

Docker / Kubernetes / Istio

Containers

Container Orchestration

Service Mesh

1

Docker



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

3

Infrastructure Design Patterns



- API Gateway
- Load Balancer
- Service Discovery
- Circuit Breaker
- Service Aggregator
- Let-it Crash Pattern

2

Kubernetes



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

4

Kubernetes Networking



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

5

Istio



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

6

Istio – Security and RBAC



- Security
- RBAC

7

Kubernetes Advanced Concepts



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

8

Logging & Monitoring



- Logging
- Distributed Tracing
- Grafana / Prometheus

9

Best Practices



- Docker Best Practices
- Kubernetes Best Practices

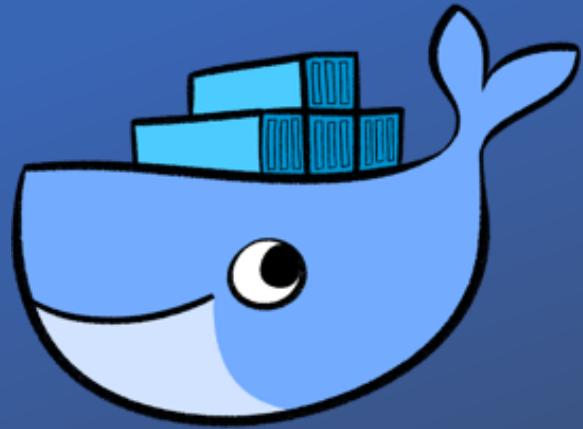
12 Factor App Methodology

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

High Level Objectives

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

Walks like a

Runs like a

Virtual
Machine

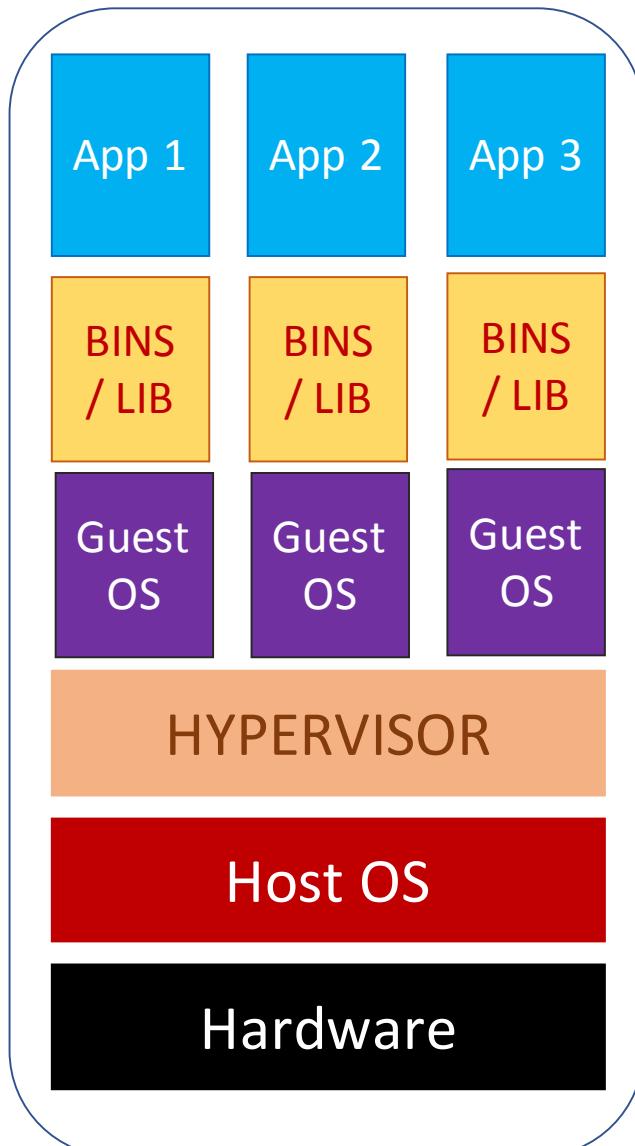


Containers are a **Sandbox** inside Linux Kernel **sharing the kernel** with **separate** Network Stack, Process Stack, IPC Stack etc.

