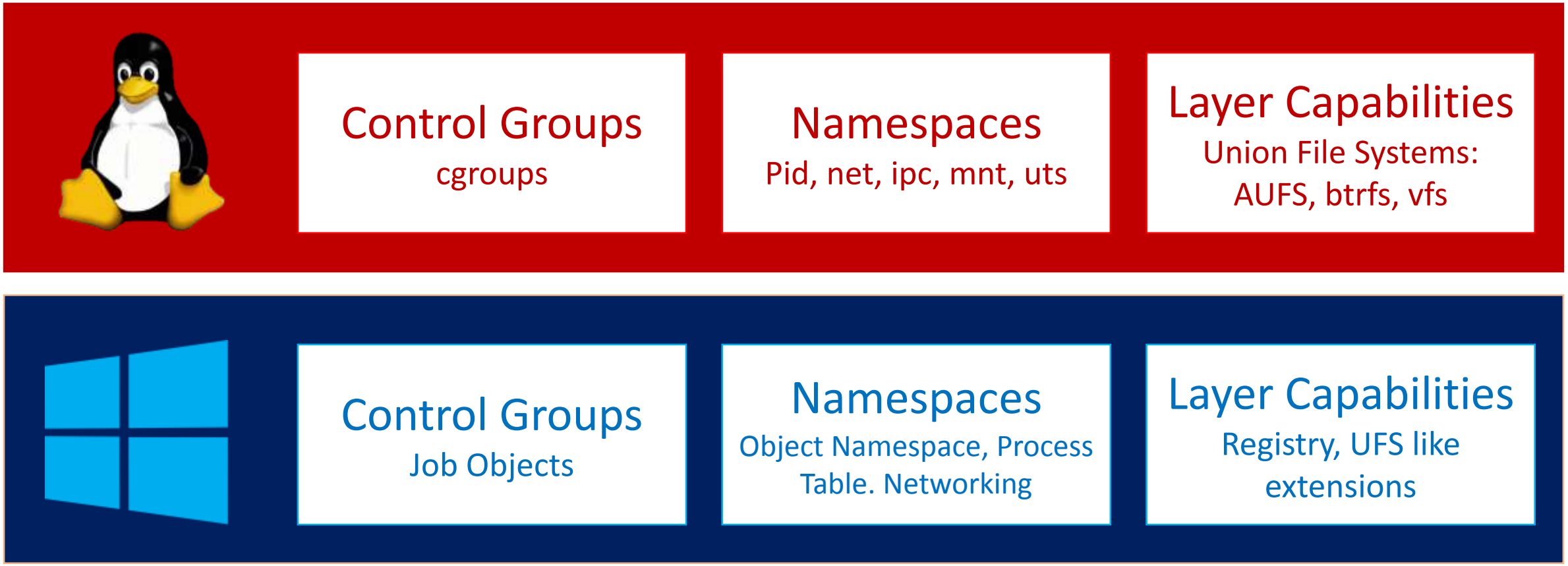
- The real magic behind containers
- It creates barriers between processes
- Different Namespaces
  - PID Namespace
  - Net Namespace
  - IPC Namespace
  - MNT Namespace
- Linux Kernel Namespace introduced between kernel 2.6.15 – 2.6.26

- Images
  - Not a File System
  - Not a VHD
  - Basically a tar file
  - Has a Hierarchy
    - Arbitrary Depth
  - Fits into Docker Registry

lxc-start = docker run

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01

Docker Container – Linux and Windows



Namespaces: Building blocks of the Containers

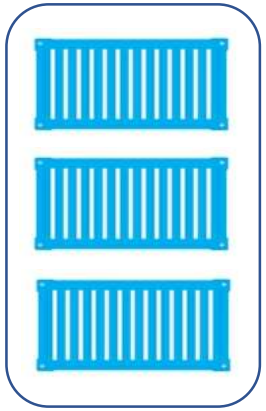
Docker Key Concepts

- Docker images
  - A Docker image is a read-only template.
  - For example, an image could contain an Ubuntu operating system with Apache and your web application installed.
  - Images are used to create Docker containers.
  - Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
  - Docker images are the build component of Docker.

Images

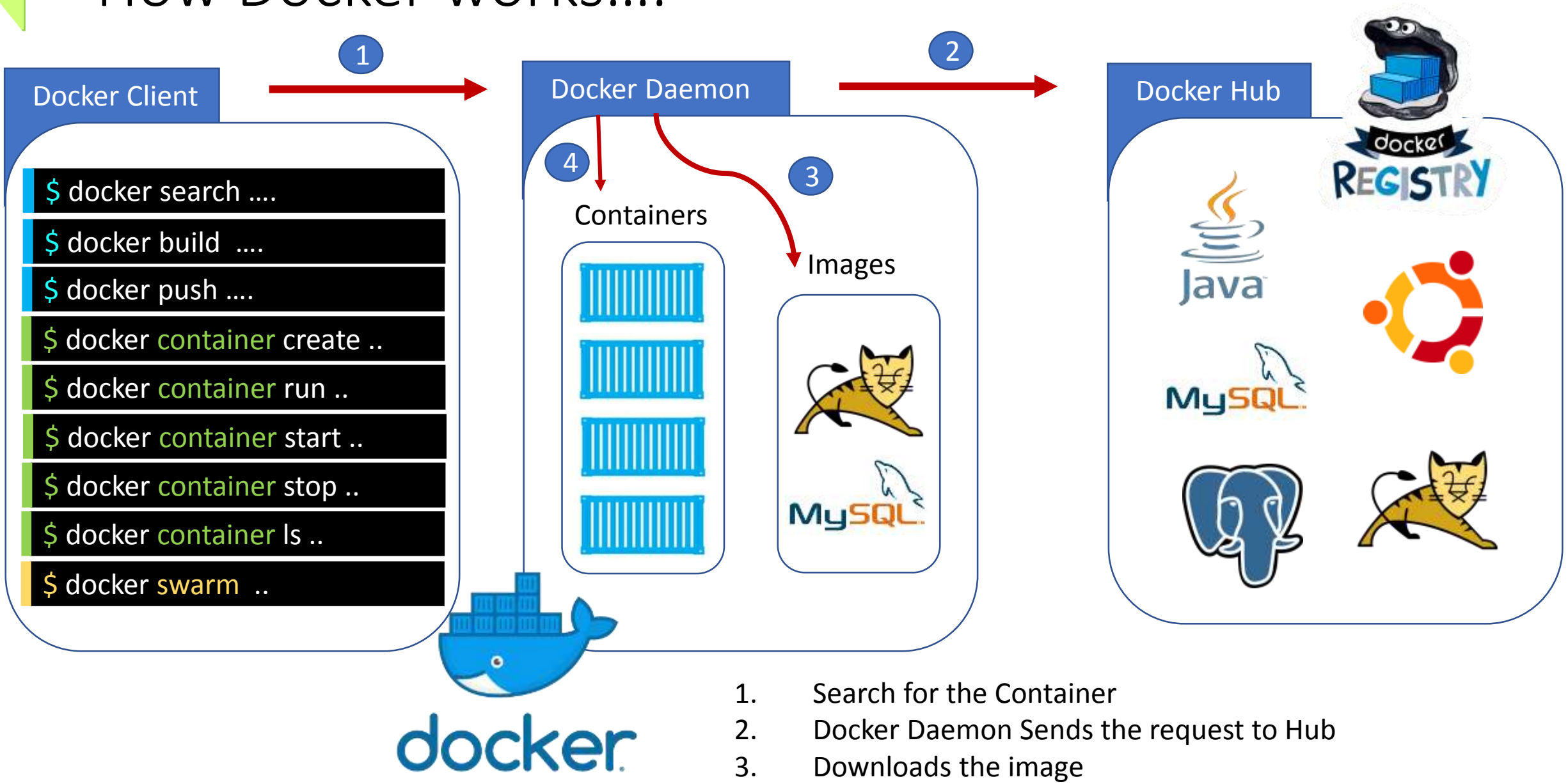
- Docker containers
  - Docker containers are similar to a directory.
  - A Docker container holds everything that is needed for an application to run.
  - Each container is created from a Docker image.
  - Docker containers can be run, started, stopped, moved, and deleted.
  - Each container is an isolated and secure application platform.
  - Docker containers are the run component of Docker.

Containers



- Docker Registries
  - Docker registries hold images.
  - These are public or private stores from which you upload or download images.
  - The public Docker registry is called Docker Hub.
  - It provides a huge collection of existing images for your use.
  - These can be images you create yourself or you can use images that others have previously created.
  - Docker registries are the distribution component of Docker.

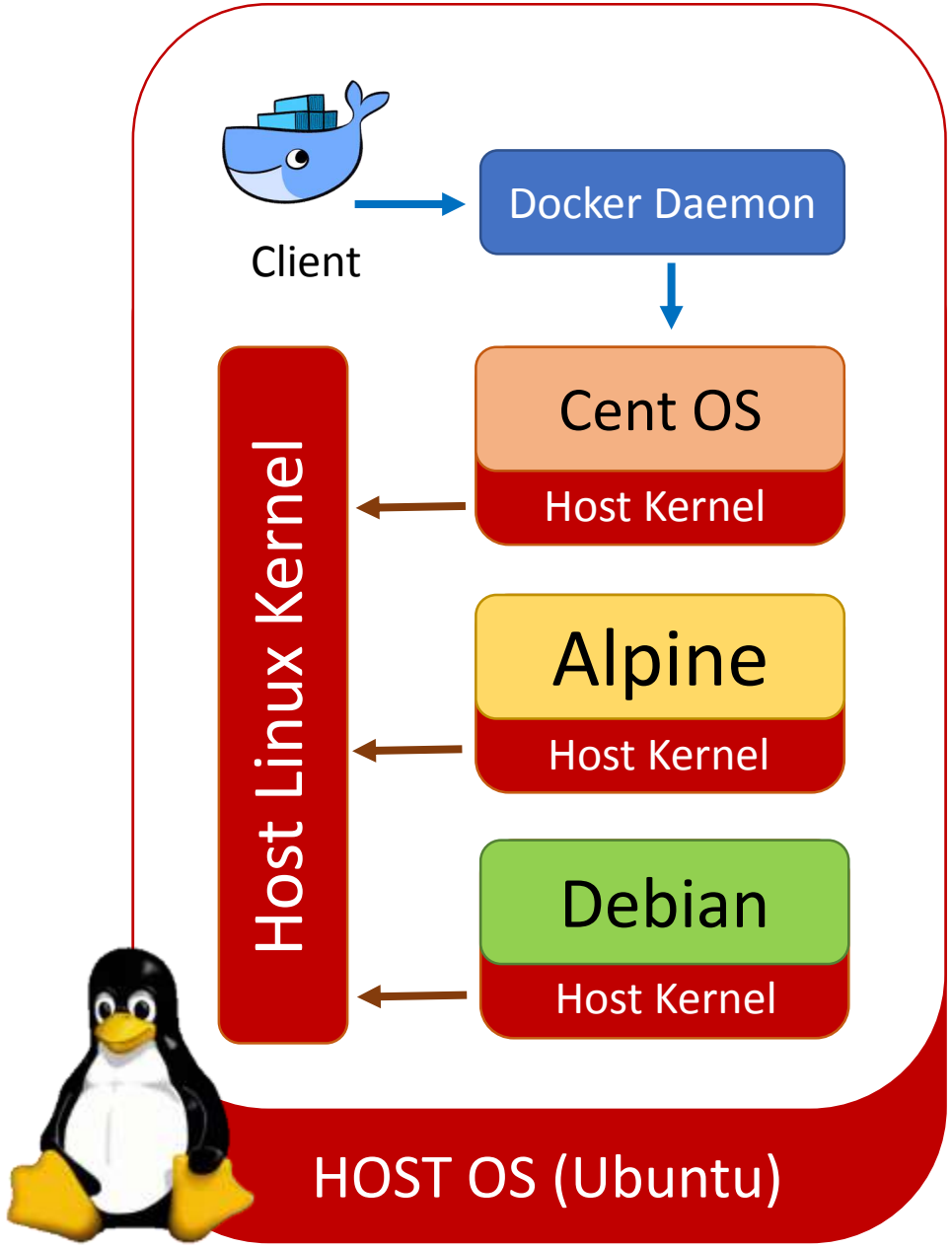
How Docker works….



1. Search for the Container
2. Docker Daemon Sends the request to Hub
3. Downloads the image
4. Run the Container from the image

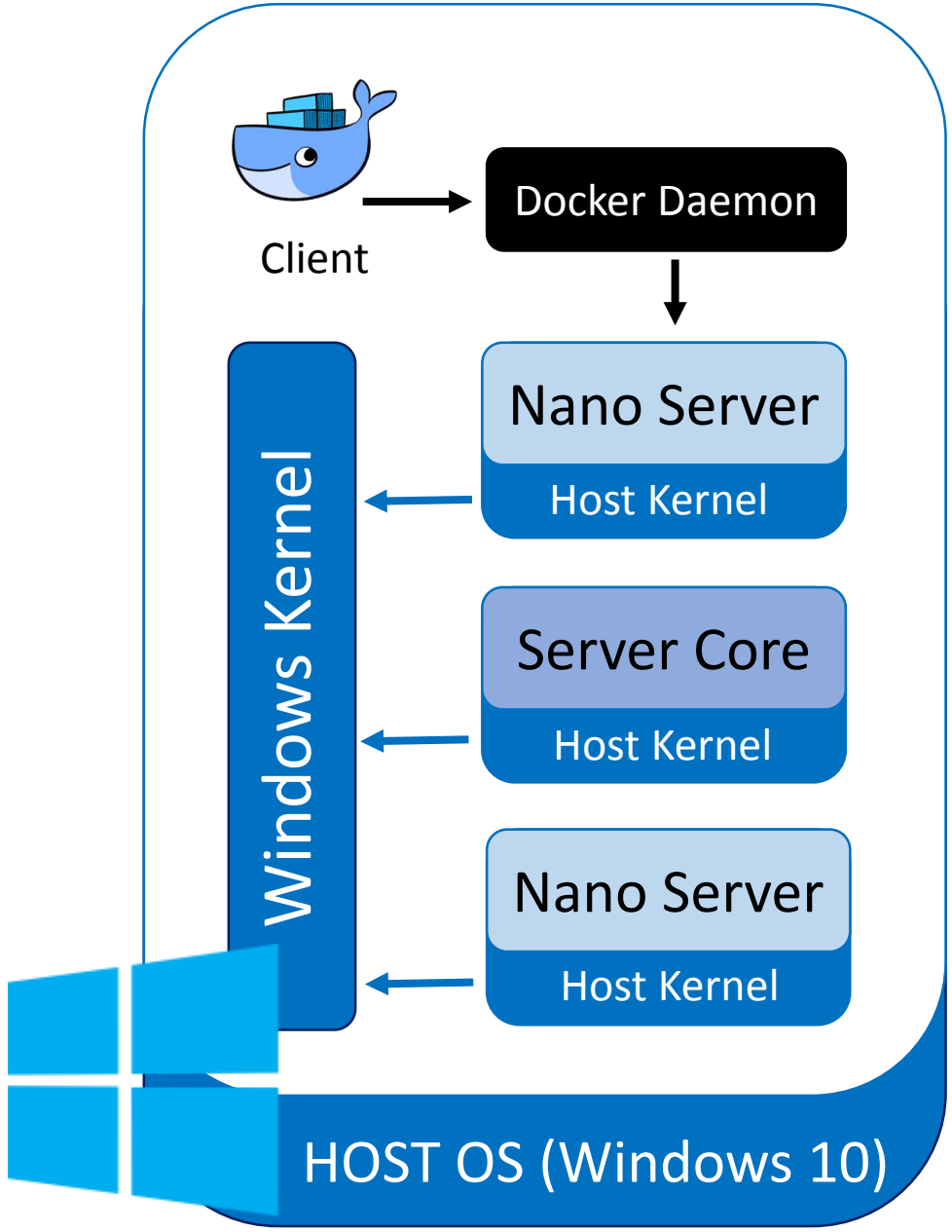
Linux Kernel

All the containers will have the same Host OS Kernel
If you require a specific Kernel version then Host Kernel needs to be updated

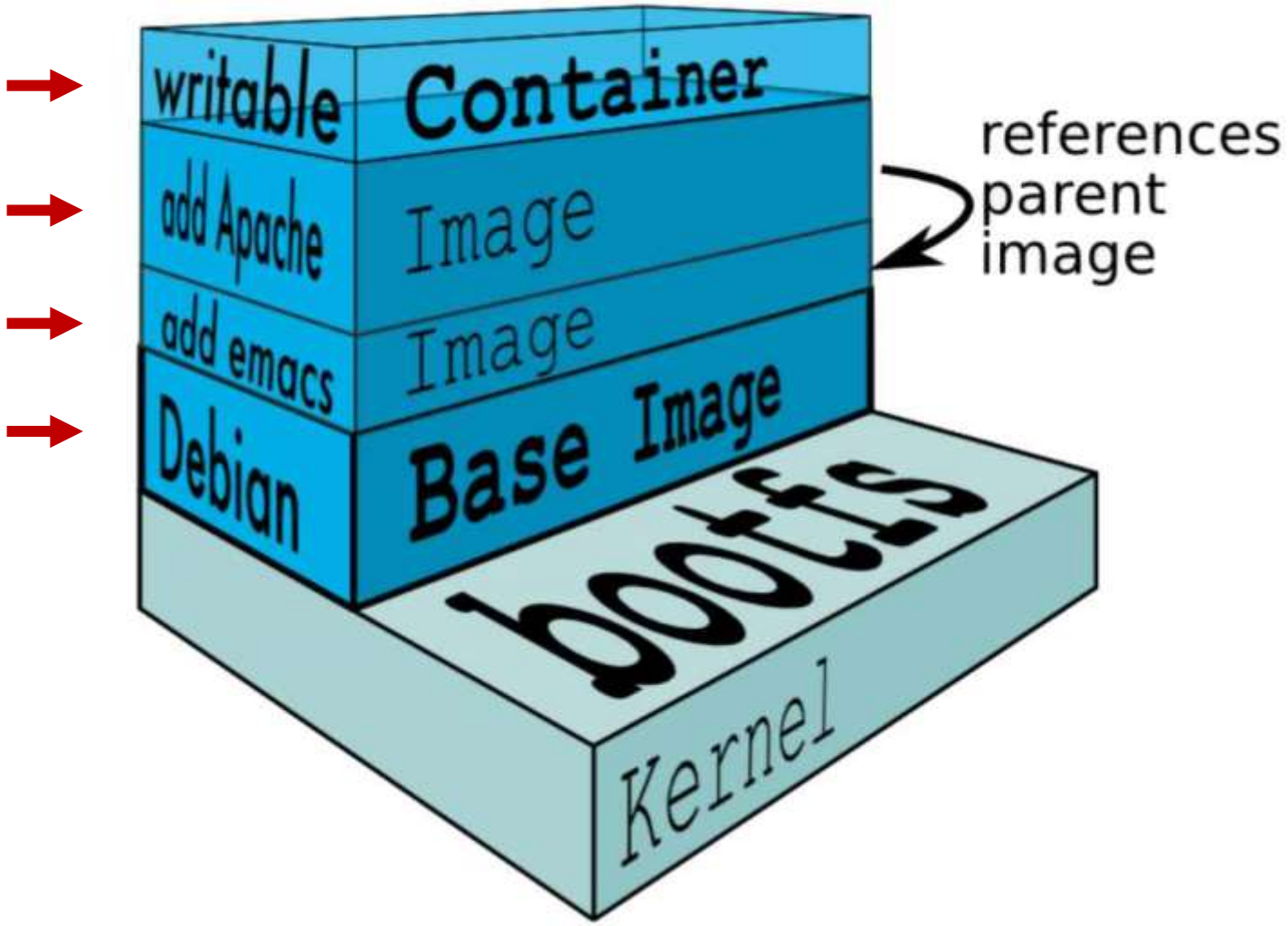


Windows Kernel

All the containers will have the same Host OS Kernel
If you require a specific Kernel version then Host Kernel needs to be updated



Docker Image structure



- Images are read-only.
- Multiple layers of image gives the final Container.
- Layers can be sharable.
- Layers are portable.

- Debian Base image
- Emacs
- Apache
- Writable Container

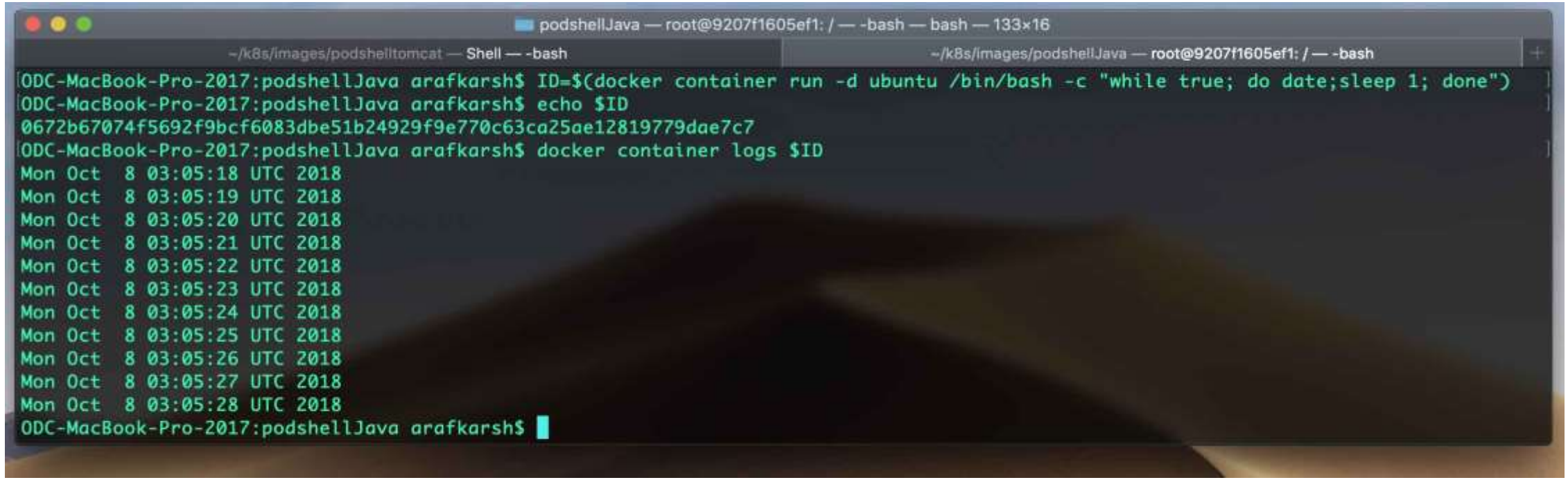
Running a Docker Container


```
$ docker pull ubuntu
```

Docker pulls the image from the Docker Registry

Creates a Docker Container of Ubuntu OS and runs the container and execute bash shell with a script.

```
$ ID=$(docker container run -d ubuntu –bin/bash -c “while true; do date; sleep 1; done”)
```




```
$ docker container logs $ID
```

Shows output from the(bash script) container

```
$ docker container ls
```

List the running Containers

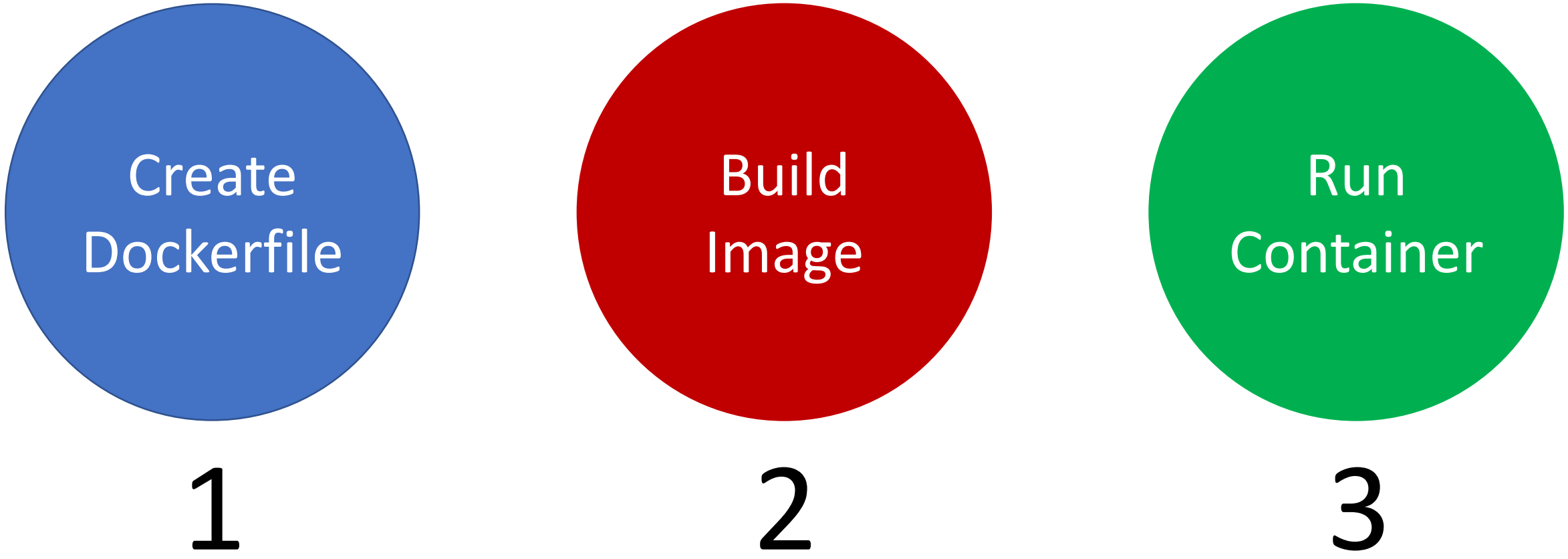
Anatomy of a Dockerfile

Command	Description	Example
FROM	The FROM instruction sets the Base Image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. The image can be any valid image – it is especially easy to start by pulling an image from the Public repositories	FROM ubuntu FROM alpine
MAINTAINER	The MAINTAINER instruction allows you to set the Author field of the generated images.	MAINTAINER johndoe
LABEL	The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and blackslashes as you would in command-line parsing.	LABEL version="1.0” LABEL vendor=“M2”
RUN	The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.	RUN apt-get install -y curl
ADD	The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the container at the path <dest>.	ADD hom* /mydir/ ADD hom?.txt /mydir/
COPY	The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.	COPY hom* /mydir/ COPY hom?.txt /mydir/
ENV	The ENV instruction sets the environment variable <key> to the value <value>. This value will be in the environment of all "descendent" Dockerfile commands and can be replaced inline in many as well.	ENV JAVA_HOME /JDK8 ENV JRE_HOME /JRE8

Anatomy of a Dockerfile

Command	Description	Example
VOLUME	The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log	VOLUME /data/webapps
USER	The USER instruction sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.	USER johndoe
WORKDIR	The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.	WORKDIR /home/user
CMD	There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect. The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.	CMD echo "This is a test." | wc -
EXPOSE	The EXPOSE instructions informs Docker that the container will listen on the specified network ports at runtime. Docker uses this information to interconnect containers using links and to determine which ports to expose to the host when using the –P flag with docker client.	EXPOSE 8080
ENTRYPOINT	An ENTRYPOINT allows you to configure a container that will run as an executable. Command line arguments to docker run <image> will be appended after all elements in an exec form ENTRYPOINT, and will override all elements specified using CMD. This allows arguments to be passed to the entry point, i.e., docker run <image> -d will pass the -d argument to the entry point. You can override the ENTRYPOINT instruction using the docker run --entrypoint flag.	ENTRYPOINT ["top", "-b"]

Build Docker Containers as easy as 1-2-3



1

Build a Docker Java image

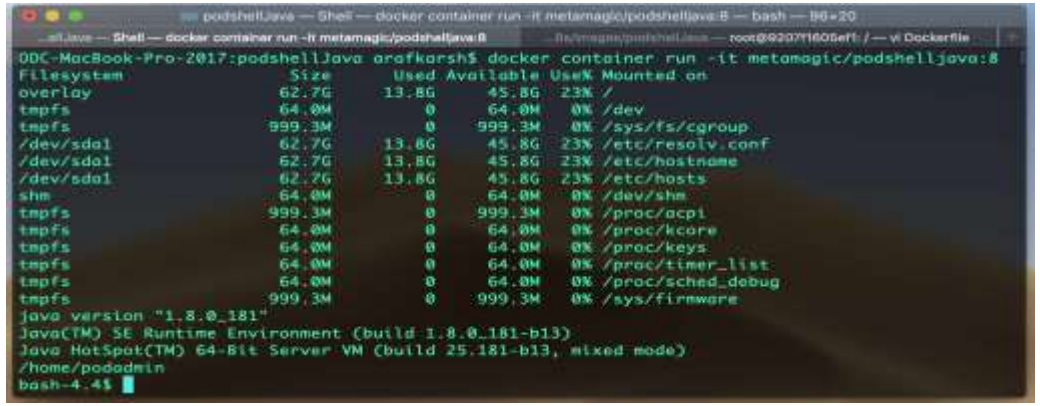
1. **Create your Dockerfile**
   - FROM
   - RUN
   - ADD
   - WORKDIR
   - USER
   - ENTRYPOINT

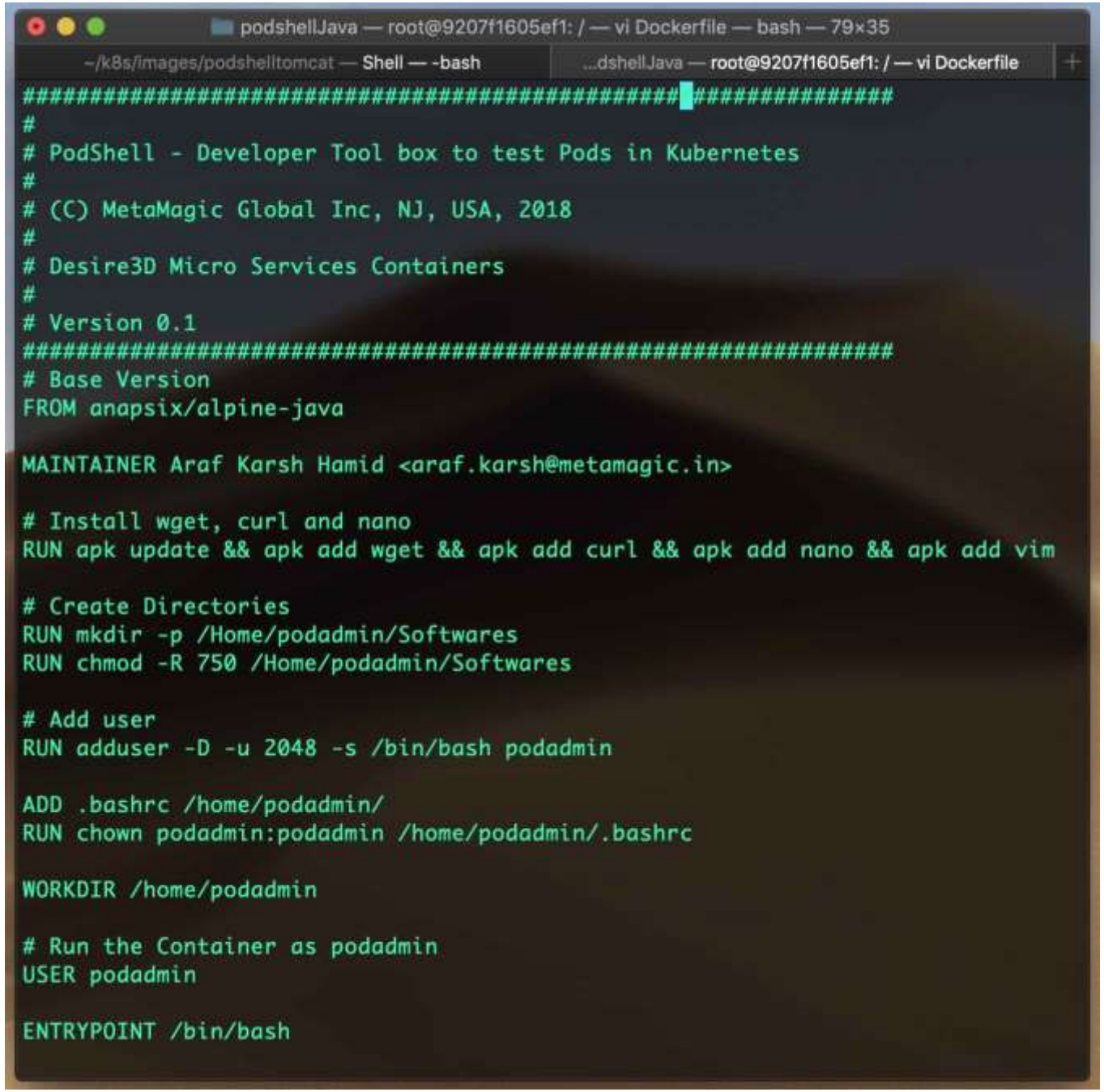
2. **Build the Docker image**

```
$ docker build -t org/java:8 .
```

3. **Run the Container**

```
$ docker container run –it org/java:8
```





Docker Container Management

```
$ ID=$(docker container run –it ubuntu /bin/bash
$ docker container stop $ID
```

Start the Container and Store ID in ID field
Stop the container using Container ID

```
$ docker container stop $(docker container ls –aq)
```

Stops all the containers

```
$ docker container rm $ID
```

Remove the Container

```
$ docker container rm $(docker container ls –aq)
```

Remove ALL the Container (in Exit status)

```
$ docker container start $ID
```

Start the container

```
$ docker container prune
```

Remove ALL stopped Containers)

```
$ docker container run –restart=Policy –d –it ubuntu /sh
```

Policies = NO / ON-FAILURE / ALWAYS

```
$ docker container run –restart=on-failure:3
–d –it ubuntu /sh
```

Will re-start container ONLY 3 times if a failure happens

Docker Container Management

```
$ ID=$(docker container run –d -i ubuntu)
$ docker container exec -it $ID /bin/bash
```

Start the Container and Store ID in ID field
Inject a Process into Running Container

```
$ ID=$(docker container run –d –i ubuntu)
$ docker container exec inspect $ID
```

Start the Container and Store ID in ID field
Read Containers MetaData

```
$ docker container run  –it ubuntu /bin/bash
# apt-get update
# apt-get install—y apache2
# exit

$ docker container ls –a
$ docker container commit –author=“name” –
message=“Ubuntu / Apache2” containerId apache2
```

Docker Commit

- Start the Ubuntu Container
- Install Apache
- Exit Container
- Get the Container ID (Ubuntu)
- Commit the Container with new name

```
$ docker container run –cap-drop=chown –it ubuntu /sh
```

To prevent Chown inside the Container

 Source: https://github.com/meta-magic/kubernetes_workshop

Docker Image Commands

`$ docker login ….`	Log into the Docker Hub to Push images
`$ docker push image-name`	Push the image to Docker Hub
`$ docker image history image-name`	Get the History of the Docker Image
`$ docker image inspect image-name`	Get the Docker Image details
`$ docker image save –output=file.tar image-name`	Save the Docker image as a tar ball.
`$ docker container export –output=file.tar c79aa23dd2`	Export Container to file.

 Source: https://github.com/meta-magic/kubernetes_workshop

Build Docker Apache image

1. Create your Dockerfile

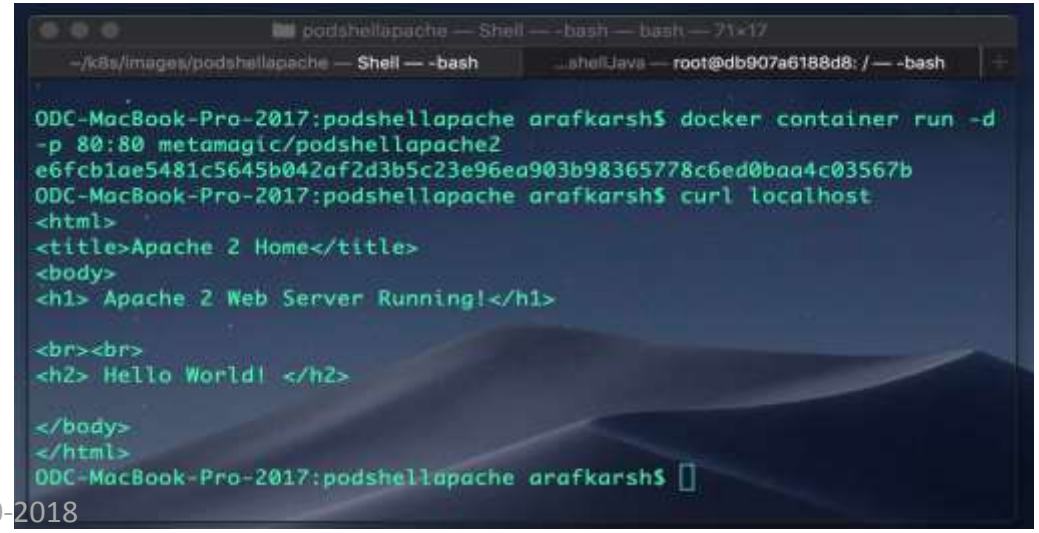
- FROM alpine
- RUN
- COPY
- EXPOSE
- ENTRYPOINT

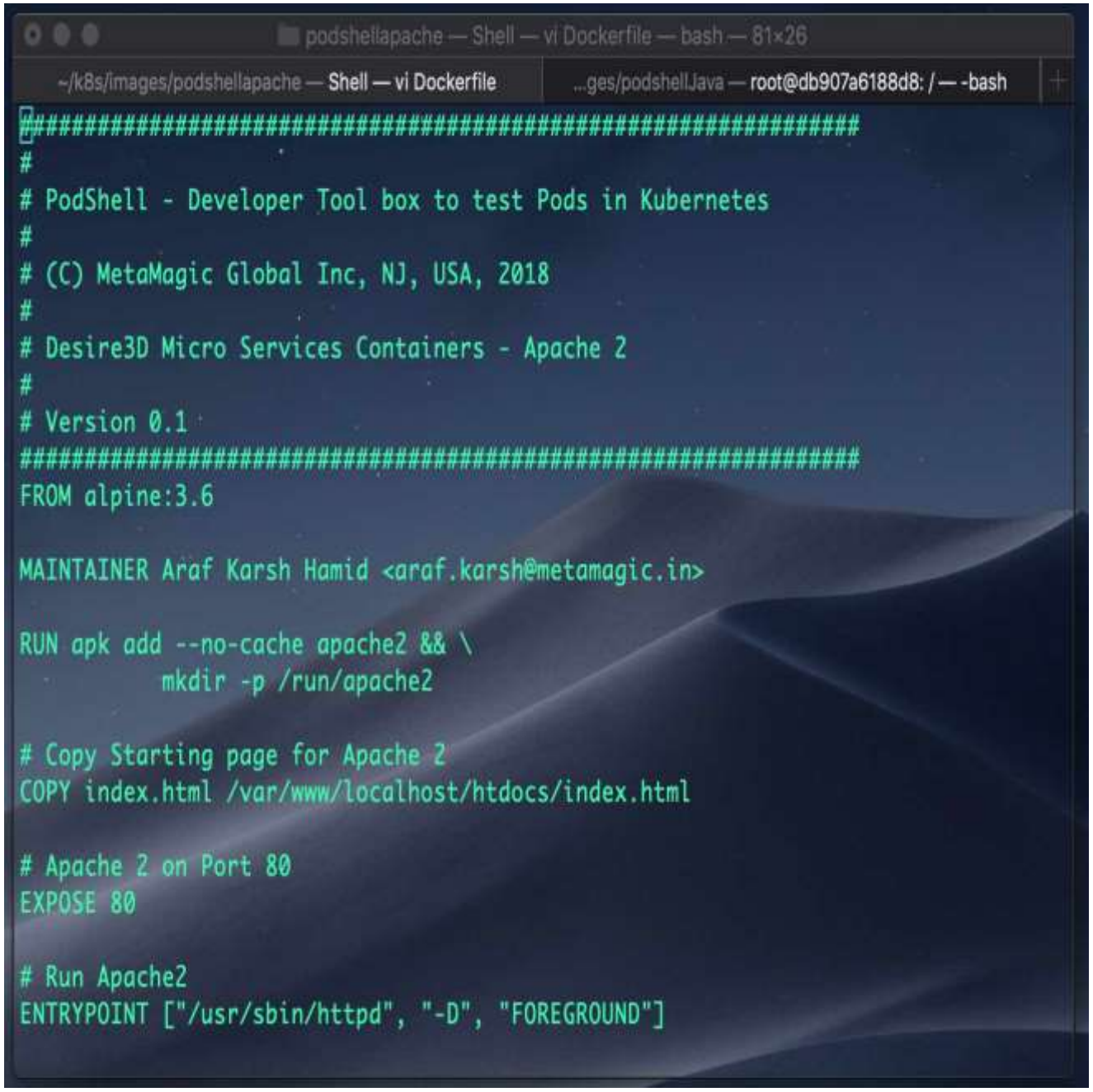
2. Build the Docker image

```
$ docker build -t org/apache2 .
```

3. Run the Container

```
$ docker container run –d –p 80:80 org/apache2
$ curl localhost
```





Build Docker Tomcat image

1. Create your Dockerfile

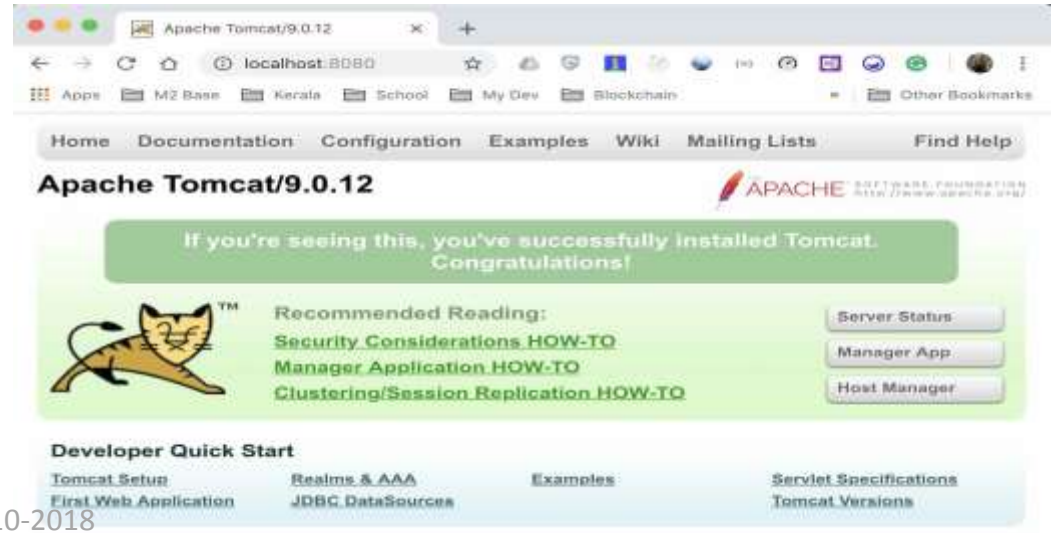
- FROM alpine
- RUN
- COPY
- EXPOSE
- ENTRYPOINT

2. Build the Docker image

```
$ docker build -t org/tomcat .
```

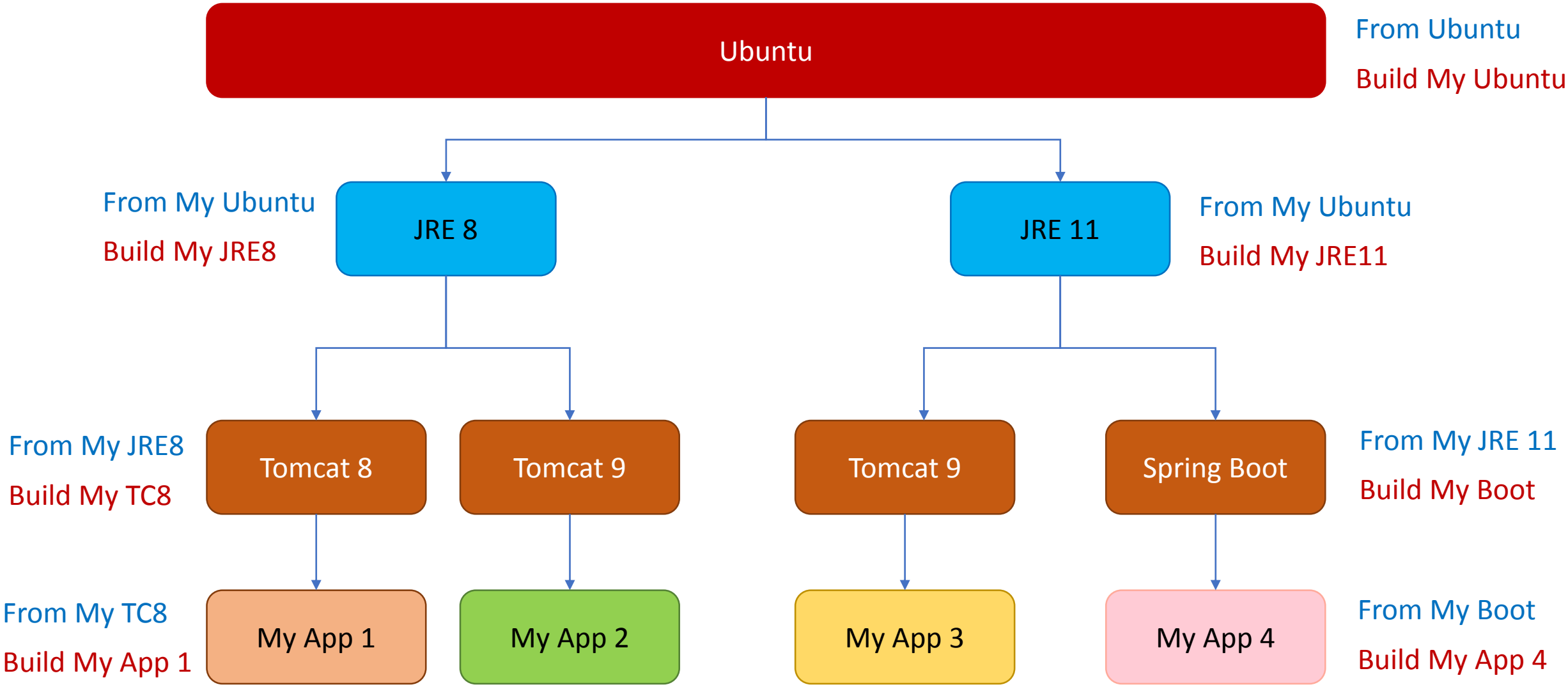
3. Run the Container

```
$ docker container run –d –p 8080:8080 org/tomcat
$ curl localhost:8080
```

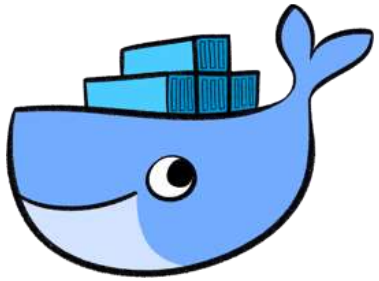



```
podshelltomcat — Shell — vi Dockerfile — bash — 70×27
.../images/podshelltomcat — Shell — vi Dockerfile
...helJava — root@db907a6188d8: / — -bash
#############################################################
#
# PodShell - Developer Tool box to test Pods in Kubernetes
#
# (C) MetaMagic Global Inc, NJ, USA, 2017
#
# Desire3D Micro Services Containers - Tomcat
#
# Version 0.1
#############################################################
# Base Version
FROM metamagic/podshelljava:8

MAINTAINER Araf Karsh Hamid <araf.karsh@metamagic.in>

RUN mkdir /Home/Softwares/tomcat/

# Copy Tomcat (Install)
COPY --chown=podadmin:podadmin tc9 /Home/Softwares/tomcat/
RUN chmod 750 /Home/Softwares/tomcat/bin/*

# Expose Tomcat port
EXPOSE 8080

# Start Tomcat
ENTRYPOINT ["/Home/Softwares/tomcat/bin/catalina.sh", "run"]
```

Docker Images in the Github Workshop



 Source: https://github.com/meta-magic/kubernetes_workshop

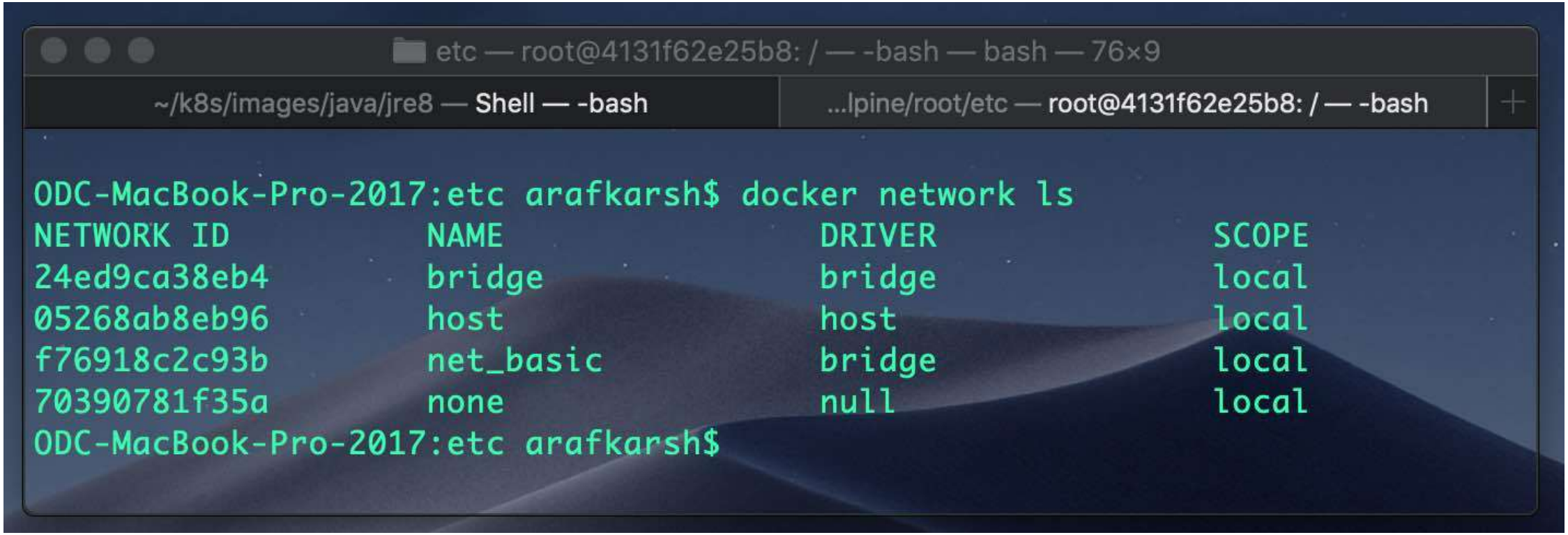


Docker Networking

- Docker Networking – Bridge / Host / None
- Docker Container sharing IP Address
- Docker Communication – Node to Node
- Docker Volumes

Docker Networking – Bridge / Host / None

```
$ docker network ls
```




```
$ docker container run --rm --network=host alpine brctl show
```

```
$ docker network create tenSubnet –subnet 10.1.0.0/16
```

Docker Networking – Bridge / Host / None

https://docs.docker.com/network/#network-drivers

$ docker container run --rm alpine ip address

```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -it metamagic/alpine
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00
245: eth0@if246: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

$ docker container run –rm –net=none alpine ip address

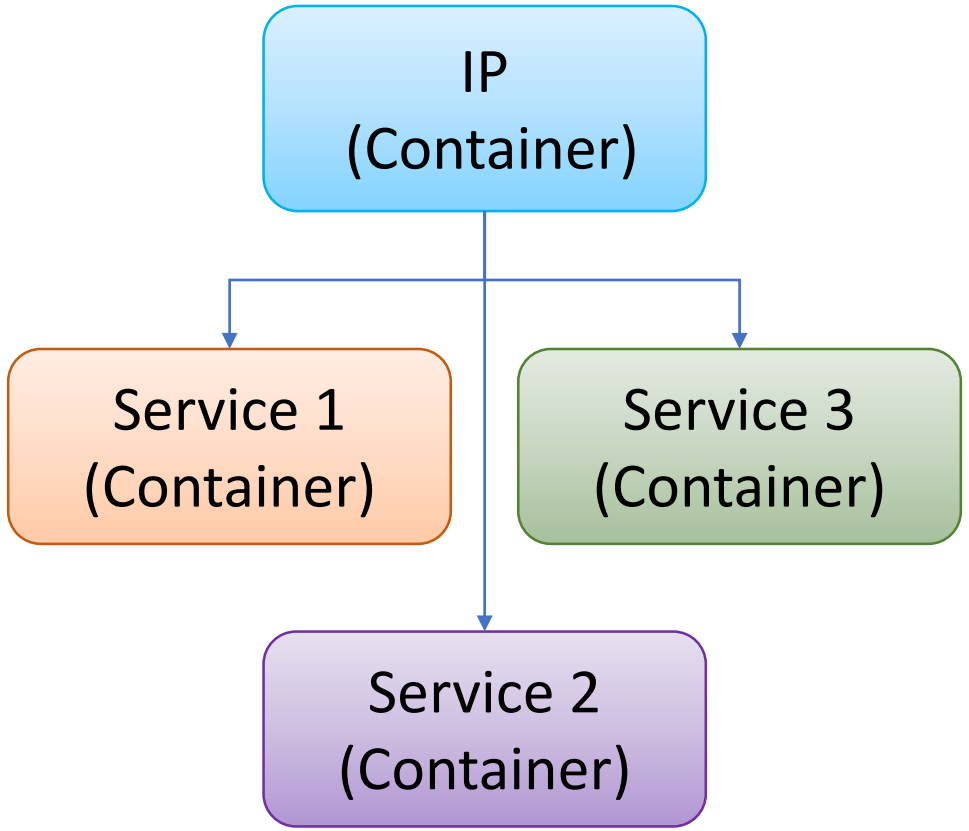
```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -it --net=none metamagic/alpine
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00
/ #
```

No Network Stack

$ docker container run --rm –net=host alpine ip address

```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -it --net=host metamagic/alpine
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 02:50:00:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet 192.168.65.3/24 brd 192.168.65.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::50:ff:fe00:1/64 scope link
       valid_lft forever preferred_lft forever
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
4: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00
5: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ef:18:2e:d0 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:efff:fe18:2ed0/64 scope link
       valid_lft forever preferred_lft forever
6: br-f76918c2c93b: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:ff:21:3e:2a brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-f76918c2c93b
       valid_lft forever preferred_lft forever
190: veth458a5ea@if189: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue mast
er docker0 state UP
    link/ether ce:2a:86:0f:37:c4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::cc2a:86ff:fe0f:37c4/64 scope link
       valid_lft forever preferred_lft forever
240: vetha7165bc@if239: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue mast
er docker0 state UP
    link/ether 46:09:8c:9f:1e:53 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::4409:8cff:fe9f:1e53/64 scope link
       valid_lft forever preferred_lft forever
/ #
```

Docker Containers Sharing IP Address



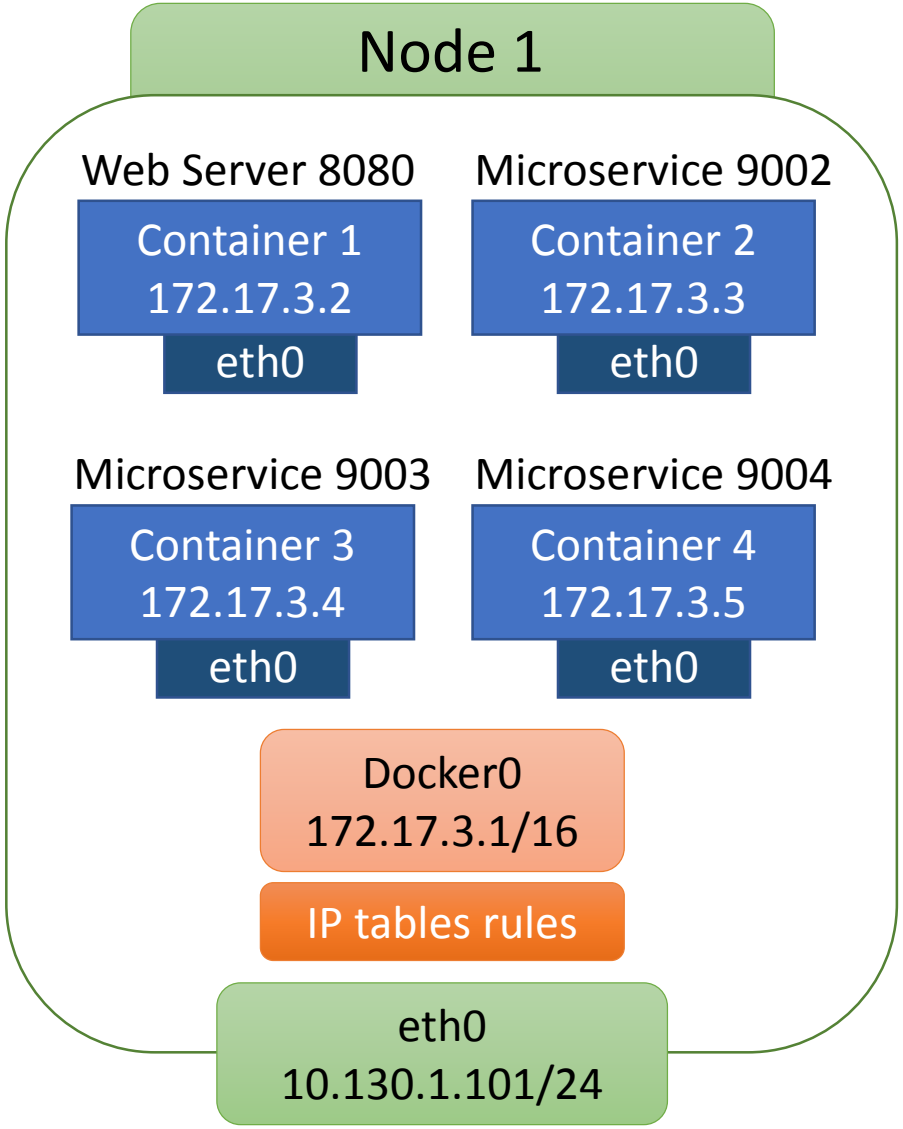
$ docker container run –itd –name ipctr alpine ip address

```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -itd --name ipctr metamagic/alpine
142cdc73df5427e013feafa996e705804134d837d39aa0688393f438c9deedb0
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container exec ipctr ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00
249: eth0@if250: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
ODC-MacBook-Pro-2017:etc arafkarsh$
```

$ docker container run –rm –net container:ipctr alpine ip address

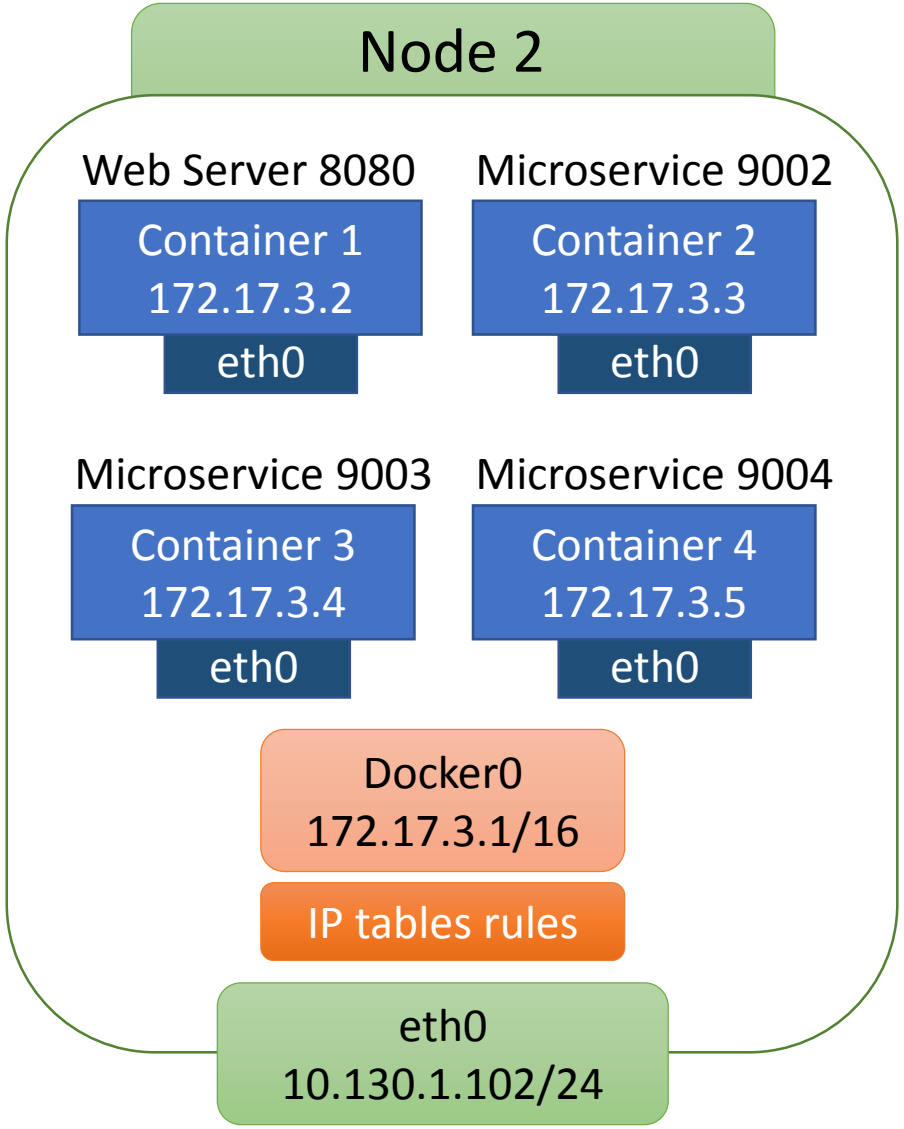
```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run --rm --net container:ipctr metamagi
c/alpine ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00
249: eth0@if250: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
ODC-MacBook-Pro-2017:etc arafkarsh$
```

Docker Networking: Node to Node



Same IP Addresses for the Containers across different Nodes.

This requires NAT.



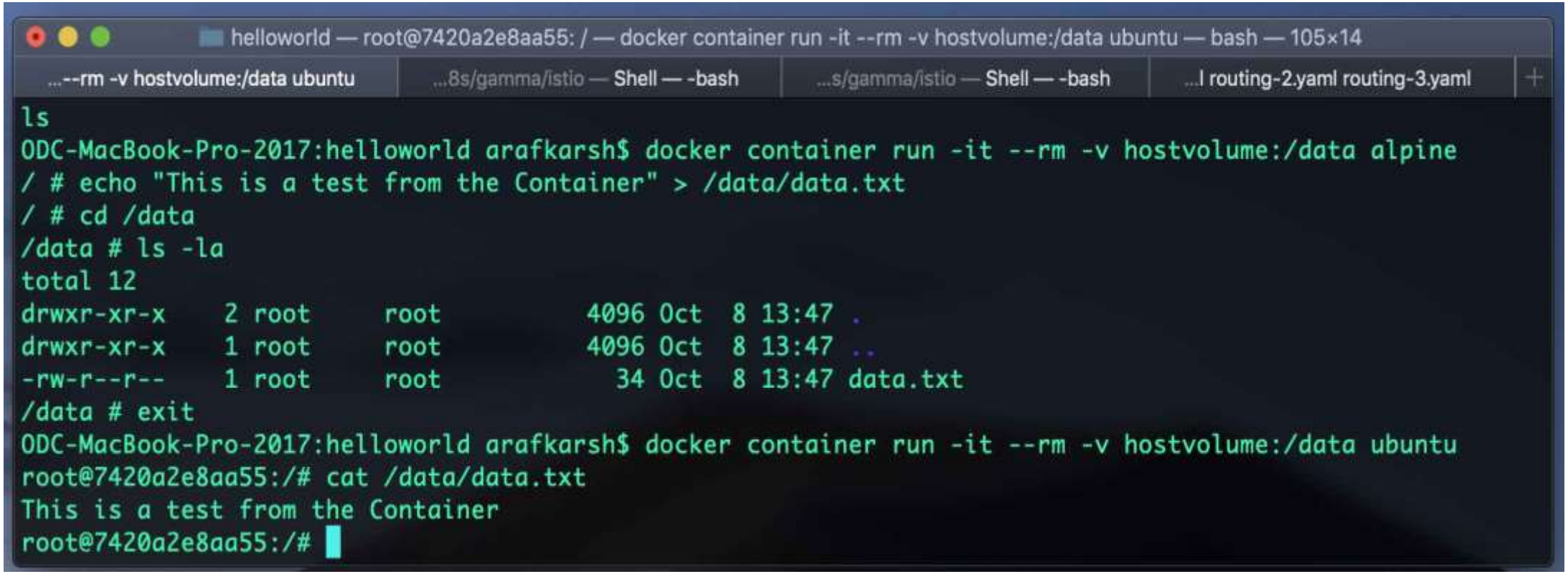
Docker Volumes

Data Volumes are special directory in the Docker Host.

```
$ docker volume create hostvolume
```

```
$ docker volume ls
```

```
$ docker container run –it –rm –v  hostvolume:/data alpine
# echo “This is a test from the Container” > /data/data.txt
```

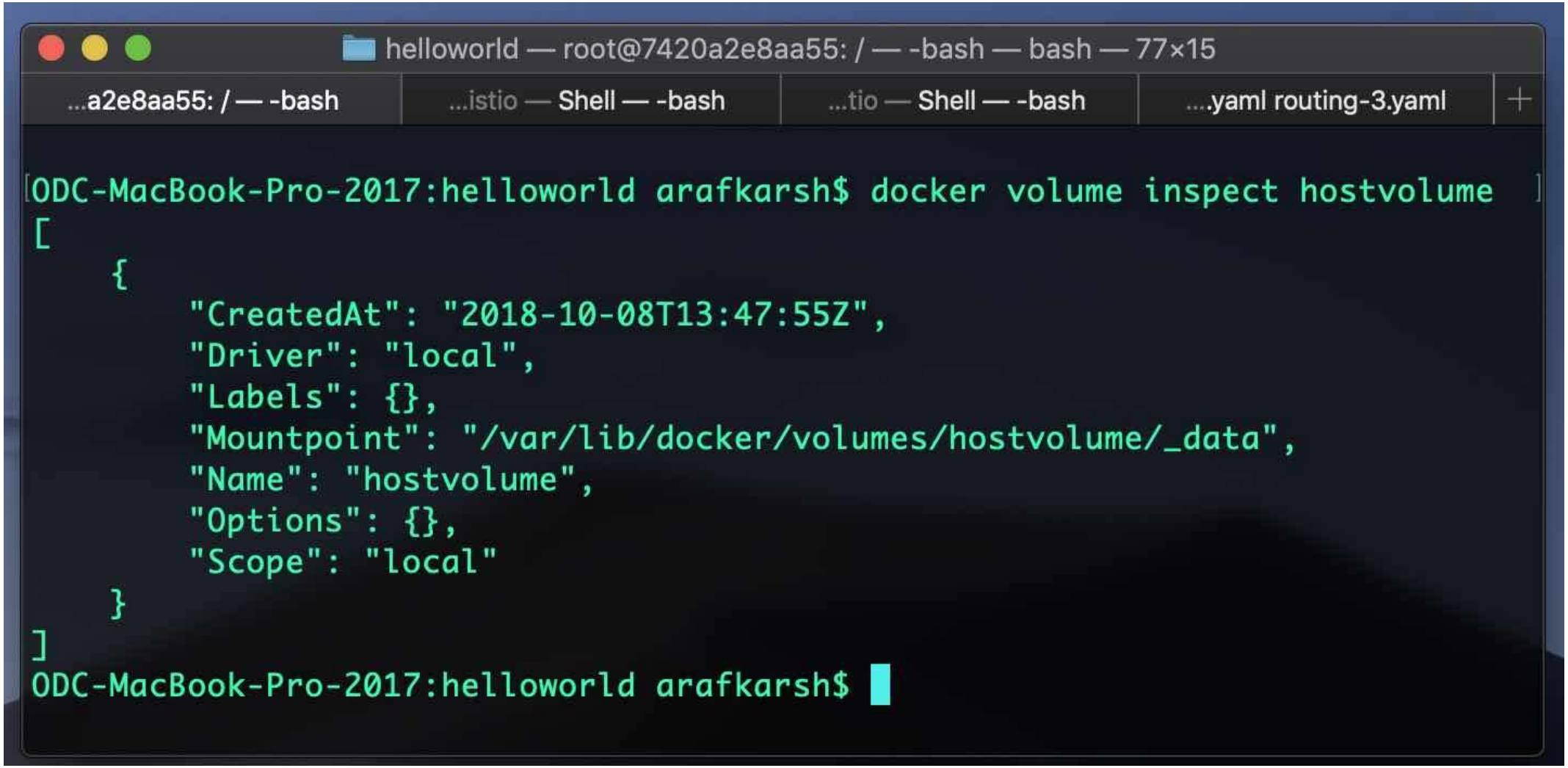


Source: https://github.com/meta-magic/kubernetes_workshop

Docker Volumes

```
$ docker container run  - - rm –v $HOME/data:/data alpine
```

Mount Specific File Path



Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes

Kubernetes Architecture

Key Aspects

- Declarative Model
- Desired State

Using yaml or json declare the desired state of the app. State is stored in the Cluster store.

Master Node (Control Plane)

Cluster

Store
etcd
Key Value Store
Secrets

API Server Port 443

RESTful yaml / json

$ kubectl ....

Scheduler

Controller Manager

Kind
- Pods
- ReplicaSet
- Deployment
- Service
- Endpoints
- StatefulSet
- Namespace
- Resource Quota
- Limit Range
- Persistent Volume

Names

Node Controller | **End Point** Controller | **Deployment** Controller | **Pod** Controller | ....

For the cloud providers to manage nodes, services, routes, volumes etc.

Cloud Controller

Declarative Model
- apiVersion:
- kind:
- metadata:
- spec:

Kind
- Pod
- ReplicaSet
- Service
- Deployment
- Virtual Service
- Gateway, SE, DR
- Policy, MeshPolicy
- RbaConfig
- Prometheus, Rule,
- ListChekcer ...

Worker Node 1

Internet → Firewall → Ingress → K8s Cluster

Allows multiple implementation of containers from v1.7

Port 10255

Kubelet Node Manager

gRPC ProtoBuf

Container Runtime Interface

Kube-Proxy Network Proxy
TCP / UDP Forwarding
IPTABLES / IPVS

POD (Cgroup / Namespaces)

POD itself is a Linux Container, Docker container will run inside the POD. PODs with single or multiple containers (Sidecar Pattern) will share Cgroup, Volumes, Namespaces of the POD.

Namespace 1

Namespace 2

Self healing is done by Kubernetes using watch loops if the desired state is changed.

Cluster IP
Node Port
Load Balancer
External Name

Service
15.1.2.100
DNS: a.b.com

Label Selector: BE, 1.2

EP | Pod IP ...34 ...35 ...36

Pod IP Address is dynamic, communication should be based on **Service which will have routable IP and DNS Name**. **Labels** (BE, 1.2) play a critical role in ReplicaSet, Deployment, & Services etc.

Label Selector selects pods based on the Labels.

Pod — BE, 1.2 — 10.1.2.34

Pod — BE, 1.2 — 10.1.2.35

Pod — BE, 1.2 — 10.1.2.36

Annotations

Labels

D — **Deployment** – Updates and rollbacks, Canary Release

R — **ReplicaSet** – Self Healing, Scalability, Desired State

POD | POD | POD

Label Selector

Label Selector

Kubernetes Setup – Minikube

- **Minikube** provides a developer environment with master and a single node installation within the Minikube with all necessary add-ons installed like DNS, Ingress controller etc.
- In a real world production environment you will have master installed (with a failover) and ‘n’ number of nodes in the cluster.
- If you go with a Cloud Provider like Amazon EKS then the node will be created automatically based on the load.
- Minikube is available for **Linux / Mac OS and Windows.**

Ubuntu Installation

https://kubernetes.io/docs/tasks/tools/install-kubectl/

```
$ sudo snap install kubectl  --classic
```

Install Kubectl using Snap Package Manager

```
$ kubectl version
```

Shows the Current version of Kubectl

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-linux-amd64
$ chmod +x minikube && sudo mv minikube /usr/local/bin/
```


Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Setup – Minikube

Mac OS Installation

https://kubernetes.io/docs/tasks/tools/install-kubectl/

```
$ brew install kubernetes-cli
```
Install Kubectl using brew Package Manager

```
$ kubectl version
```
Shows the Current version of Kubectl

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-darwin-amd64
$ chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Windows Installation

```
C:\> choco install kubernetes-cli
```
Install Kubectl using Choco Package Manager

```
C:\> kubectl version
```
Shows the Current version of Kubectl

```
C:\> cd c:\users\youraccount
C:\> mkdir .kube
```
Create .kube directory

```
C:\> minikube-installer.exe
```
Install Minikube using Minikube Installer

Kubernetes Setup – Master / Nodes


```
$ kubeadm init
```

```
node1$ kubeadm join --token enter-token-from-kubeadm-cmd Node-IP:Port
```

Adds a Node

```
$ kubectl get nodes
```

List all Nodes

```
$ kubectl cluster-info
```

Shows the cluster details

```
$ kubectl get namespace
```

Shows all the namespaces

```
$ kubectl config current-context
```

Shows Current Context

Create a set of Pods for Hello World App with an External IP Address

(Imperative Model)

```
$ kubectl run hello-world --replicas=7 --labels="run=load-balancer-example" --image=metamagic/hello:1.0 --port=8080
```

Creates a Deployment Object and a ReplicaSet object with 7 replicas of Hello-World Pod running on port 8080

```
$ kubectl expose deployment hello-world --type=LoadBalancer --name=hello-world-service
```

Creates a Service Object that exposes the deployment (Hello-World) with an external IP Address.

```
$ kubectl get deployments hello-world
```

List all the Hello-World Deployments

```
$ kubectl describe deployments hello-world
```

Describe the Hello-World Deployments

```
$ kubectl get replicasets
```

List all the ReplicaSet

```
$ kubectl describe replicasets
```

Describe the ReplicaSet

```
$ kubectl get services hello-world-service
```

List the Service Hello-World-Service with Custer IP and External IP

```
$ kubectl describe services hello-world-service
```

Describe the Service Hello-World-Service

```
$ kubectl get pods –o wide
```

List all the Pods with internal IP Address

```
$ kubectl delete services hello-world-service
```

Delete the Service Hello-World-Service

```
$ kubectl delete deployment hello-world
```

Delete the Hello-Word Deployment

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Object Management

The `kubectl` command-line tool supports several different ways to create and manage Kubernetes objects. This document provides an overview of the different approaches.

- **Management techniques**
- **Imperative commands**
- **Imperative object configuration**
- **Declarative object configuration**
- **What's next**

Focus on the Declarative Model

Management techniques

Warning: A Kubernetes object should be managed using only one technique. Mixing and matching techniques for the same object results in undefined behavior.

3 Fundamental Concepts

1. Desired State
2. Current State
3. Declarative Model

Kubernetes Commands – Namespace

(Declarative Model)

- Namespaces are used to group your teams and software's in logical business group.
- A definition of Service will add a entry in DNS with respect to Namespace.
- Not all objects are there in Namespace. Ex. Nodes, Persistent Volumes etc.

sigma-ns.yaml

```
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: sigma
5    labels:
6      name: sigma
7
```

```
$ kubectl get namespace
```
List all the Namespaces

```
$ kubectl describe ns ns-name
```
Describe the Namespace

```
$ kubectl get pods –namespace= ns-name
```
List the Pods from your namespace

```
$ kubectl create –f app-ns.yml
```
Create the Namespace

```
$ kubectl apply –f app-ns.yml
```
Apply the changes to the Namespace

```
$ kubectl config set-context $(kubectl config current-context) --namespace=your-ns
```

The above command will let you switch the namespace to your namespace (your-ns).

Kubernetes Pods

Atomic Unit

Virtual Server | Pod | Container

Big ← → Small

- Pod is a shared environment for one of more Containers.
- Pod in a Kubernetes cluster has a unique IP address, even Pods on the same Node.
- Pod is a pause Container

```
$ kubectl create –f app1-pod.yml
$ kubectl get pods
```

```
ODC-MacBook-Pro-2017:omega arafkarsh$ kubectl create -f app1-pod.yaml
pod "app1pod" created
ODC-MacBook-Pro-2017:omega arafkarsh$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
app1pod   1/1       Running   0          10s
ODC-MacBook-Pro-2017:omega arafkarsh$
```

app1-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app1pod
  namespace: sigma
  labels:
    desire3d.io/name: app1pod
    desire3d.io/version: "1.0.0"
    desire3d.io/release: stable
    desire3d.io/tier: fe
    desire3d.io/zone: prod
    desire3d.io/managed-by: m2
spec:
  containers:
  - name: app1pod-ctr
    image: metamagicglobal/omega
    imagePullPolicy: Always
    ports:
    - containerPort: 8080
    resources:
      requests:
        memory: "64Mi"
        cpu: "200m"
      limits:
        memory: "96Mi"
        cpu: "250m"
```

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – Pods

(Declarative Model)


```
$ kubectl get pods
```
List all the pods

```
$ kubectl describe pods pod-name
```
Describe the Pod details

```
$ kubectl get pods  -o json pod-name
```
List the Pod details in JSON format

```
$ kubectl get pods  -o wide
```
List all the Pods with Pod IP Addresses

```
$ kubectl describe pods –l app=name
```
Describe the Pod based on the label value

```
$ kubectl create –f app-pod.yml
```
Create the Pod

```
$ kubectl apply –f app-pod.yml
```
Apply the changes to the Pod

```
$ kubectl replace –f app-pod.yml
```
Replace the existing config of the Pod

```
$ kubectl exec pod-name ps aux
```
Execute commands in the first Container in the Pod

```
$ kubectl exec –it pod-name sh
```
Log into the Container Shell

```
$ kubectl exec –it –container container-name pod-name sh
```
By default kubectl executes the commands in the first container in the pod. If you are running multiple containers (sidecar pattern) then you need to pass –container flag and give the name of the container in the Pod to execute your command. You can see the ordering of the containers and its name using describe command.

```
$ kubectl logs pod-name container-name
```

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes ReplicaSet

- Pods wrap around containers with benefits like shared location, secrets, networking etc.
- ReplicaSet wraps around Pods and brings in Replication requirements of the Pod
- **ReplicaSet Defines 2 Things**
  - Pod Template
  - Desired No. of Replicas

What we want is the Desired State.

Game On!

app1-rs.yaml

```yaml
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: app1pod-rs
  namespace: sigma
  labels:
    desire3d.io/name: app1rs
spec:
  replicas: 5
  selector:
    matchLabels:
      desire3d.io/name: app1pod
      desire3d.io/zone: prod
      desire3d.io/managed-by: m2
      desire3d.io/release: stable
  template:
    metadata:
      labels:
        desire3d.io/name: app1pod
        desire3d.io/version: "1.0.0"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      containers:
        - name: app1pod-ctr
          image: metamagicglobal/omega
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 8080
          resources:
            requests:
              memory: "64Mi"
              cpu: "200m"
            limits:
              memory: "96Mi"
              cpu: "250m"
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – ReplicaSet

(Declarative Model)

```
$ kubectl get rs
```

List all the ReplicaSets

```
$ kubectl describe rs rs-name
```

Describe the ReplicaSet details

```
$ kubectl get rs/rs-name
```

Get the ReplicaSet status

```
$ kubectl create –f app-rs.yml
```

Create the ReplicaSet which will automatically create all the Pods

```
$ kubectl apply –f app-rs.yml
```

Applies new changes to the ReplicaSet. For example Scaling the replicas from x to x + new value.

```
$ kubectl delete rs/app-rs cascade=false
```

Deletes the ReplicaSet. If the cascade=true then deletes all the Pods, Cascade=false will keep all the pods running and ONLY the ReplicaSet will be deleted.

app1-rs.yaml

```yaml
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: app1pod-rs
  namespace: sigma
  labels:
    desire3d.io/name: app1rs
spec:
  replicas: 5
  selector:
    matchLabels:
      desire3d.io/name: app1pod
      desire3d.io/zone: prod
      desire3d.io/managed-by: m2
      desire3d.io/release: stable
  template:
    metadata:
      labels:
        desire3d.io/name: app1pod
        desire3d.io/version: "1.0.0"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      containers:
        - name: app1pod-ctr
          image: metamagicglobal/omega
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 8080
          resources:
            requests:
              memory: "64Mi"
              cpu: "200m"
            limits:
              memory: "96Mi"
              cpu: "250m"
```

Kubernetes Commands – Deployment

(Declarative Model)

- Deployments manages ReplicaSets and
- ReplicaSets manages Pods
- Deployment is all about Rolling updates and
- Rollbacks
- Canary Deployments

app1-dep.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: app1pod-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      desire3d.io/name: app1pod
      desire3d.io/zone: prod
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        desire3d.io/name: app1pod
        desire3d.io/version: "1.0.0"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      containers:
        - name: app1pod-ctr
          image: metamagicglobal/omega
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 80
          resources:
            requests:
              memory: "64Mi"
              cpu: "200m"
            limits:
              memory: "96Mi"
              cpu: "250m"
```

app1-dep-v2.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: app1pod-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      desire3d.io/name: app1pod
      desire3d.io/zone: prod
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        desire3d.io/name: app1pod
        desire3d.io/version: "1.0.2"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      containers:
        - name: app1pod-ctr
          image: metamagicglobal/omega:v2
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 80
          resources:
            requests:
              memory: "64Mi"
              cpu: "200m"
            limits:
              memory: "96Mi"
              cpu: "250m"
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – Deployment

(Declarative Model)

```
$ kubectl get deploy app-deploy
```

List all the Deployments

```
$ kubectl describe deploy app-deploy
```

Describe the Deployment details

```
$ kubectl rollout status deployment app-deploy
```

Show the Rollout status of the Deployment

```
$ kubectl rollout history deployment app-deploy
```

Show Rollout History of the Deployment

```
$ kubectl create –f app-deploy.yml
```

Creates Deployment
Deployments contains Pods and its Replica information. Based on the Pod info Deployment will start downloading the containers (Docker) and will install the containers based on replication factor.

```
$ kubectl apply –f app-deploy.yml --record
```

Updates the existing deployment.

```
$ kubectl rollout undo deployment app-deploy  - -to-revision=1
```

Rolls back or Forward to a specific version number of your app.

```
$ kubectl rollout undo deployment app-deploy  - -to-revision=2
```

```
$ kubectl scale deployment app-deploy  - -replicas=6
```

Scale up the pods to 6 from the initial 2 Pods.

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Services

Why do we need Services?

- Accessing Pods from Inside the Cluster
- Accessing Pods from Outside
- Autoscale brings Pods with new IP Addresses or removes existing Pods.
- Pod IP Addresses are dynamic.

Service will have a stable IP Address.

Service uses Labels to associate with a set of Pods

app1-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app1pod-svc
  namespace: sigma
  labels:
    desire3d.io/name: app1svc
spec:
  type: NodePort
  selector:
    desire3d.io/name: app1pod
    desire3d.io/zone: prod
    desire3d.io/release: stable
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
```

Service Types

1. Cluster IP (Default)
2. Node Port
3. Load Balancer
4. External Name

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – Service / Endpoints

(Declarative Model)

`$ kubectl get svc`	List all the Services
`$ kubectl describe svc app-service`	Describe the Service details
`$ kubectl get ep app-service`	List the status of the Endpoints
`$ kubectl describe ep app-service`	Describe the Endpoint Details
`$ kubectl create –f app-service.yml`	Create a Service for the Pods. Service will focus on creating a routable IP Address and DNS for the Pods Selected based on the labels defined in the service. Endpoints will be automatically created based on the labels in the Selector.
`$ kubectl delete svc app-service`	Deletes the Service.

- ***Cluster IP*** (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
- ***Node Port*** - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
- ***Load Balancer*** - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.
- ***External Name*** - Exposes the Service using an arbitrary name (specified by external Name in the spec) by returning a CNAME record with the name. No proxy is used. This type requires v1.7 or higher of kube-dns.

Kubernetes Ingress

(Declarative Model)

An Ingress is a collection of rules that allow inbound connections to reach the cluster services.

Ingress is still a beta feature in Kubernetes

Ingress Controllers are Pluggable.

Ingress Controller in AWS is linked to AWS Load Balancer.

shopping-ingress.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: shoppingportal-ingress
  namespace: shoppingportal
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: \"false\"
spec:
  rules:
  - http:
      paths:
      - path: /ui
        backend:
          serviceName: k8uiworkshopservice
          servicePort: 80
      - path: /productms
        backend:
          serviceName: productservice
          servicePort: 80
      - path: /productreviewms
        backend:
          serviceName: productreviewservice
          servicePort: 80
```

Source: https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-controllers

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Ingress

(Declarative Model)

An Ingress is a collection of rules that allow inbound connections to reach the cluster services.

Ingress is still a beta feature in Kubernetes

Ingress Controllers are Pluggable.

Ingress Controller in AWS is linked to AWS Load Balancer.

shopping-ingress-aws.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: workshop-ingress
  namespace: shoppingportal
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/tags: Environment=dev,Team
    alb.ingress.kubernetes.io/subnets: subnet-05ea8630b6
spec:
  rules:
  - http:
      paths:
      - path: /ui/*
        backend:
          serviceName: k8uiworkshopservice
          servicePort: 80
      - path: /productms/*
        backend:
          serviceName: productservice
          servicePort: 80
      - path: /productreviewms/*
        backend:
          serviceName: productreviewservice
          servicePort: 80
```

Source: https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-controllers

Kubernetes Auto Scaling Pods

(Declarative Model)

- You can declare the Auto scaling requirements for every Deployment (Microservices).
- Kubernetes will add Pods based on the CPU Utilization automatically.
- Kubernetes Cloud infrastructure will automatically add Nodes if it ran out of available Nodes.

product-horizontal-scaler.yaml

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: product-hpa
  namespace: shoppingportal
spec:
  scaleTargetRef:
    apiVersion:  apps/v1beta2
    kind: Deployment
    name: productreview-deploy
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 10
```

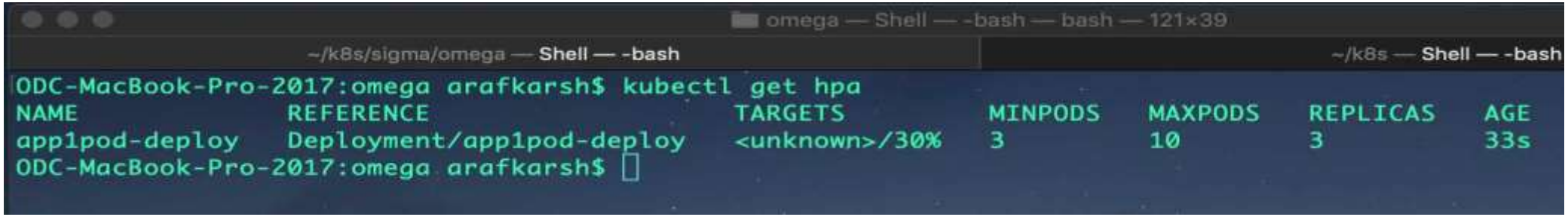
CPU utilization kept at 10% to demonstrate the auto scaling feature. Ideally it should be around 80% - 90%

Kubernetes Horizontal Pod Auto Scaler

Deploy your app with auto scaling parameters

```
$ kubectl autoscale deployment appname --cpu-percent=50 --min=1 --max=10
```

```
$ kubectl get hpa
```



Generate load to see auto scaling in action

```
$ kubectl run -it podshell --image=metamagicglobal/podshell

Hit enter for command prompt

$ while true; do wget -q -O- http://yourapp.default.svc.cluster.local; done
```

To attach to the running container

```
$ kubectl attach podshell-name -c podshell -it
```

Kubernetes Networking

- Comparison between Docker and Kubernetes Networking
- Kubernetes DNS
- Pod to Pod Networking within the same Node
- Pod to Pod Networking across the Node
- Pod to Service Networking
- Ingress - Internet to Service Networking
- Egress – Pod to Internet Networking

Kubernetes Networking

Mandatory requirements for Network implementation

1. All Pods can communicate with All other Pods without using Network Address Translation (NAT).
2. All Nodes can communicate with all the Pods without NAT.
3. The IP that is assigned to a Pod is the same IP the Pod sees itself as well as all other Pods in the cluster.

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Networking

3 Networks

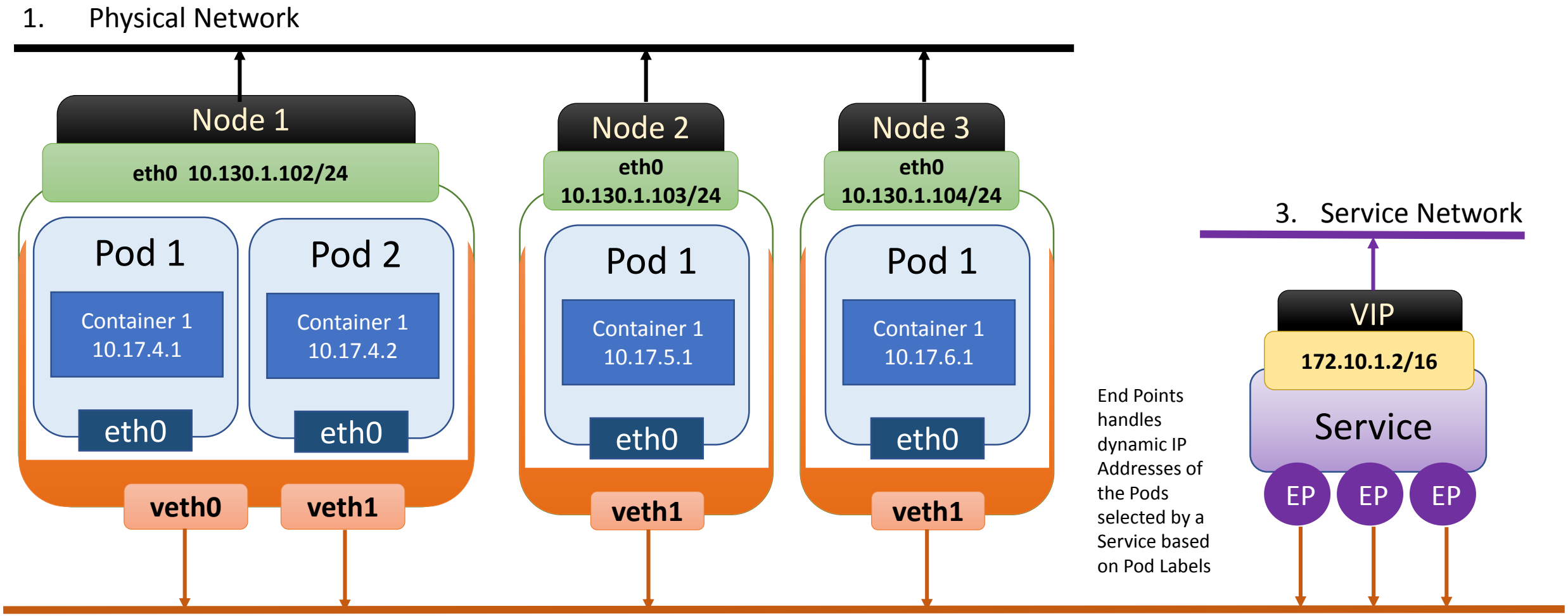
Networks	CIDR Range (RFC 1918)
1. Physical Network	1. 10.0.0.0/8
2. Pod Network	2. 172.0.0.0/11
3. Service Network	3. 192.168.0.0/16

Keep the Address ranges separate.

Source: https://github.com/meta-magic/kubernetes_workshop

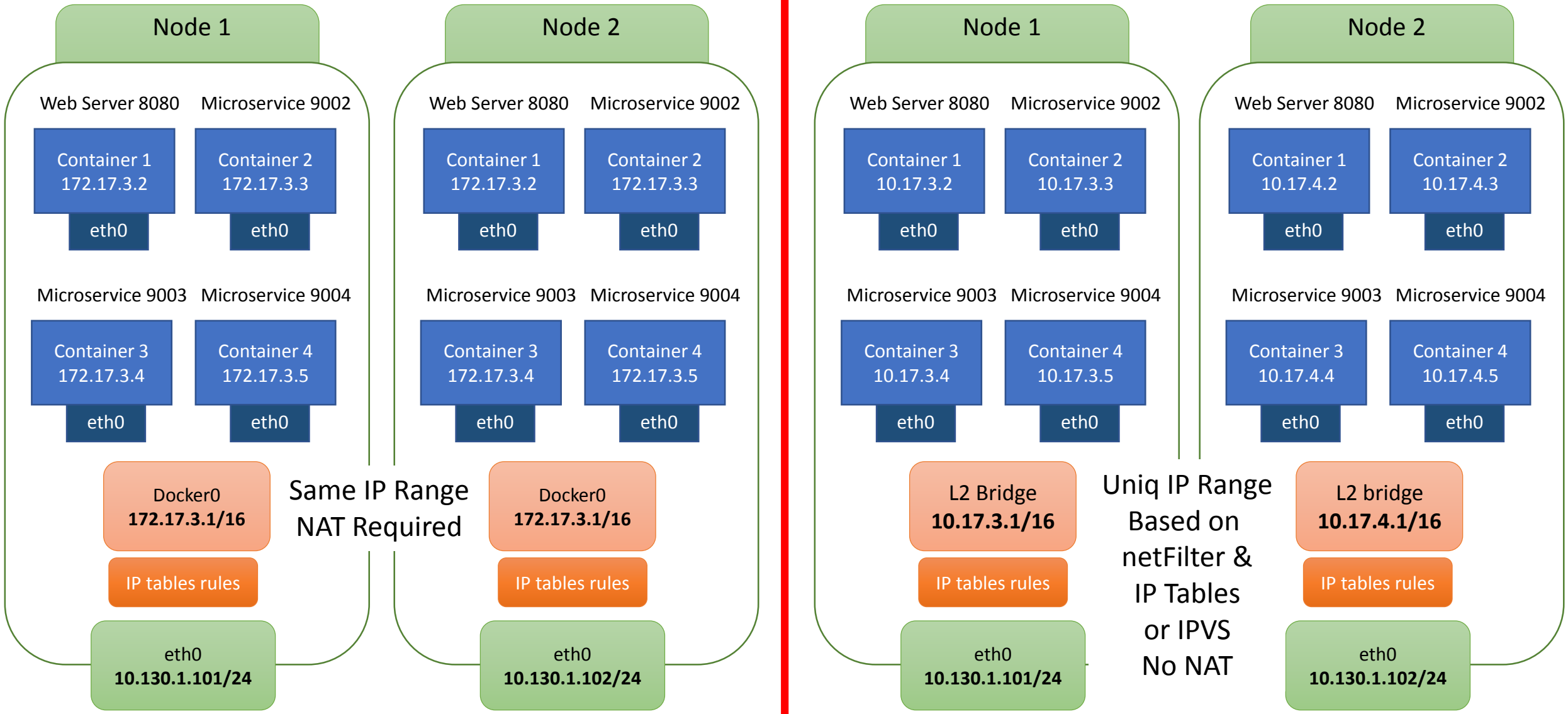
Kubernetes Networking

3 Networks



Source: https://github.com/meta-magic/kubernetes_workshop

Docker Networking Vs. Kubernetes Networking



Docker Networking

Kubernetes Networking

Kubernetes DNS

Kubernetes DNS to avoid IP Addresses in the configuration or Application Codebase.

It Configures Kubelet running on each Node so the containers uses DNS Service IP to resolve the IP Address.

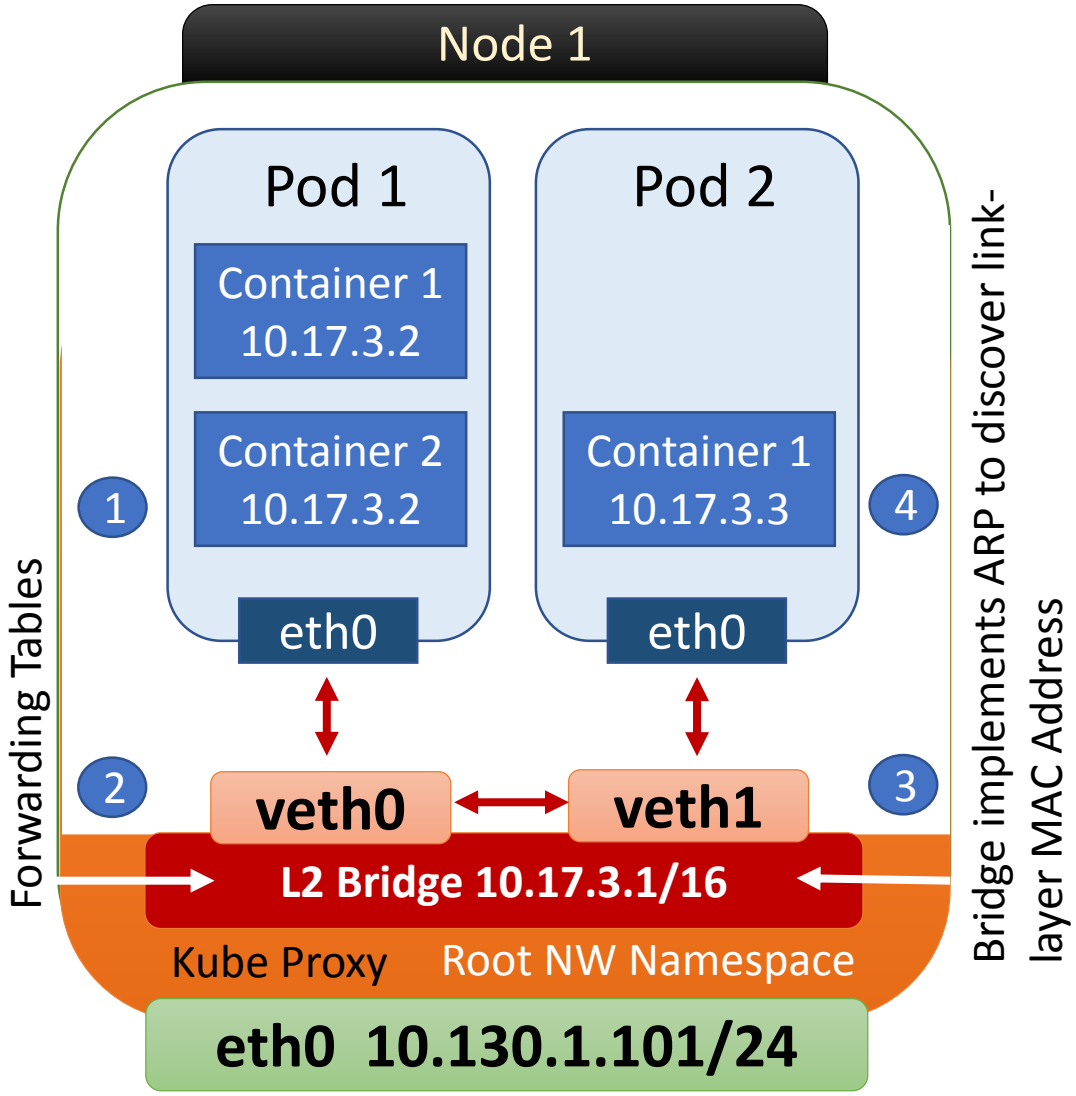
A DNS Pod consists of three separate containers

1. **Kube DNS**: Watches the Kubernetes Master for changes in Service and Endpoints
2. **DNS Masq**: Adds DNS caching to Improve the performance
3. **Sidecar**: Provides a single health check endpoint to perform health checks for Kube DNS and DNS Masq.

- DNS Pod itself is a Kubernetes Service with a Cluster IP.
- DNS State is stored in etcd.
- Kube DNS uses a library the converts etcd name – value pairs into DNS Records.
- **Core DNS** is similar to Kube DNS but with a plugin Architecture in v1.11 Core DNS is the default DNS Server.

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes: Pod to Pod Networking inside a Node



By Default Linux has a Single Namespace and all the process in the namespace share the Network Stack. If you create a new namespace then all the process running in that namespace will have its own Network Stack, Routes, Firewall Rules etc.

```
$ ip netns add namespace1
```

Create Namespace

A mount point for namespace1 is created under /var/run/netns

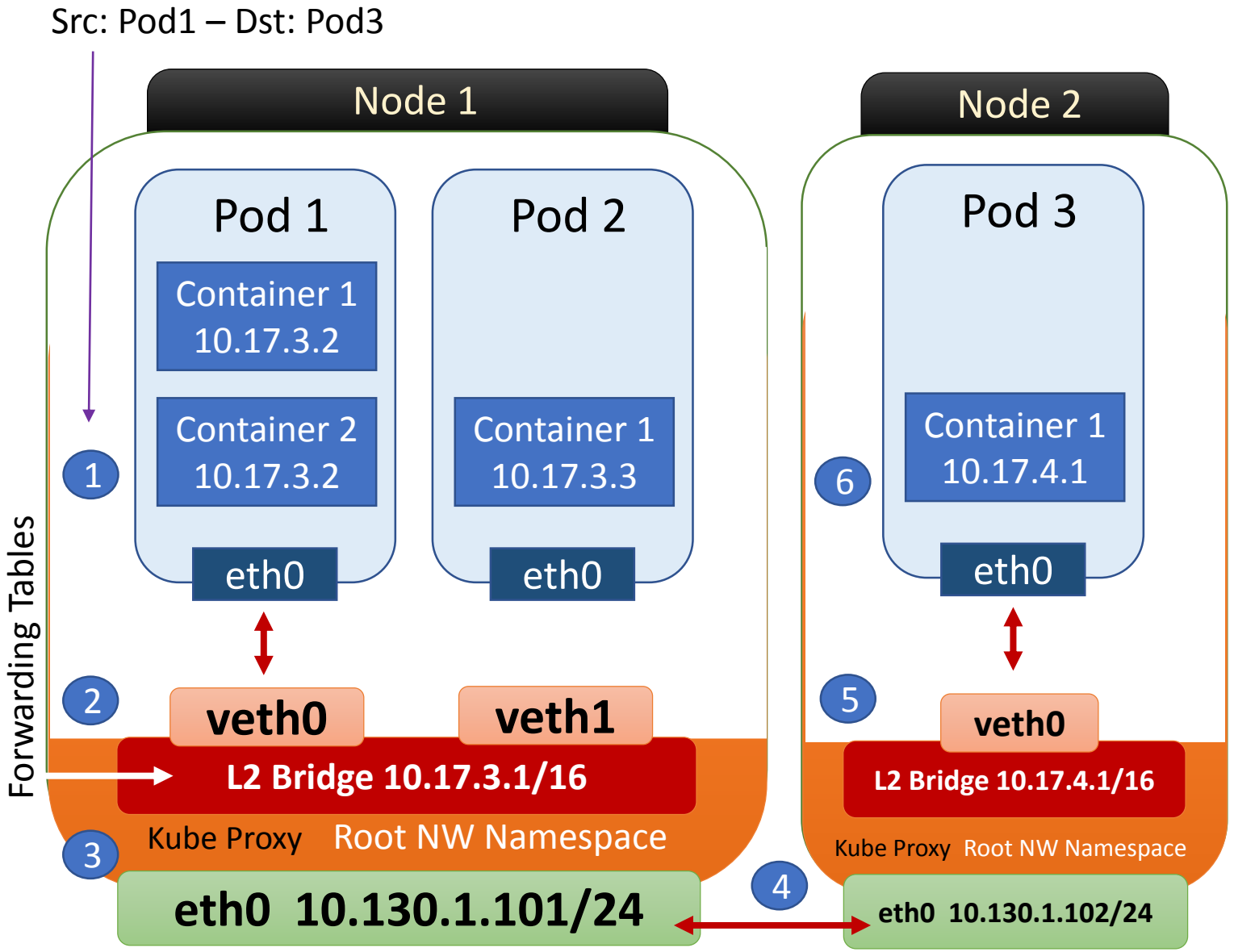
```
$ ip netns
```

List Namespace

1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge resolves the Destination with ARP protocol and
3. Bridge sends the packet to veth1
4. veth1 forwards the packet directly to Pod 2 thru eth0

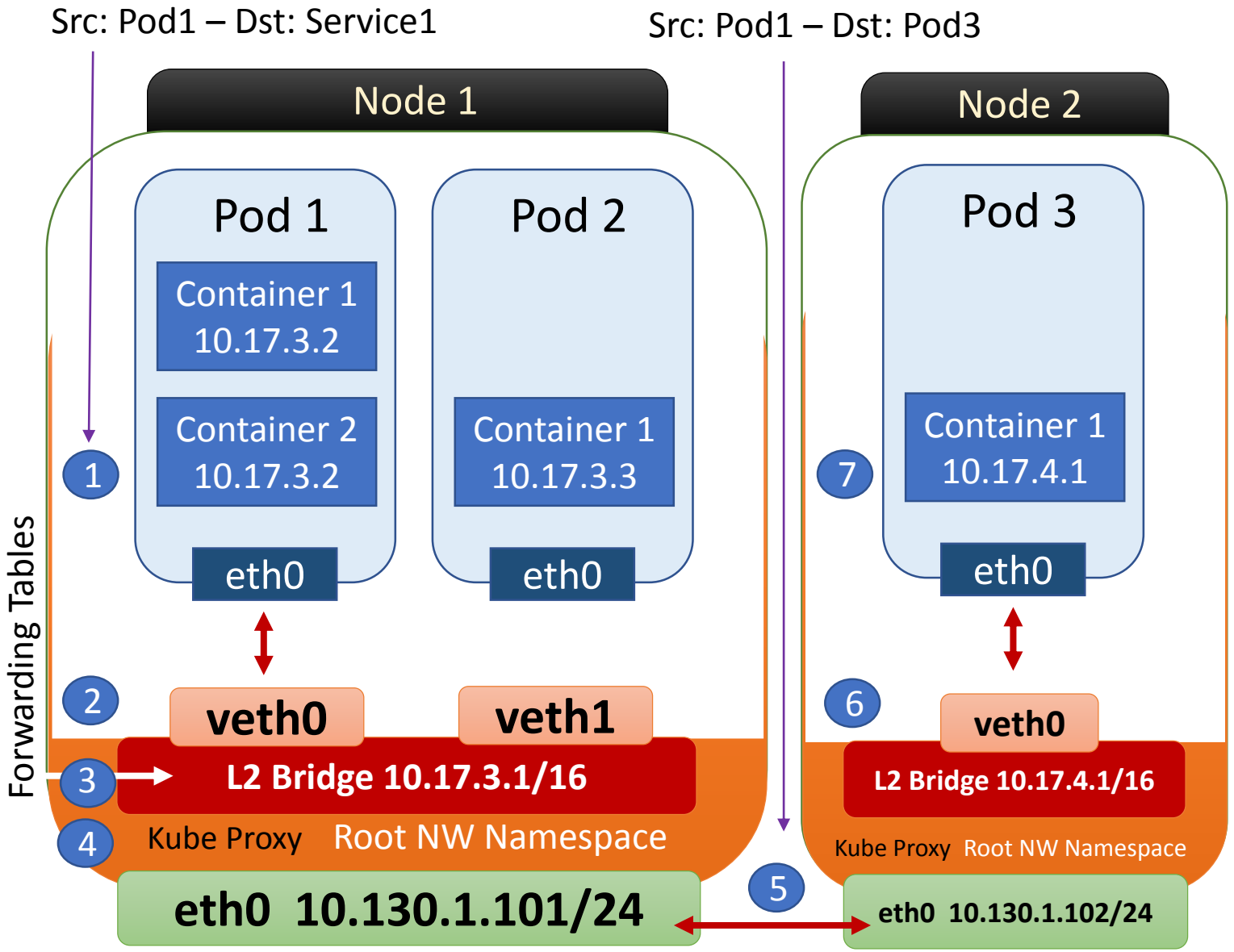
This entire communication happens in localhost. So Data transfer speed will NOT be affected by Ethernet card speed.

Kubernetes: Pod to Pod Networking Across Node



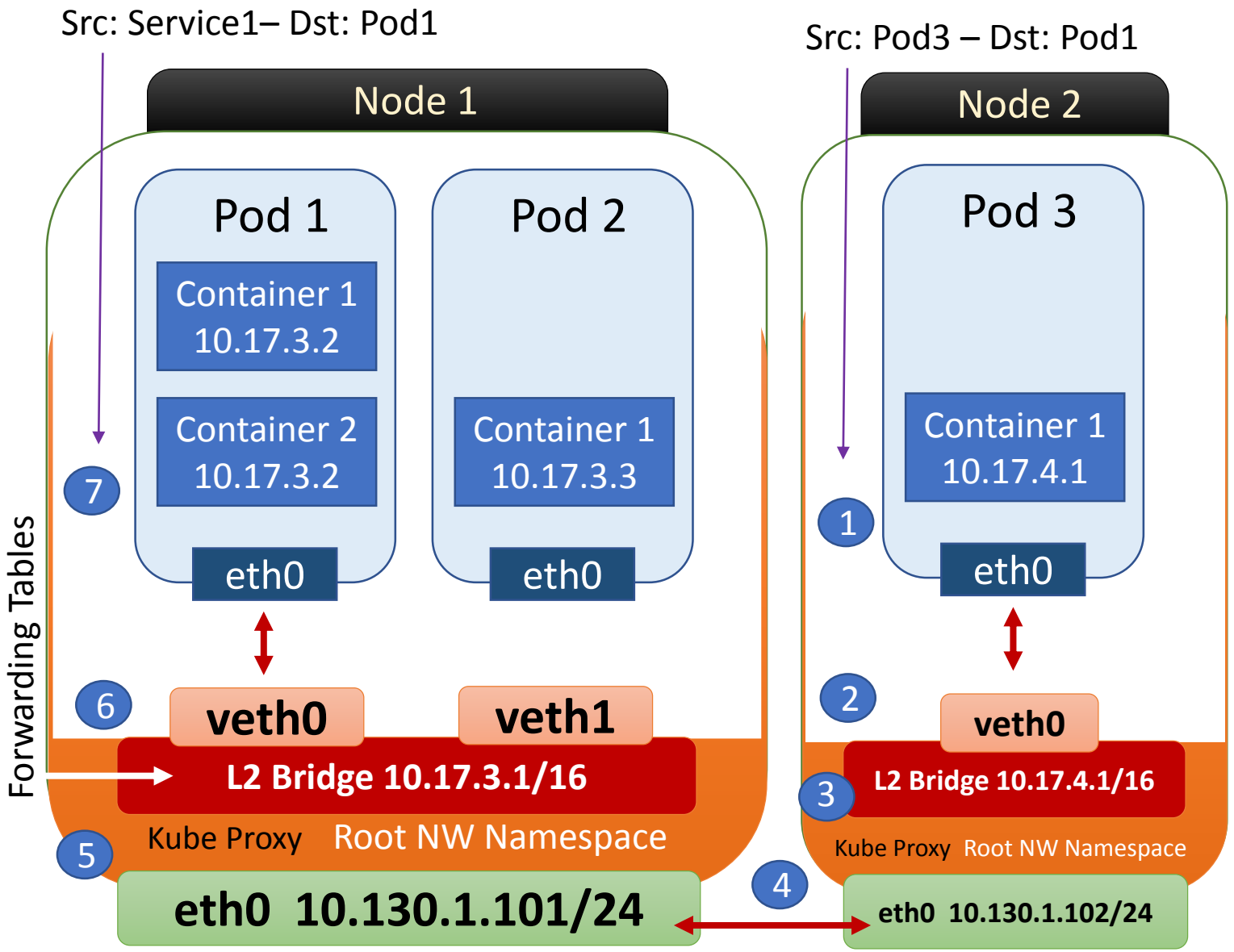
1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge will try to resolve the Destination with ARP protocol and ARP will fail because there is no device connected to that IP.
3. On Failure Bridge will send the packet to eth0 of the Node 1.
4. At this point packet leaves eth0 and enters the Network and network routes the packet to Node 2.
5. Packet enters the Root namespace and routed to the L2 Bridge.
6. veth0 forwards the packet to eth0 of Pod 3

Kubernetes: Pod to Service to Pod – Load Balancer



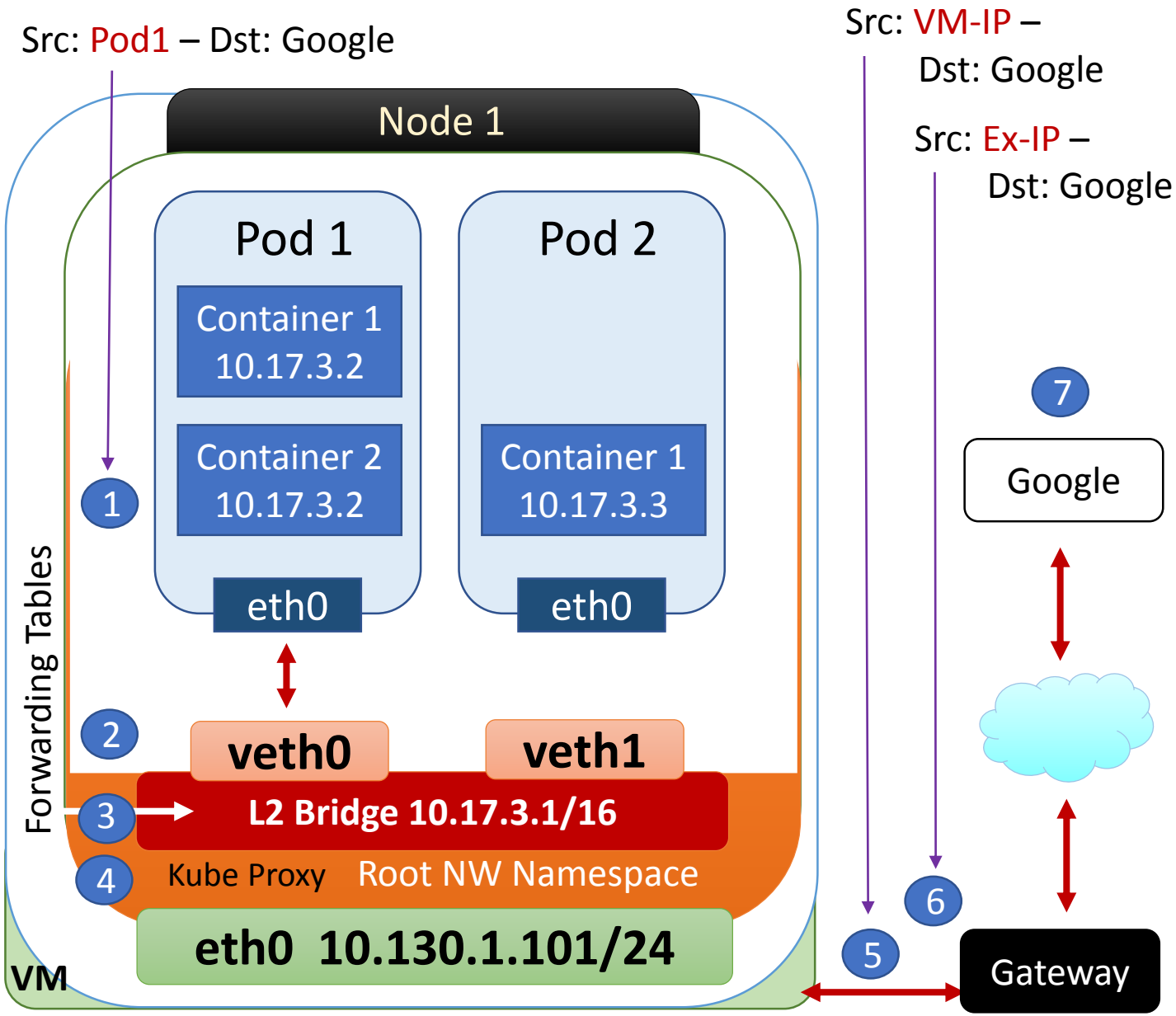
1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge will try to resolve the Destination with ARP protocol and ARP will fail because there is no device connected to that IP.
3. On Failure Bridge will give the packet to Kube Proxy
4. it goes thru ip tables rules installed by Kube Proxy and rewrites the Dst-IP with Pod3-IP. IP tables has done the Cluster load Balancing directly on the node and packet is given to eth0.
5. Now packet leaves Node 1 eth0 and enters the Network and network routes the packet to Node 2.
6. Packet enters the Root namespace and routed to the L2 Bridge.
7. veth0 forwards the packet to eth0 of Pod 3

Kubernetes Pod to Service to Pod – Return Journey



1. Pod 3 receives data from Pod 1 and sends the reply back with Source as Pod3 and Destination as Pod1
2. Bridge will try to resolve the Destination with ARP protocol and ARP will fail because there is no device connected to that IP.
3. On Failure Bridge will give the packet Node 2 eth0
4. Now packet leaves Node 2 eth0 and enters the Network and network routes the packet to Node 1. (Dst = Pod1)
5. it goes thru ip tables rules installed by Kube Proxy and rewrites the Src-IP with Service-IP. Kube Proxy gives the packet to L2 Bridge.
6. L2 bridge makes the ARP call and hand over the packet to veth0
7. veth0 forwards the packet to eth0 of Pod1

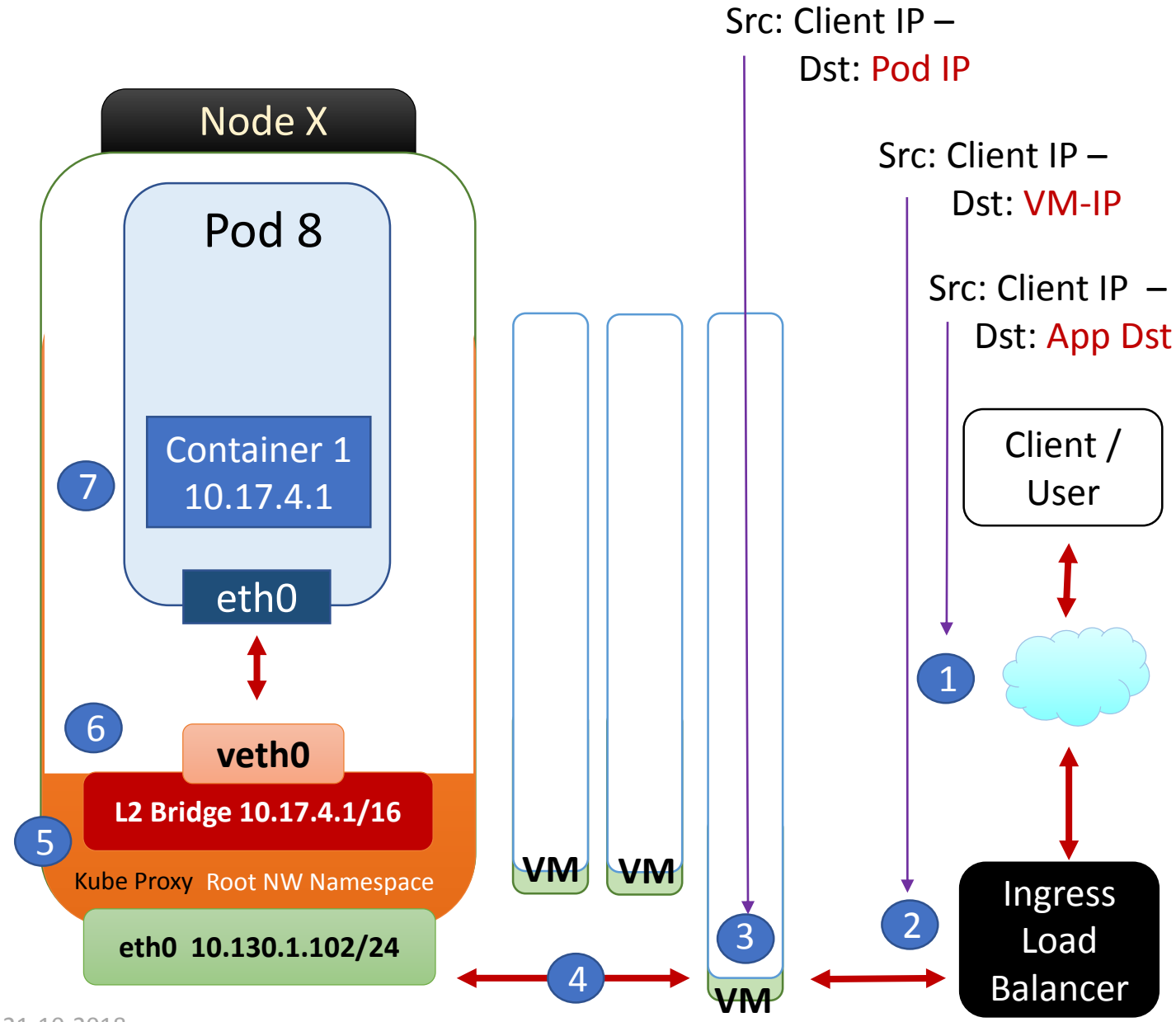
Kubernetes: Pod to Internet



1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge will try to resolve the Destination with ARP protocol and ARP will fail because there is no device connected to that IP.
3. On Failure Bridge will give the packet to IP Tables
4. The Gateway will reject the Pod IP as it will recognize only the VM IP. So source IP is replaced with VM-IP
5. Packet enters the network and routed to Internet Gateway.
6. Packet reaches the GW and it replaces the VM-IP (internal) with an External IP.
7. Packet Reaches External Site (Google)

On the way back the packet follows the same path and any Src IP mangling is un done and each layer understands VM-IP and Pod IP within Pod Namespace.

Kubernetes: Internet to Pod



1. Client Connects to App published Domain.
2. Once the Load Balancer receives the packet it picks a VM.
3. Once inside the VM IP Tables knows how to redirect the packet to the Pod using internal load Balancing rules installed into the cluster using Kube Proxy.
4. Traffic enters Kubernetes cluster and reaches the Node X
5. Node X gives the packet to the L2 Bridge
6. L2 bridge makes the ARP call and hand over the packet to veth0
7. veth0 forwards the packet to eth0 of Pod 8

Networking Glossary

Layer 2 Networking

Layer 2 is the Data Link Layer (OSI Mode) providing Node to Node Data Transfer.

Layer 4 Networking

Transport layer controls the reliability of a given link through flow control.

Layer 7 Networking

Application layer networking (HTTP, FTP etc) This is the closet layer to the end user.

Source Network Address Translation

SNAT refers to a NAT procedure that modifies the source address of an IP Packet.

Destination Network Address Translation

DNAT refers to a NAT procedure that modifies the Destination address of an IP Packet.

ConnTrack

Conntrack is built on top of netfilter to handle connection tracking..

Netfilter – Packet Filtering in Linux

Software that does packet filtering, NAT and other Packet mangling

IP Tables

It allows Admin to configure the netfilter for managing IP traffic.

IPVS – IP Virtual Server

Implements a transport layer load balancing as part of the Linux Kernel. It's similar to IP Tables and based on netfilter hook function and uses hash table for the lookup.

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Network Policies

a4-Ingress-Allow-All.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
  namespace: sigma
spec:
  podSelector: {}
  ingress:
  - {}
```

a4-Ingress-Deny-All.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: sigma
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

a4-Egress-Allow-All.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
  namespace: sigma
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

a4-Egress-Deny-All.yaml

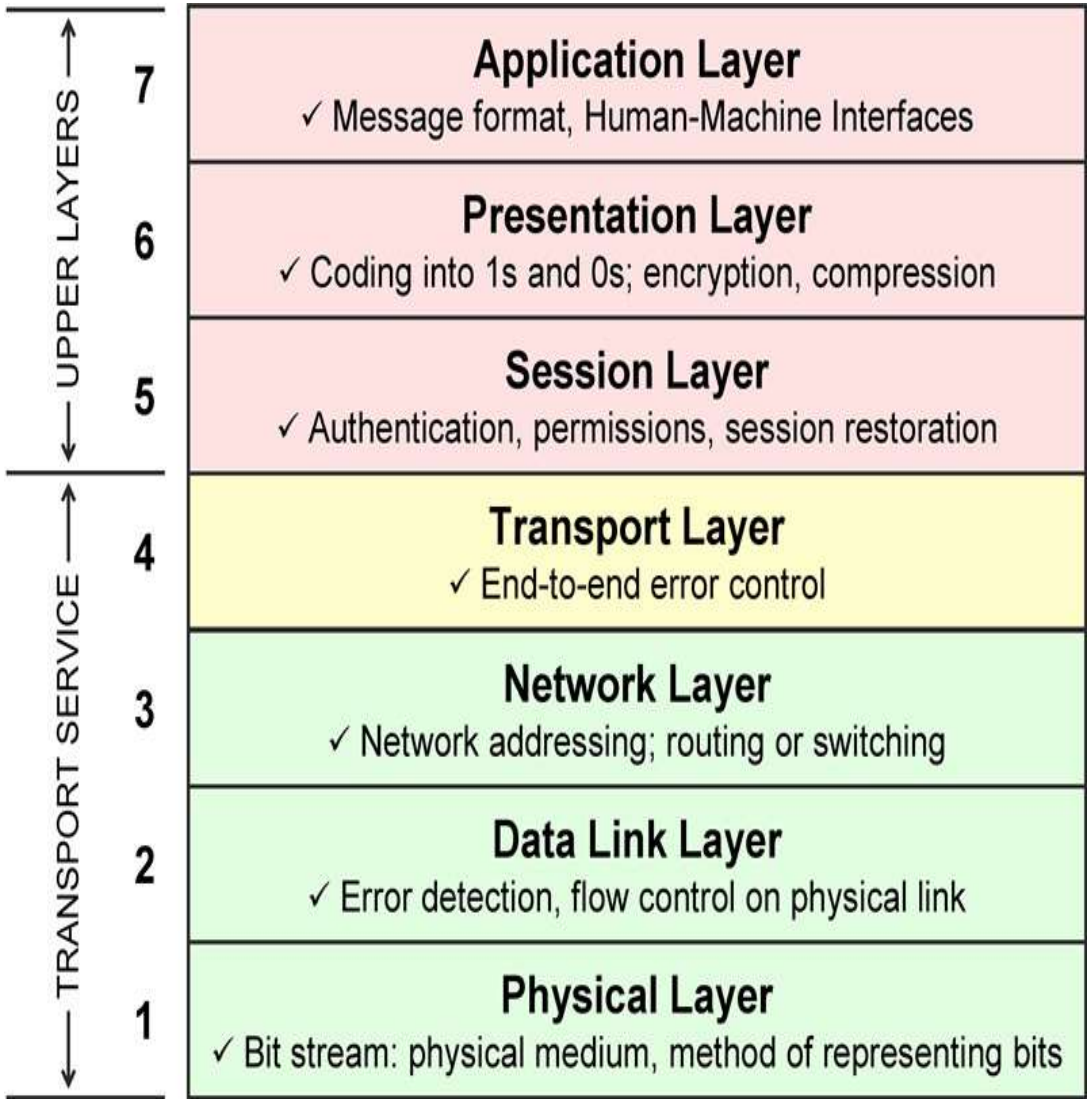
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: sigma
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

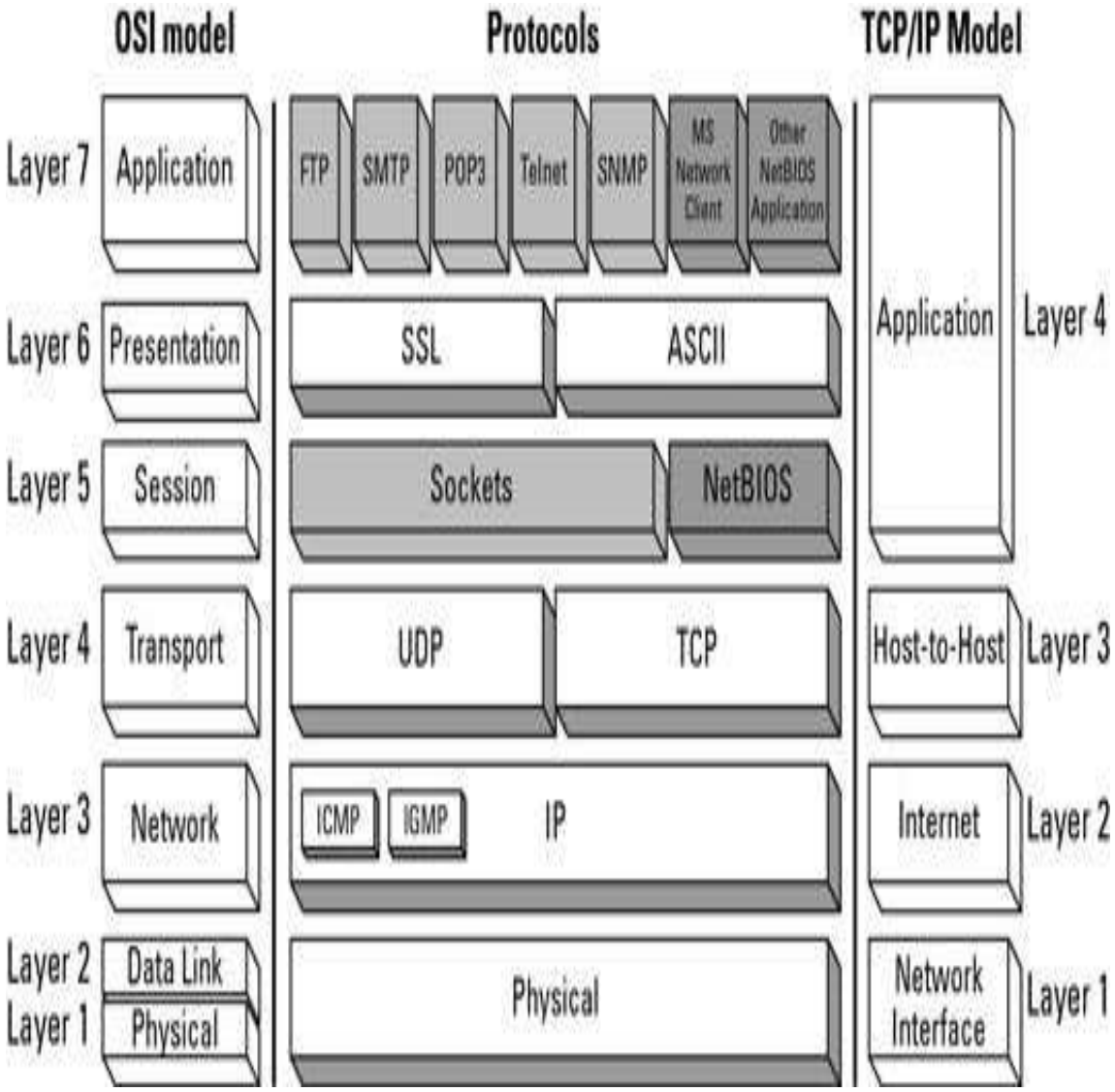
a4-networkpolicy.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: sigma
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

 Source: https://github.com/meta-magic/kubernetes_workshop

OSI Layers


UPPER LAYERS
TRANSPORT SERVICE
7
Application Layer
✓ Message format, Human-Machine Interfaces
6
Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
5
Session Layer
✓ Authentication, permissions, session restoration
4
Transport Layer
✓ End-to-end error control
3
Network Layer
✓ Network addressing; routing or switching
2
Data Link Layer
✓ Error detection, flow control on physical link
1
Physical Layer
✓ Bit stream: physical medium, method of representing bits


OSI model
Protocols
TCP/IP Model
Layer 7
Application
FTP
SMTP
POP3
Telnet
SNMP
MS Network Client
Other NetBIOS Application
Application
Layer 4
Layer 6
Presentation
SSL
ASCII
Layer 5
Session
Sockets
NetBIOS
Layer 4
Transport
UDP
TCP
Host-to-Host
Layer 3
Layer 3
Network
ICMP
IGMP
IP
Internet
Layer 2
Layer 2
Data Link
Layer 1
Physical
Physical
Network Interface
Layer 1

Kubernetes Pods Advanced

- Quality of Service: Resource Quota and Limits
- Environment Variables and Config Maps
- Pod in Depth / Secrets / Presets
- Pod Disruption Range
- Pod / Node Affinity
- Persistent Volume / Persistent Volume Claims

Kubernetes Pod Quality of Service

QoS: Guaranteed	**QoS: Burstable**	**QoS: Best Effort**
Memory limit = Memory Request	!= Guaranteed	No
CPU Limit = CPU Request	and Has either Memory OR CPU Request	Memory OR CPU Request / limits

Source: https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/

Kubernetes Resource Quotas

- A resource quota, defined by a Resource Quota object, provides constraints that limit aggregate resource consumption per namespace.

- It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that project.

Source: https://kubernetes.io/docs/concepts/policy/resource-quotas/

sigma-rq.yaml

```
1  apiVersion: v1
2  kind: ResourceQuota
3  metadata:
4    name: sigma-dev
5  spec:
6    hard:
7      requests.cpu: 500m
8      requests.memory: 1000Mib
9      limits.cpu: 700m
10     limits.memory: 500Mib
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Limit Range

- Limits specifies the Max resource a Pod can have.
- If there is NO limit is defined, Pod will be able to consume more resources than requests. However, the eviction chances of Pod is very high if other Pods with Requests and Resource Limits are defined.

sigma-lr.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: sigma-lr
spec:
  limits:
  - default:
      memory: 256Mi
      cpu: 600m
    defaultRequest:
      memory: 128Mi
      cpu: 100m
    max:
      cpu: 1000m
      memory: 200Mib
    min:
      cpu: 10m
      memory: 10Mib
    type: Container
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod Environment Variables

app1-pod-log-env-vars.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-envars-fieldref
  namespace: sigma
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "sh", "-c"]
      args:
      - while true; do
          echo -en '\n';
          printenv MY_NODE_NAME MY_POD_NAME MY_POD_NAMESPACE;
          printenv MY_POD_IP MY_POD_SERVICE_ACCOUNT;
          sleep 10;
        done;
      env:
        - name: MY_NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: MY_POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: MY_POD_SERVICE_ACCOUNT
          valueFrom:
            fieldRef:
              fieldPath: spec.serviceAccountName
  restartPolicy: Never
```

app1-pod-log-env-vars-2.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-envars-resourcefieldref
  namespace: sigma
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox:1.24
      command: [ "sh", "-c"]
      args:
      - while true; do
          echo -en '\n';
          printenv MY_CPU_REQUEST MY_CPU_LIMIT;
          printenv MY_MEM_REQUEST MY_MEM_LIMIT;
          sleep 10;
        done;
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      env:
        - name: MY_CPU_REQUEST
          valueFrom:
            resourceFieldRef:
              containerName: test-container
              resource: requests.cpu
        - name: MY_CPU_LIMIT
          valueFrom:
            resourceFieldRef:
              containerName: test-container
              resource: limits.cpu
        - name: MY_MEM_REQUEST
          valueFrom:
            resourceFieldRef:
              containerName: test-container
              resource: requests.memory
        - name: MY_MEM_LIMIT
          valueFrom:
            resourceFieldRef:
              containerName: test-container
              resource: limits.memory
  restartPolicy: Never
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Adding Config to Pod

Config Maps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

app1-configmap-2a.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: sigma
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

app1-configmap-2.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-configmap
  namespace: sigma
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c",
      "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
  restartPolicy: Never
```

Source: https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod in Depth

A probe is an indicator to a container's health. It judges the health through periodically performing a diagnostic action against a container via kubelet:

- **Liveness probe**: Indicates whether a container is alive or not. If a container fails on this probe, kubelet kills it and may restart it based on the restartPolicy of a pod.

- **Readiness probe**: Indicates whether a container is ready for incoming traffic. If a pod behind a service is not ready, its endpoint won't be created until the pod is ready.

3 kinds of action handlers can be configured to perform against a container:

exec: Executes a defined command inside the container. Considered to be successful if the exit code is 0.

tcpSocket: Tests a given port via TCP, successful if the port is opened.

httpGet: Performs an HTTP GET to the IP address of target container. Headers in the request to be sent is customizable. This check is considered to be healthy if the status code satisfies: 400 > CODE >= 200.

Additionally, there are five parameters to define a probe's behavior:

initialDelaySeconds: How long kubelet should be waiting for before the first probing.
successThreshold: A container is considered to be healthy when getting consecutive times of probing successes passed this threshold.
failureThreshold: Same as preceding but defines the negative side.
timeoutSeconds: The time limitation of a single probe action.
periodSeconds: Intervals between probe actions.

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes
Pod Liveness Probe

- **Liveness probe**: Indicates whether a container is alive or not. If a container fails on this probe, kubelet kills it and may restart it based on the restartPolicy of a pod.

Source: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

productreview-deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: productreview-deploy
  namespace: shoppingportal
spec:
  replicas: 1
  selector:
    matchLabels:
      name: productreview
      zone: prod
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        name: productreview
        version: "1.0.0"
        release: stable
        tier: fe
        zone: prod
        managed-by: m2
    spec:
      containers:
        - name: productreview-ctr
          image: metamagicglobal/productreviewms
          imagePullPolicy: Always
          ports:
          - containerPort: 8082
          livenessProbe:
            httpGet:
              path: /productreviewms/check/live
              port: 8082
            initialDelaySeconds: 15
            periodSeconds: 15
```

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod Secrets

Objects of type secret are intended to hold sensitive information,

such as passwords,

OAuth tokens, and ssh keys.

Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a docker

productreview-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: productreviewmssecret
  namespace: shoppingportal
type: Opaque
data:
  secretkey: cm9vdA==
```

productreview-deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: productreview-deploy
  namespace: shoppingportal
spec:
  replicas: 1
  selector:
    matchLabels:
      name: productreview
      zone: prod
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        name: productreview
        version: "1.0.0"
        release: stable
        tier: fe
        zone: prod
        managed-by: m2
    spec:
      containers:
        - name: productreview-ctr
          image: metamagicglobal/productreviewms
          imagePullPolicy: Always
          ports:
          - containerPort: 8082
          env:
          - name: PRODUCTREVIEW_VERSION
            valueFrom:
              secretKeyRef:
                name: productreviewmssecret
                key: secretkey
```

Source: https://kubernetes.io/docs/concepts/configuration/secret/

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod Presets

A Pod Preset is an API resource for injecting additional runtime requirements into a Pod at creation time. You use label selectors to specify the Pods to which a given Pod Preset applies.

Using a Pod Preset allows pod template authors to not have to explicitly provide all information for every pod. This way, authors of pod templates consuming a specific service do not need to know all the details about that service.

Source: https://kubernetes.io/docs/concepts/workloads/pods/podpreset/

app1-pod-presets-def.yaml

```yaml
apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: allow-database
  namespace: sigma
spec:
  selector:
    matchLabels:
      role: backend
  env:
    - name: DB_PORT
      value: "6379"
```

app1-pod-presets.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: customersvc
  namespace: sigma
  labels:
    app: customersvc
    role: backend
spec:
  containers:
    - name: customersvc-ctr
      image: nginx
      ports:
        - containerPort: 8080

```

Kubernetes Pod Disruption Range

- A PDB limits the number pods of a replicated application that are down simultaneously from voluntary disruptions.

- Cluster managers and hosting providers should use tools which respect Pod Disruption Budgets by calling the Eviction API instead of directly deleting pods.

app1-pdb.yaml

```yaml
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: app1pod-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      desire3d.io/name: app1pod
      desire3d.io/zone: prod
```

```
$ kubectl drain NODE [options]
```

Source: https://kubernetes.io/docs/tasks/run-application/configure-pdb/

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod/Node Affinity / Anti-Affinity

- You can constrain a pod to only be able to run on particular nodes or to prefer to run on particular nodes. There are several ways to do this, and they all uselabel selectors to make the selection.

- Assign the label to Node
- Assign Node Selector to a Pod

app1-pod-affinity1.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: sigma
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

```
$ kubectl label nodes k8s.node1 disktype=ssd
```

Source: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Pod Configuration

Pod configuration

You use labels and annotations to attach metadata to your resources. To inject data into your resources, you'd likely create ConfigMaps (for non-confidential data) or Secrets (for confidential data).

Taints and Tolerations - These provide a way for nodes to "attract" or "repel" your Pods. They are often used when an application needs to be deployed onto specific hardware, such as GPUs for scientific computing. Read more.

Pod Presets - Normally, to mount runtime requirements (such as environmental variables, ConfigMaps, and Secrets) into a resource, you specify them in the resource's configuration file. PodPresets allow you to dynamically inject these requirements instead, when the resource is created. For instance, this allows team A to mount any number of new Secrets into the resources created by teams B and C, without requiring action from B and C.

Source: https://kubernetes.io/docs/user-journeys/users/application-developer/advanced/

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Volumes for Stateful Pods

Persistent Volume
/ Storage Class

Provision
Network
Storage

Static / Dynamic

1

Persistent Volume Claim

Request
Storage

2

Claims are mounted as
Volumes inside the Pod

Use
Storage

3

Kubernetes Volume

Persistent Volume

- A **Persistent Volume** is the physical storage available.
- **Storage Class** is used to configure custom Storage option (nfs, cloud storage) in the cluster. They are the *foundation of Dynamic Provisioning.*
- **Persistent Volume Claim** is used to mount the required storage into the Pod.

Access Mode

- **ReadOnlyMany:** Can be mounted as read-only by many nodes
- **ReadWriteOnce:** Can be mounted as read-write by a single node
- **ReadWriteMany**: Can be mounted as read-write by many nodes

Volume Mode

- There are two modes
- **File System** and or
- raw Storage **Block**.
- Default is **File System.**

Reclaim Policy

Retain: The volume will need to be reclaimed manually

Delete: The associated storage asset, such as AWS EBS, GCE PD, Azure disk, or OpenStack Cinder volume, is deleted

Recycle: Delete content only (rm -rf /volume/*)

Persistent Volume

Storage Class

Persistent Volume Claim

Source: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#claims-as-volumes

Kubernetes Volume Types

Host Based

- EmptyDir
- HostPath
- Local

Distributed File System

- NFS
- Ceph
- Gluster
- FlexVolume
- PortworxVolume
- Amazon EFS
- Azure File System

Block Storage

- Amazon EBS
- OpenStack Cinder
- GCE Persistent Disk
- Azure Disk
- vSphere Volume

Others

- iScsi
- Flocker
- Git Repo
- Quobyte

Life cycle of a Persistent Volume

- Provisioning
- Binding
- Using
- Releasing
- Reclaiming

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Persistent Volume - hostPath

- **HostPath** option is to make the Volume available from the Host Machine.
- A Volume is created and its linked with a storage provider. In the following example the storage provider is Minikube for the host path.
- Any PVC (Persistent Volume Claim) will be bound to the Persistent Volume which matches the storage class.
- If it doesn't match a dynamic persistent volume will be created.

Storage class is mainly meant for dynamic provisioning of the persistent volumes.

Persistent Volume is not bound to any specific namespace.

v-local-storage.yaml

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: omega-sc-local
provisioner: k8s.io/minikube-hostpath
volumeBindingMode: Immediate
```

1

v-local-volume.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: omega-volume-local
  labels:
    name: storage-local
spec:
  storageClassName: omega-sc-local
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/Users/arafkarsh/data"
```

Change the above path in your system

Source: https://github.com/meta-magic/kubernetes_workshop

Persistent Volume - hostPath

- Persistent Volume Claim and Pods with Deployment properties are bound to a specific namespace.
- Developer is focused on the availability of storage space using PVC and is not bothered about storage solutions or provisioning.
- Ops Team will focus on Provisioning of Persistent Volume and Storage class.

Pod Access storage by issuing a Persistent Volume Claim.
In the following example Pod claims for 2Gi Disk space from the network on the host machine.

v-local-pvc.yaml

2

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: omega-pvc-local
  namespace: omega-local
spec:
  storageClassName: omega-sc-local
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

v-local-dep.yaml

3

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: omega-local-deploy
  namespace: omega-local
spec:
  replicas: 3
  selector:
    matchLabels:
      desire3d.io/name: omega-pod
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        desire3d.io/name: omega-pod
        desire3d.io/version: "1.0.0"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      volumes:
        - name: omega-volume-local
          persistentVolumeClaim:
           # Local Storage Claim
           claimName: omega-pvc-local
      containers:
        - name: omega-ctr
          image: metamagic/oshell
          env:
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
          # Mount omega-volume-local from
          # persistentVolumeClaim
            - mountPath: "/home/data"
              name: omega-volume-local
```

Source: https://github.com/meta-magic/kubernetes_workshop

Persistent Volume - hostPath

1. Create Static Persistent Volumes and Dynamic Volumes (using Storage Class)
2. Persistent Volume Claim is created and bound static and dynamic volumes.
3. Pods refer PVC to mount volumes inside the Pod.

Running the Yaml's from the Github

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Persistent Volume – AWS EBS

- Use a Network File System or Block Storage for Pods to access and data from multiple sources. **AWS EBS** is such a storage system.
- A Volume is created and its linked with a storage provider. In the following example the storage provider is AWS for the EBS.
- Any PVC (Persistent Volume Claim) will be bound to the Persistent Volume which matches the storage class.

Storage class is mainly meant for dynamic provisioning of the persistent volumes.

Persistent Volume is not bound to any specific namespace.

v-aws-storage.yaml

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: omega-sc-aws-slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  zone: us-east-1b
  iopsPerGB: "10"
```

1

v-aws-volume.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: omega-volume-aws
  labels:
    name: storage-aws
spec:
  storageClassName: omega-sc-aws-slow
  capacity:
    storage: 30Gi
  accessModes:
    - ReadWriteMany
  awsElasticBlockStore:
    # Volume ID is auto generated by AWS
    # $ aws ec2 create-volume --size 100
    # Returns the volume ID
    # with volume size of 100 Gi
    volumeID: vol-06e25d93b3b4e3302
    fsType: ext4
```

$ aws ec2 create-volume - -size 100

Volume ID is auto generated

Source: https://github.com/meta-magic/kubernetes_workshop

Persistent Volume – AWS EBS

- Manual Provisioning of the AWS EBS supports ReadWriteMany, However all the pods are getting scheduled into a Single Node.
- For Dynamic Provisioning use ReadWriteOnce.
- Google Compute Engine also doesn't support ReadWriteMany for dynamic provisioning.

Pod Access storage by issuing a Persistent Volume Claim.
In the following example Pod claims for 2Gi Disk space from the network on AWS EBS.

v-aws-pvc.yaml

2

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: omega-pvc-aws
  namespace: omega-aws
spec:
  storageClassName: omega-sc-aws-slow
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi

```

v-omega-dep.yaml

3

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: omega-aws-deploy
  namespace: omega-aws
spec:
  replicas: 3
  selector:
    matchLabels:
      desire3d.io/name: omega-pod-aws
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        desire3d.io/name: omega-pod-aws
        desire3d.io/version: "1.0.0"
        desire3d.io/release: stable
        desire3d.io/tier: fe
        desire3d.io/zone: prod
        desire3d.io/managed-by: m2
    spec:
      volumes:
        - name: omega-volume-aws
          persistentVolumeClaim:
           # Local Storage Claim
           claimName: omega-pvc-aws
      containers:
        - name: omega-ctr
          image: metamagic/oshell
          env:
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
          # Mount omega-fs from persistentVolumeClaim
            - mountPath: "/home/data"
              name: omega-volume-aws
```

https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes

Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Advanced features

- Jobs
- Daemon Set
- Container Level features
- Kubernetes Commands – Quick Help
- Kubernetes Commands – Field Selectors

Kubernetes Jobs

A *job* creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the *job* tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete. Deleting a Job will cleanup the pods it created.

A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first pod fails or is deleted (for example due to a node hardware failure or a node reboot).

A Job can also be used to run multiple pods in parallel.

app1-jobs.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
  namespace: sigma
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",
        "-Mbignum=bpi", "-wle",
        "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Command is wrapped for display purpose.

Source: https://github.com/meta-magic/kubernetes_workshop

Source: https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

Kubernetes DaemonSet

A *DaemonSet* ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a cluster storage daemon, such as glusterd, ceph, on each node.
- running a logs collection daemon on every node, such as fluentd or logstash.
- running a node monitoring daemon on every node, such as Prometheus Node Exporter, collectd, Dynatrace OneAgent, Datadog agent, New Relic agent, Ganglia gmond or Instana agent.

app1-daemon.yaml

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: k8s.gcr.io/fluentd-elasticsearch:1.20
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```


Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Container Level Features

Container-level features

Sidecar container: Although your Pod should still have a single main container, you can add a secondary container that acts as a helper (see a logging example). Two containers within a single Pod can communicate via a shared volume.

Init containers: *Init containers* run before any of a Pod's *app containers* (such as main and sidecar containers)

Source: https://kubernetes.io/docs/user-journeys/users/application-developer/advanced/

Kubernetes Commands – Quick Help

(Declarative Model)

Pods

```
$ kubectl create –f app-pod.yml
$ kubectl apply –f app-pod.yml
$ kubectl replace –f app-pod.yml
```

```
$ kubectl describe pods pod-name
$ kubectl get pods
$ kubectl get pods  -o json pod-name
$ kubectl get pods –show-labels
$ kubectl get pods –all-namespaces
```

```
$ kubectl exec –it pod-name sh
$ kubectl exec pod-name ps aux
```

ReplicaSet

```
$ kubectl create –f app-rs.yml
$ kubectl apply –f app-rs.yml
$ kubectl replace –f app-rs.yml
```

```
$ kubectl describe rs app-rs
$ kubectl get rs
$ kubectl get rs/app-rs
```

```
$ kubectl delete rs/app-rs cascade=false
```

Cascade=true will delete all the pods

 Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – Quick Help

(Declarative Model)

Service

```
$ kubectl create –f app-service.yml
$ kubectl apply –f app-service.yml
$ kubectl replace –f app-service.yml
```

```
$ kubectl get svc
$ kubectl describe svc app-service
$ kubectl get ep app-service
$ kubectl describe ep app-service
```

```
$ kubectl delete svc app-service
```

Deployment

```
$ kubectl create –f app-deploy.yml
$ kubectl apply –f app-deploy.yml
$ kubectl replace –f app-deploy.yml
```

```
$ kubectl get deploy app-deploy
$ kubectl describe deploy app-deploy
$ kubectl rollout status deployment app-deploy
$ kubectl rollout history deployment app-deploy
```

```
$ kubectl rollout undo deployment app-deploy  - -to-revision=1
```


Source: https://github.com/meta-magic/kubernetes_workshop

Kubernetes Commands – Field Selectors

Field selectors let you select Kubernetes resources based on the value of one or more resource fields. Here are some example field selector queries:

- metadata.name=my-service
- metadata.namespace!=default
- status.phase=Pending

```
$ kubectl get pods --field-selector status.phase=Running
```

Get the list of pods where status.phase = Running

Supported Operators

You can use the =, ==, and != operators with field selectors (= and == mean the same thing). This kubectl command, for example, selects all Kubernetes Services that aren't in the default namespace:

```
$ kubectl get services --field-selector metadata.namespace!=default
```

Source: https://kubernetes.io/docs/concepts/overview/working-with-objects/field-selectors/

Kubernetes Commands – Field Selectors

Chained Selectors

As with label and other selectors, field selectors can be chained together as a comma-separated list. This kubectl command selects all Pods for which the status.phase does not equal Running and the spec.restartPolicy field equals Always:

```
$ kubectl get pods --field-selector=status.phase!=Running,spec.restartPolicy=Always
```

Multiple Resource Type

You use field selectors across multiple resource types. This kubectl command selects all Statefulsets and Services that are not in the default namespace:

```
$ kubectl get statefulsets,services --field-selector metadata.namespace!=default
```

Source: https://kubernetes.io/docs/concepts/overview/working-with-objects/field-selectors/

Service Mesh: Istio

Gateway

Virtual Service

Destination Rule

Istio Components

Data Plane

Control Plane

Envoy

Envoy is deployed as a Sidecar in the same K8S Pod.

- Dynamic Service Discovery
- Load Balancing
- TLS Termination
- HTTP/2 and gRPC Proxies
- Circuit Breakers
- Health Checks
- Staged Rollouts with % based traffic split
- Fault Injection
- Rich Metrics

Mixer

- Enforces access control and usage policies across service mesh and
- Collects telemetry data from Envoy and other services.
- Also includes a flexible plugin model.

Pilot

Provides

- Service Discovery
- Traffic Management
- Routing
- Resiliency (Timeouts, Circuit Breakers, etc.)

Galley

Provides

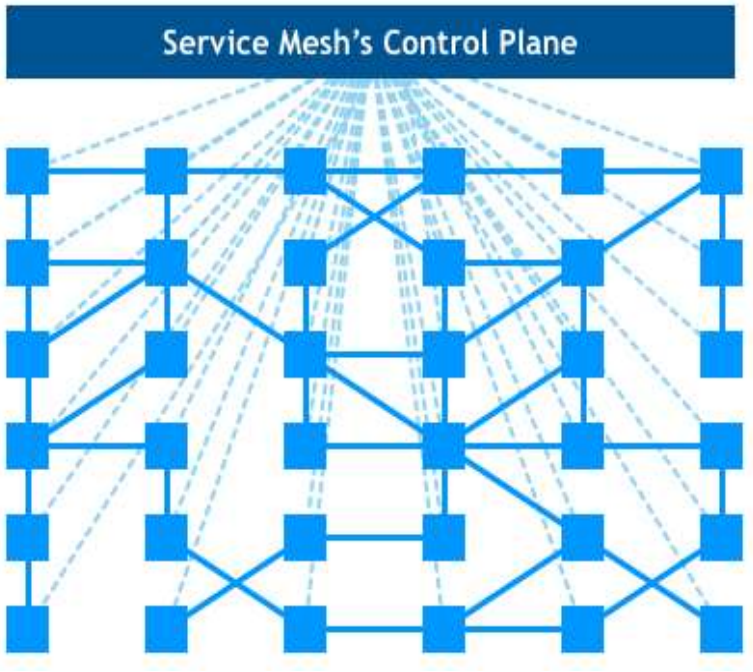
- Configuration Injection
- Processing and
- Distribution Component of Istio

Citadel

Provides

- Strong Service to Service and end user Authentication with built-in Identity and credential management.
- Can enforce policies based on Service identity rather than network controls.

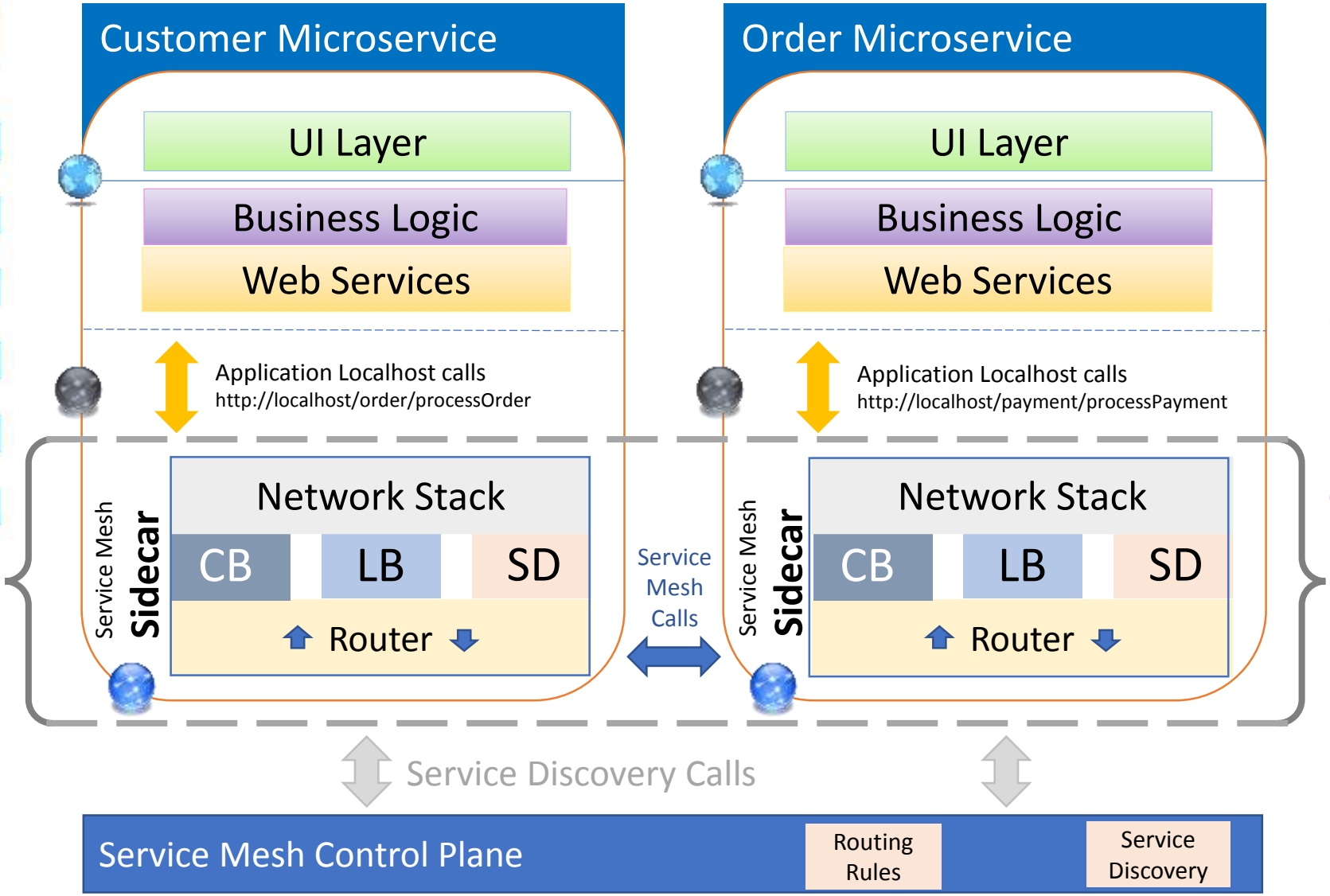
Service Mesh – Sidecar Design Pattern



Microservice

CB – Circuit Breaker
LB – Load Balancer
SD – Service Discovery

Data Plane



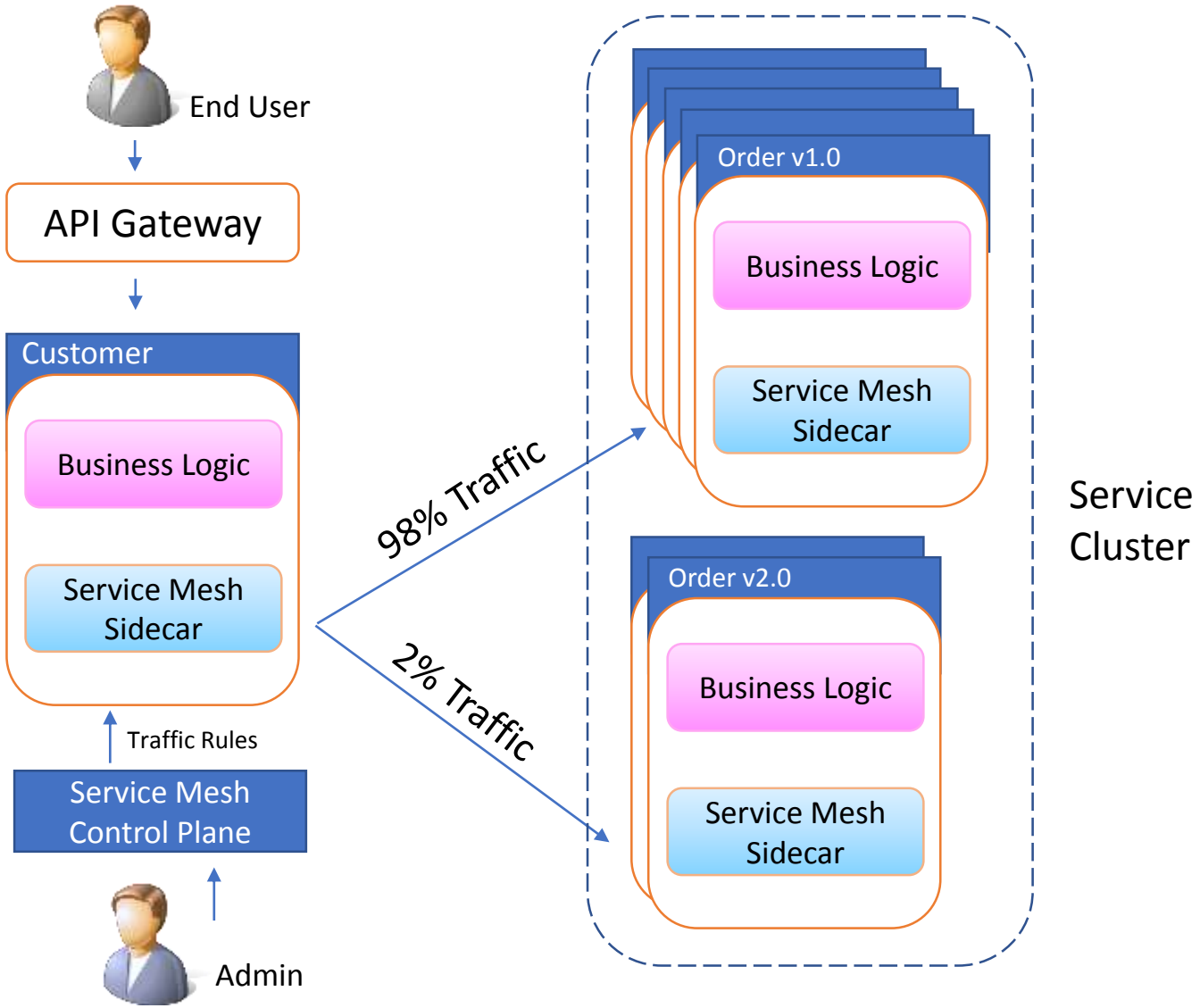
Control Plane will have all the rules for Routing and Service Discovery. Local Service Mesh will download the rules from the Control pane will have a local copy.

Source: https://github.com/meta-magic/kubernetes_workshop

Service Mesh – Traffic Control

Traffic Control rules can be applied for

- different Microservices versions
- Re Routing the request to debugging system to analyze the problem in real time.
- Smooth migration path



Source: https://github.com/meta-magic/kubernetes_workshop

Why Service Mesh?

- Multi Language / Technology stack Microservices requires a standard telemetry service.
- Adding SSL certificates across all the services.
- Abstracting Horizontal concerns

- Stakeholders: Identify whose affected.
- Incentives: What Service Mesh brings onto the table.
- Concerns: Their worries
- Mitigate Concerns

Source: https://github.com/meta-magic/kubernetes_workshop

Istio Sidecar Automatic Injection

shopping-ns.yaml

```
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4      name: shoppingportal
5      labels:
6          name: shoppingportal
7          istio-injection: enabled
8
```


Source: https://github.com/meta-magic/kubernetes_workshop

Istio – Traffic Management

Configures a load balancer for HTTP/TCP traffic, most commonly operating at the edge of the mesh to enable ingress traffic for an application.

Gateway

Defines the rules that control how requests for a service are routed within an Istio service mesh.

Virtual Service

Routing Rules

- Match
  - URI Patterns
  - URI ReWrites
  - Headers
  - Routes
  - Fault
- Fault
- Route
- Weightages

Configures the set of policies to be applied to a request after Virtual Service routing has occurred.

Destination Rule

Policies

- Traffic Policies
  - Load Balancer

Source: https://github.com/meta-magic/kubernetes_workshop

Istio Gateway

Configures a load balancer for HTTP/TCP traffic, most commonly operating at the edge of the mesh to enable ingress traffic for an application.

shoppingportal-gw.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: shoppingportal-gateway
  namespace: shoppingportal
spec:
  selector:
    istio: ingressgateway # use istio defa
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
     - "*"
```


Source: https://github.com/meta-magic/kubernetes_workshop

Istio Virtual Service

Defines the rules that control how requests for a service are routed within an Istio service mesh.

shoppingportal-virtualservice.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: shoppingportal-vs
  namespace: shoppingportal
spec:
  gateways:
  - shoppingportal-gateway
  hosts:
   - "*"
  http:
  - match:
    - uri:
        prefix: /ui
      headers:
        user-agent:
          regex: ".*Firefox.*"
    route:
    - destination:
        host: k8uiworkshopservice.shoppingportal.svc.cluster.local
        subset: canary # match v2 only
        port:
          number: 80
  - match:
    - uri:
        prefix: /ui
    route:
    - destination:
        host: k8uiworkshopservice.shoppingportal.svc.cluster.local
        subset: stable # match v2 only
        port:
          number: 80
  - match:
    - uri:
        prefix: /productms
      headers:
        end-user:
          exact: metamagic
    route:
    - destination:
        host: productservice.shoppingportal.svc.cluster.local
        subset: canary # match v2 only
        port:
          number: 80
  - match:
    - uri:
        prefix: /productms
    route:
    - destination:
        host: productservice.shoppingportal.svc.cluster.local
        subset: stable # match v2 only
        port:
          number: 80
  - match:
    - uri:
        prefix: /productreviewms
    route:
    - destination:
        host: productreviewservice.shoppingportal.svc.cluster.local
        port:
          number: 80
```


Source: https://github.com/meta-magic/kubernetes_workshop

Istio Destination Rule

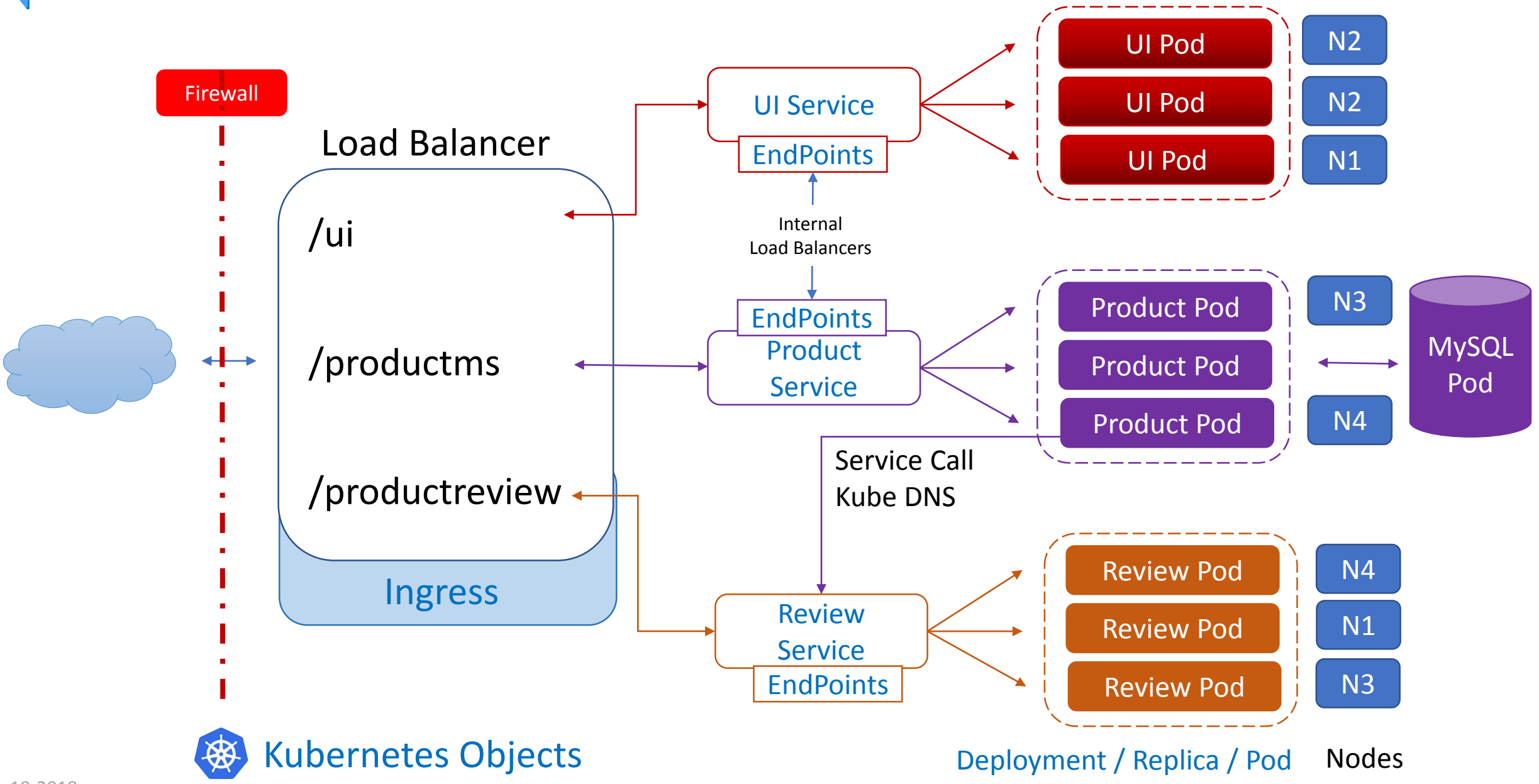
Configures the set of policies to be applied to a request after Virtual Service routing has occurred.

product-destination.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product-destination-rules
  namespace: shoppingportal
spec:
  host: productservice.shoppingportal.svc.cluster.local
  subsets:
  - name: stable
    labels:
      version: v1
  - name: canary
    labels:
      version: v2
```

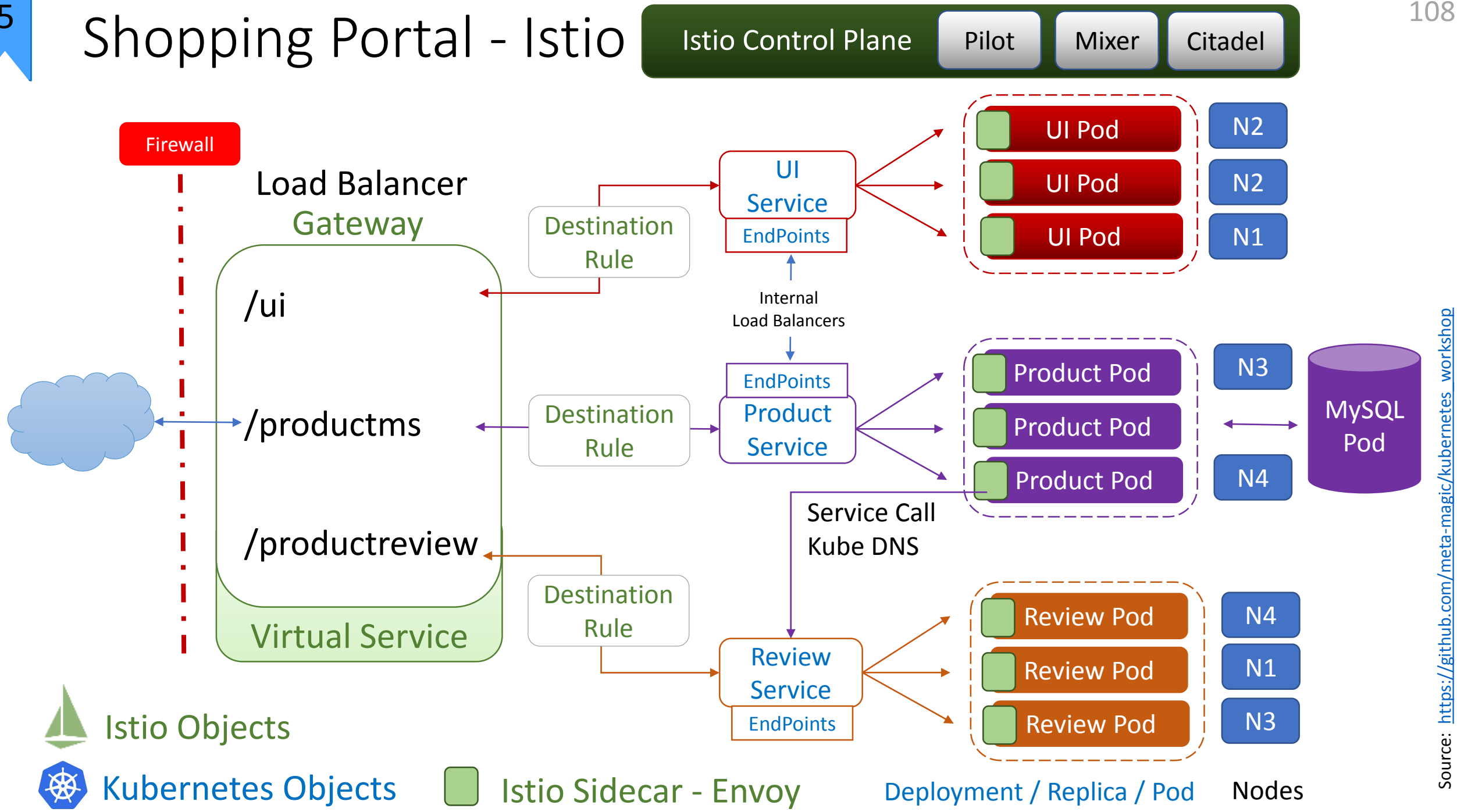
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal – Docker / Kubernetes



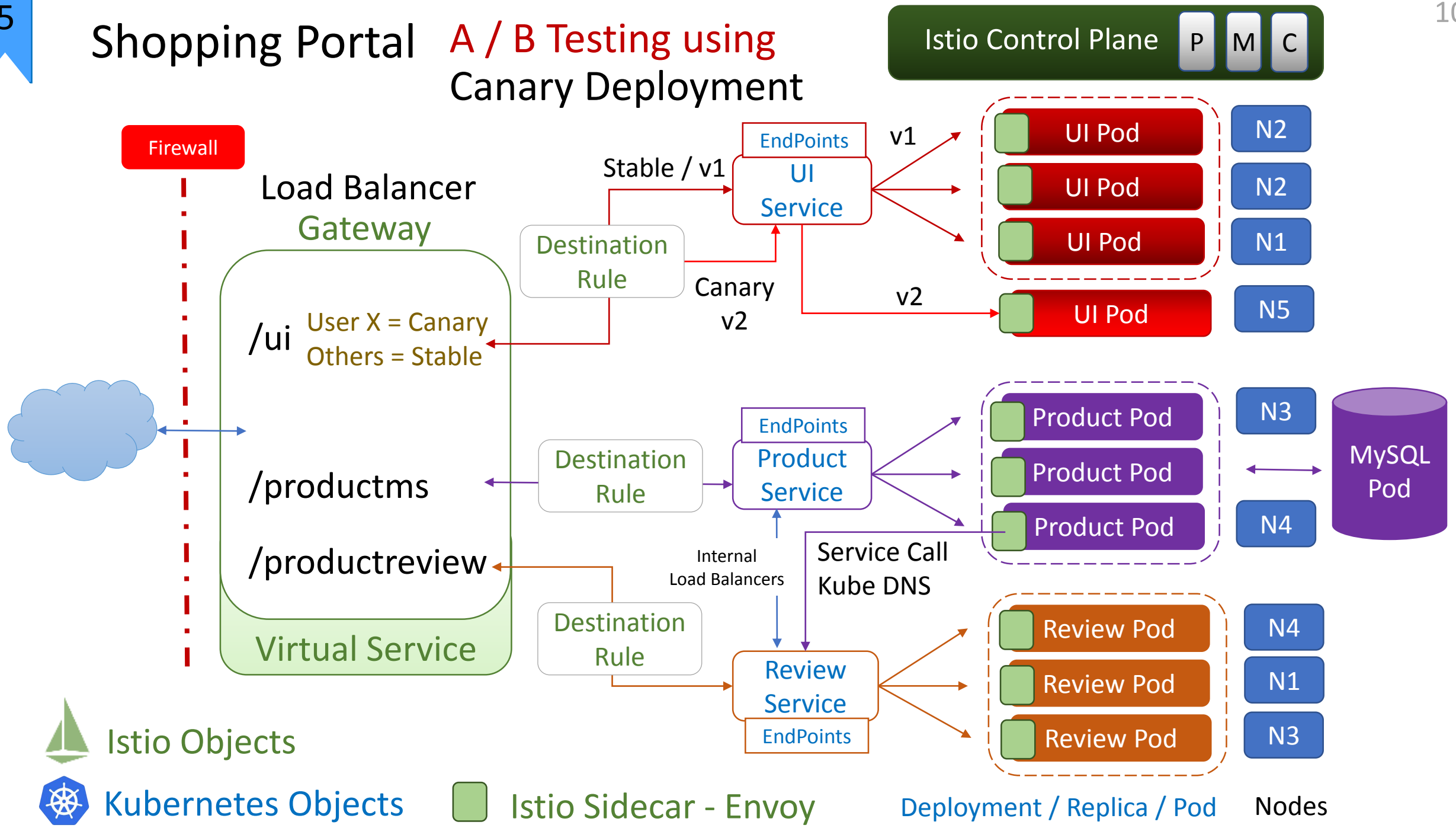
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal - Istio



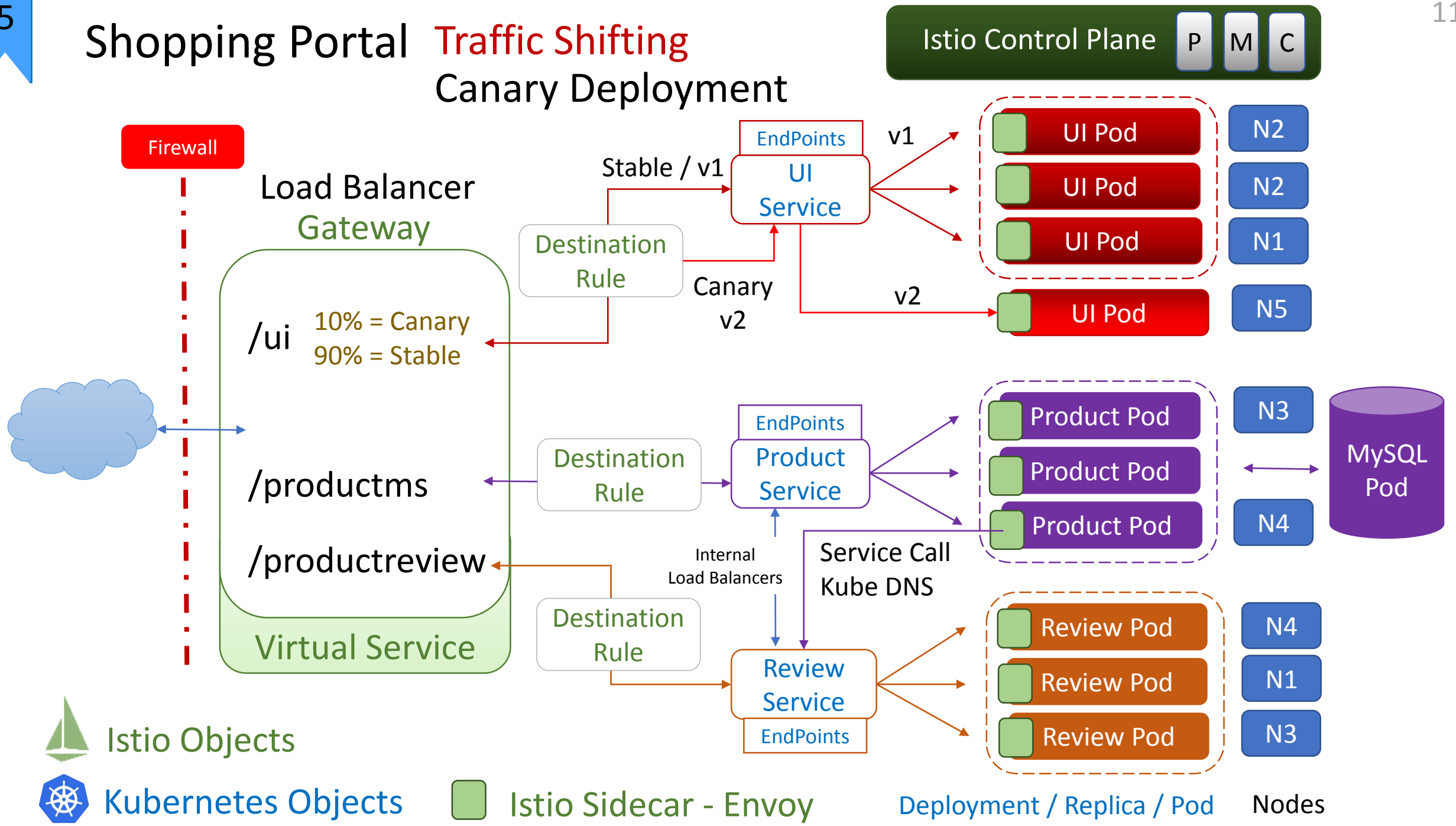
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal A / B Testing using Canary Deployment



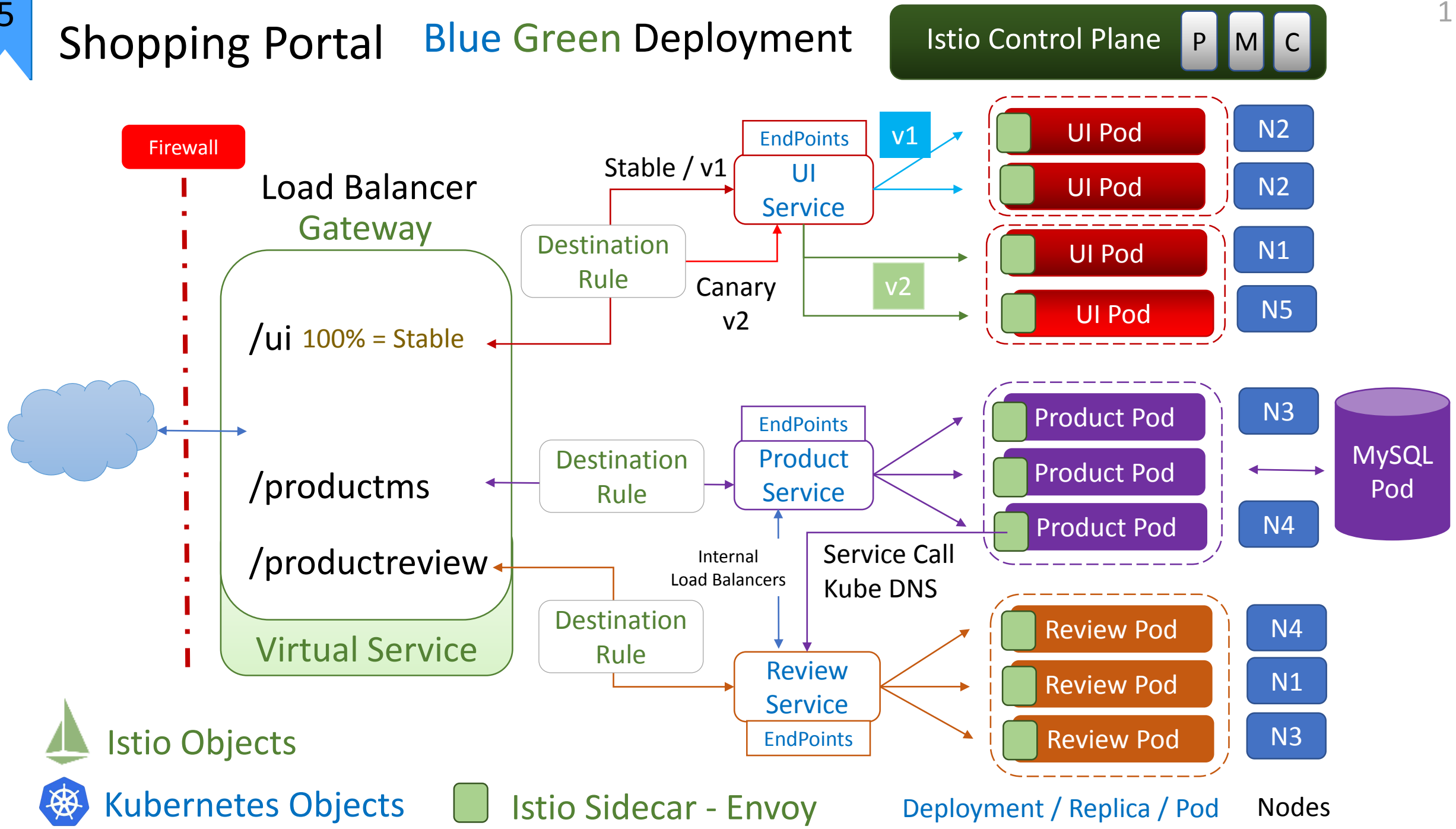
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal Traffic Shifting
Canary Deployment



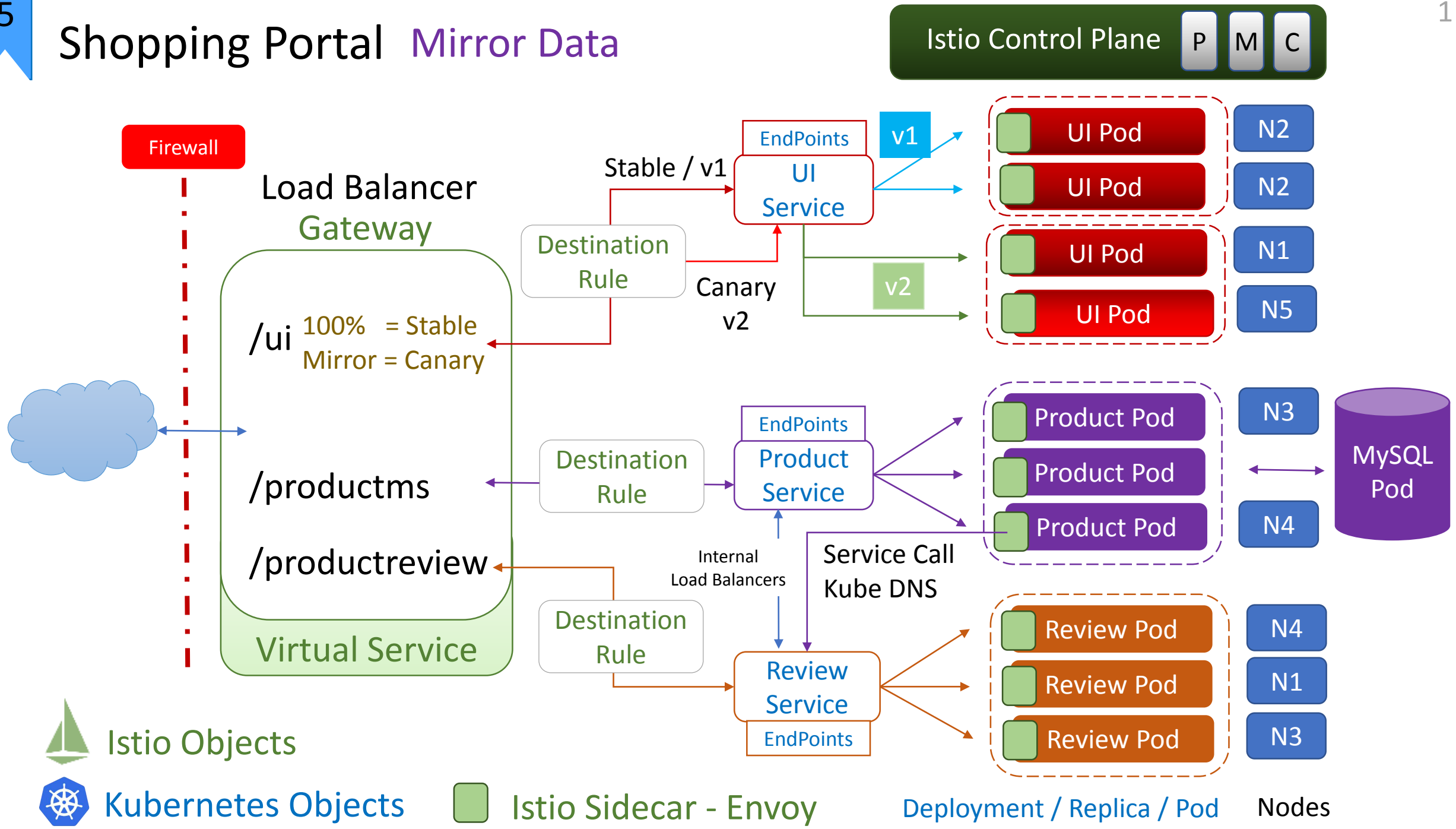
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal Blue Green Deployment



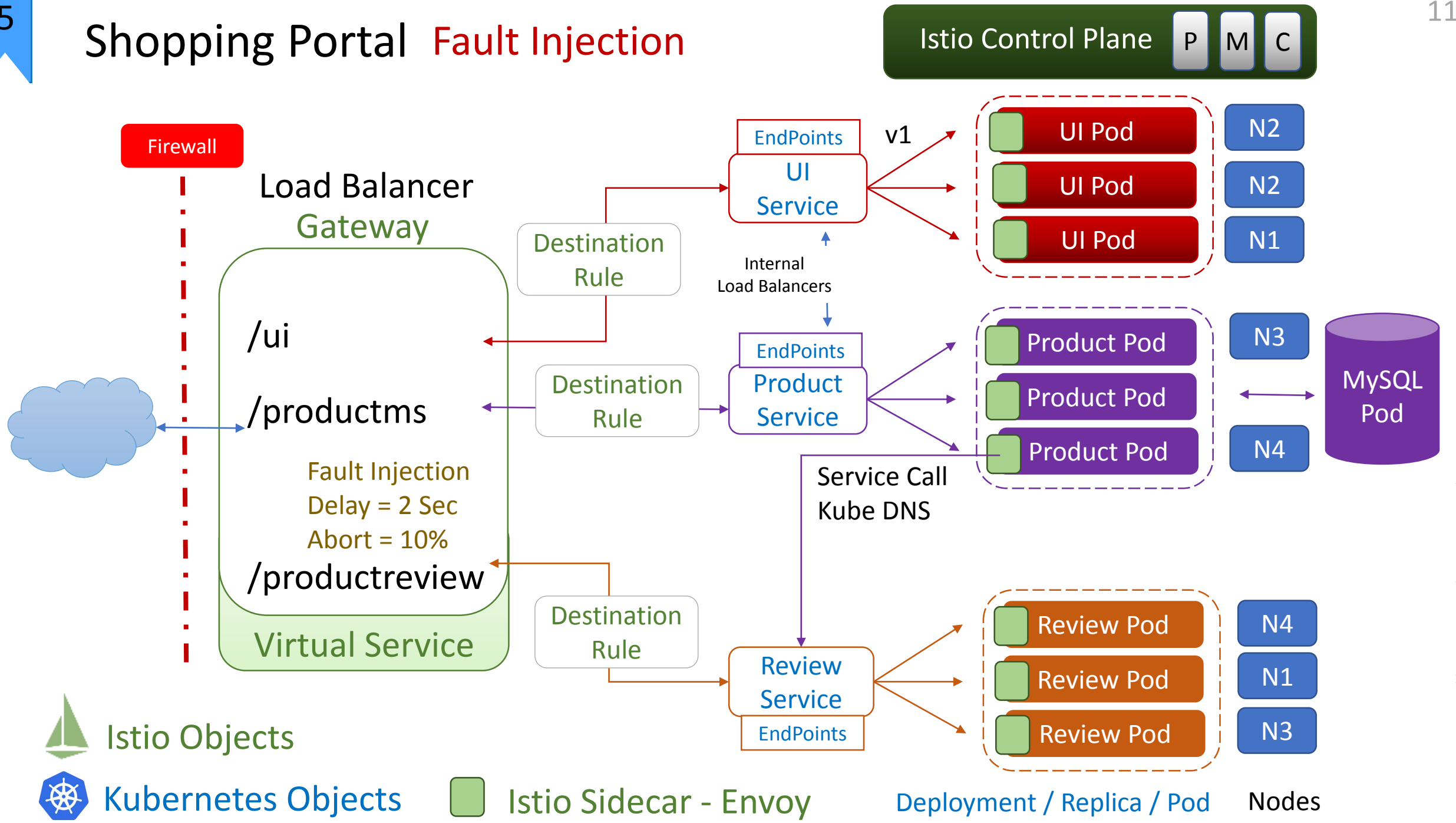
Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal Mirror Data



Source: https://github.com/meta-magic/kubernetes_workshop

Shopping Portal Fault Injection



Source: https://github.com/meta-magic/kubernetes_workshop

Amazon AWS

- Virtual Private Network / Subnets
- Internet Gateway
- Routes

Create VPC & Subnet

```
$ aws ec2 create-vpc --cidr-block 10.0.0.0/16
{
   "Vpc": {
     "VpcId": "vpc-7532a92g",
      "InstanceTenancy": "default",
      "Tags": [],
      "State": "pending",
      "DhcpOptionsId": "dopt-3d901958",
      "CidrBlock": "10.0.0.0/16"
    }
}
```

When you create a VPC, just define

- one network CIDR block and
- AWS region.
- For example, CIDR 10.0.0.0/16 on us-east-1.

You can define any network address range (between /16 to /28 netmask range).

Create one or more subnets within VPC.

```
$ aws ec2 create-subnet --vpc-id 7532a92g", --cidr-block 10.0.1.0/24 -- availability-zone us-east-1a

{ "Subnet": { "VpcId": "vpc- 7532a92g",  ", "CidrBlock": "10.0.1.0/24", "State": "pending",
"AvailabilityZone": "us-east-1a", "SubnetId": "subnet-f92x9g72", "AvailableIpAddressCount": 251 } }

$ aws ec2 create-subnet --vpc-id vpc- 7532a92g --cidr-block 10.0.2.0/24 -- availability-zone us-east-1b

{ "Subnet": { "VpcId": " vpc- 7532a92g ", "CidrBlock": "10.0.2.0/24", "State": "pending", "AvailabilityZone":
"us-east-1b", "SubnetId": "subnet-16938e09", "AvailableIpAddressCount": 251 } }
```

Create Gateway and Attach it

```
$ aws ec2 create-internet-gateway
{
  "InternetGateway": {
   "Tags": [],
   "InternetGatewayId": "igw-b837249v1",
    “Attachments": []
   }
}
```

You need to have a Internet Gateway for your VPC to connect to the internet.

Create an Internet Gateway and attach that to the VPC.

Set the routing rules for the subnet to point to the gateway.

Attach VPC to the Gateway

```
$ aws ec2 attach-internet-gateway --vpc-id vpc-7532a92g --internet-gateway- id igw-b837249v1
```

Create Route table for the VPC

```
$ aws ec2 create-route-table --vpc-id vpc-7532a92g
```

Create Routes

```
$ aws ec2 create-route-table --vpc-id vpc-7532a92g
{ "RouteTable":
   { "Associations": [],
     "RouteTableId": "rtb-ag89x582",
     "VpcId": "vpc-7532a92g",
     "PropagatingVgws": [],
     "Tags": [], "Routes": [
            { "GatewayId": "local",
              "DestinationCidrBlock": "10.0.0.0/16",
               "State": "active",
                "Origin": "CreateRouteTable"
             }
        ]
}}
```

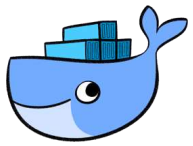
Create Route table for the VPC

```
$ aws ec2 create-route --route-table-id rtb-ag89x582 --gateway-id igw-b837249v1 --destination-cidr-block 0.0.0.0/0
```

Best Practices

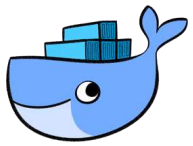
Docker Best Practices

Kubernetes Best Practices



Build Small Container Images

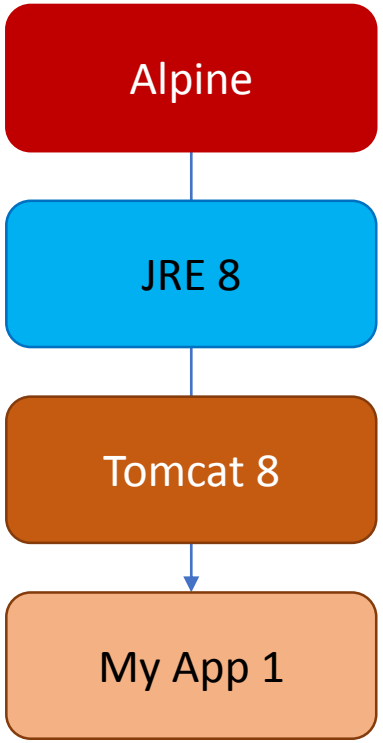
- Simple Java Web Apps with Ubuntu & Tomcat can have a size of 700 MB
- Use Alpine Image as your base Linux OS
- Alpine images are 10x smaller than base Ubuntu images
- Smaller Image size reduce the Container vulnerabilities.
- Ensure that only Runtime Environments are there in your container. For Example your Alpine + Java + Tomcat image should contain only the JRE and NOT JDK.
- Log the App output to Container Std out and Std error.

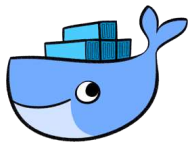


Docker: To Root or Not to Root!

2

- Create Multiple layers of Images
- Create a User account
- Add Runtime software's based on the User Account.
- Run the App under the user account
- This gives added security to the container.
- Add Security module SELinux or AppArmour to increase the security,





Docker: Container Security

1. Secure your HOST OS! Containers runs on Host Kernel.

2. No Runtime software downloads inside the container. Declare the software requirements at the build time itself.

3. Download Docker base images from Authentic site.

4. Limit the resource utilization using Container orchestrators like Kubernetes.

5. Don’t run anything on Super privileged mode.

Kubernetes: Naked Pods

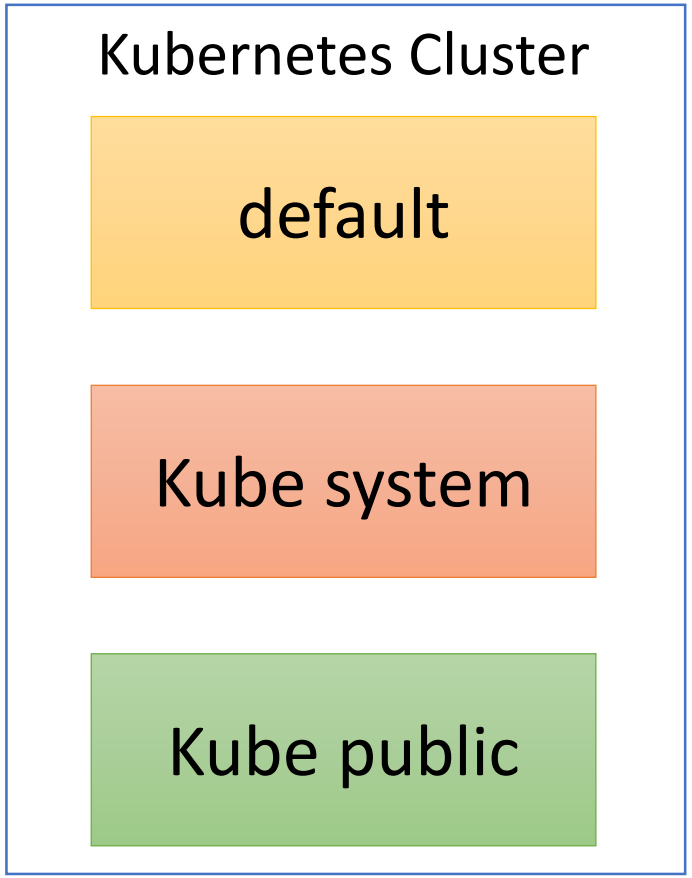
- Never use a Naked Pod, that is Pod without any ReplicaSet or Deployments. Naked pods will never get re-scheduled if the Pod goes down.
- Never access a Pod directly from another Pod. Always use a Service to access a Pod.
- User labels to select the pods { app: myapp, tier: frontend, phase: test, deployment: v3 }.
- Never use :latest tag in the image in the production scenario.

Kubernetes: Namespace

5

Service-Name.Namespace.svc.cluster.local

- Group your Services / Pods / Traffic Rules based on Specific Namespace.
- This helps you apply specific Network Policies for that Namespace with increase in Security and Performance.
- Handle specific Resource Allocations for a Namespace.
- If you have more than a dozen Microservices then it's time to bring in Namespaces.




```
$ kubectl config set-context $(kubectl config current-context) --namespace=your-ns
```

The above command will let you switch the namespace to your namespace (your-ns).

Kubernetes: Pod Health Check

- Pod Health check is critical to increase the overall resiliency of the network.
- Readiness
- Liveness
- Ensure that all your Pods have Readiness and Liveness Probes.
- Choose the Protocol wisely (HTTP, Command & TCP)

Kubernetes: Resource Utilization

- For the Best Quality define the requests and limits for your Pods.
- You can set specific resource requests for a Dev Namespace to ensure that developers don’t create pods with a very large resource or a very small resource.
- Limit Range can be set to ensure that containers were create with too low resource or too large resource.

Kubernetes: Pod Termination Lifecycle

- Make sure that the Application to Handle SIGTERM message.
- You can use preStop Hook
- Set the terminationGracePeriodSeconds: 60
- Ensure that you clean up the connections or any other artefacts and ready for clean shutdown of the App (Microservice).
- If the Container is still running after the grace period, Kubernetes sends a SIGKILL event to shutdown the Pod.

Kubernetes: External Services

- There are systems that can be outside the Kubernetes cluster like
  - Databases or
  - external services in the cloud.
- You can create an Endpoint with Specific IP Address and Port with the same name as Service.
- You can create a Service with an External Name (URL) which does a CNAME redirection at the Kernel level.

Kubernetes: Upgrade Cluster

- Make sure that the Master behind a Load Balancer.
- Upgrade Master
  - Scale up the Node with an extra Node
  - Drain the Node and
  - Upgrade Node
- Cluster will be running even if the master is not working. Only Kubectl and any master specific functions will be down until the master is up.

Source: https://github.com/meta-magic/kubernetes_workshop

Thank you

Araf Karsh Hamid : Co-Founder / CTO
araf.karsh@metamagic.in
USA: +1 (973) 969-2921
India: +91.999.545.8627
Skype / LinkedIn / Twitter / Slideshare : arafkarsh
http://www.slideshare.net/arafkarsh
https://www.linkedin.com/in/arafkarsh/
http://www.arafkarsh.com/