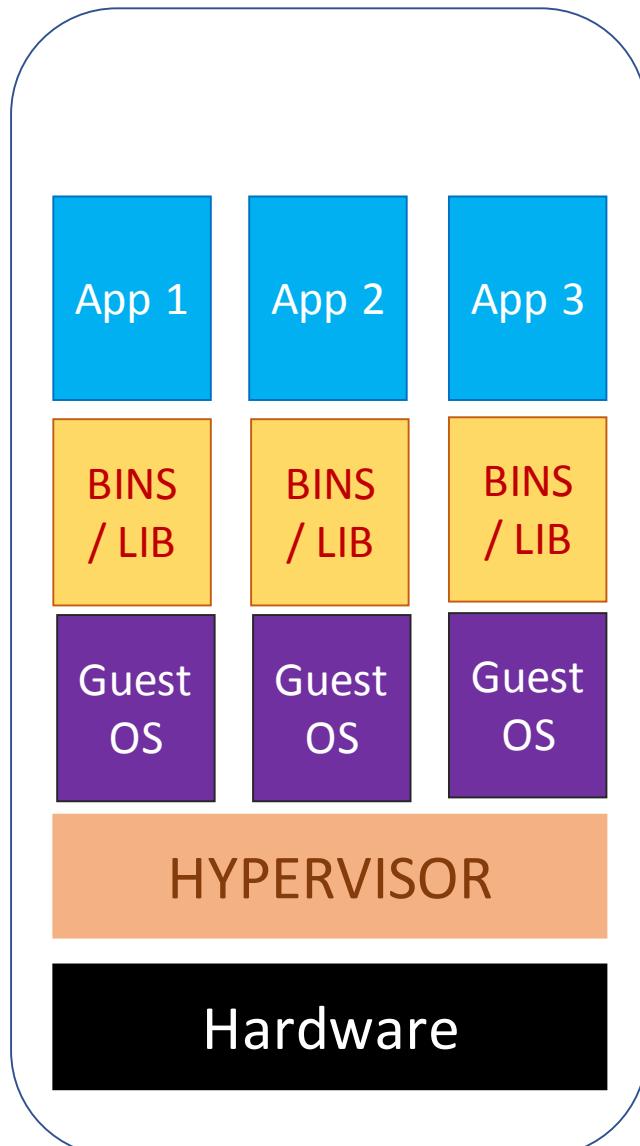
Servers / Virtual Machines / Containers



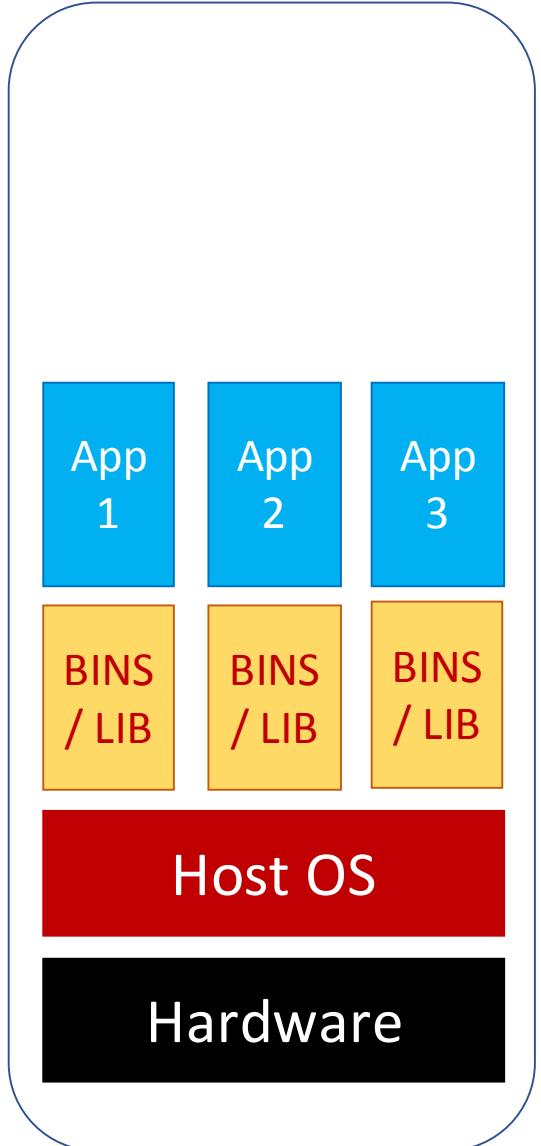
Server



Type 1 Hypervisor

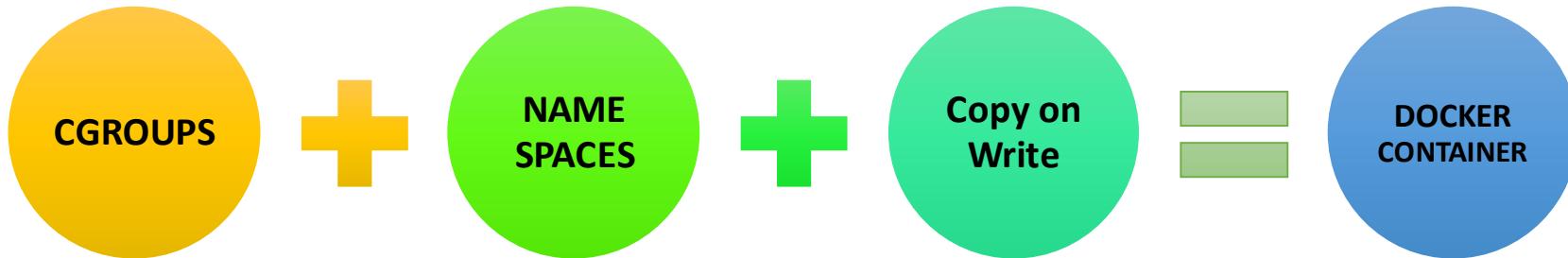


Type 2 Hypervisor



Container

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

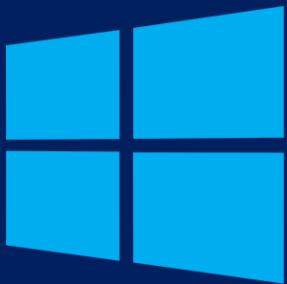
Docker Container – Linux and Windows



Control Groups
cgroups

Namespaces
Pid, net, ipc, mnt, uts

Layer Capabilities
Union File Systems:
AUFS, btrfs, vfs



Control Groups
Job Objects

Namespaces
Object Namespace, Process
Table, Networking

Layer Capabilities
Registry, UFS like
extensions

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.

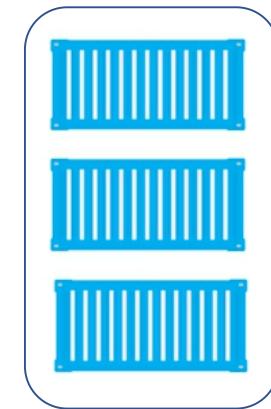
- Docker containers
 - Docker containers are similar to a directory.
 - A Docker container holds everything that is needed for an application to run.
 - Each container is created from a Docker image.
 - Docker containers can be run, started, stopped, moved, and deleted.
 - Each container is an isolated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

- 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.

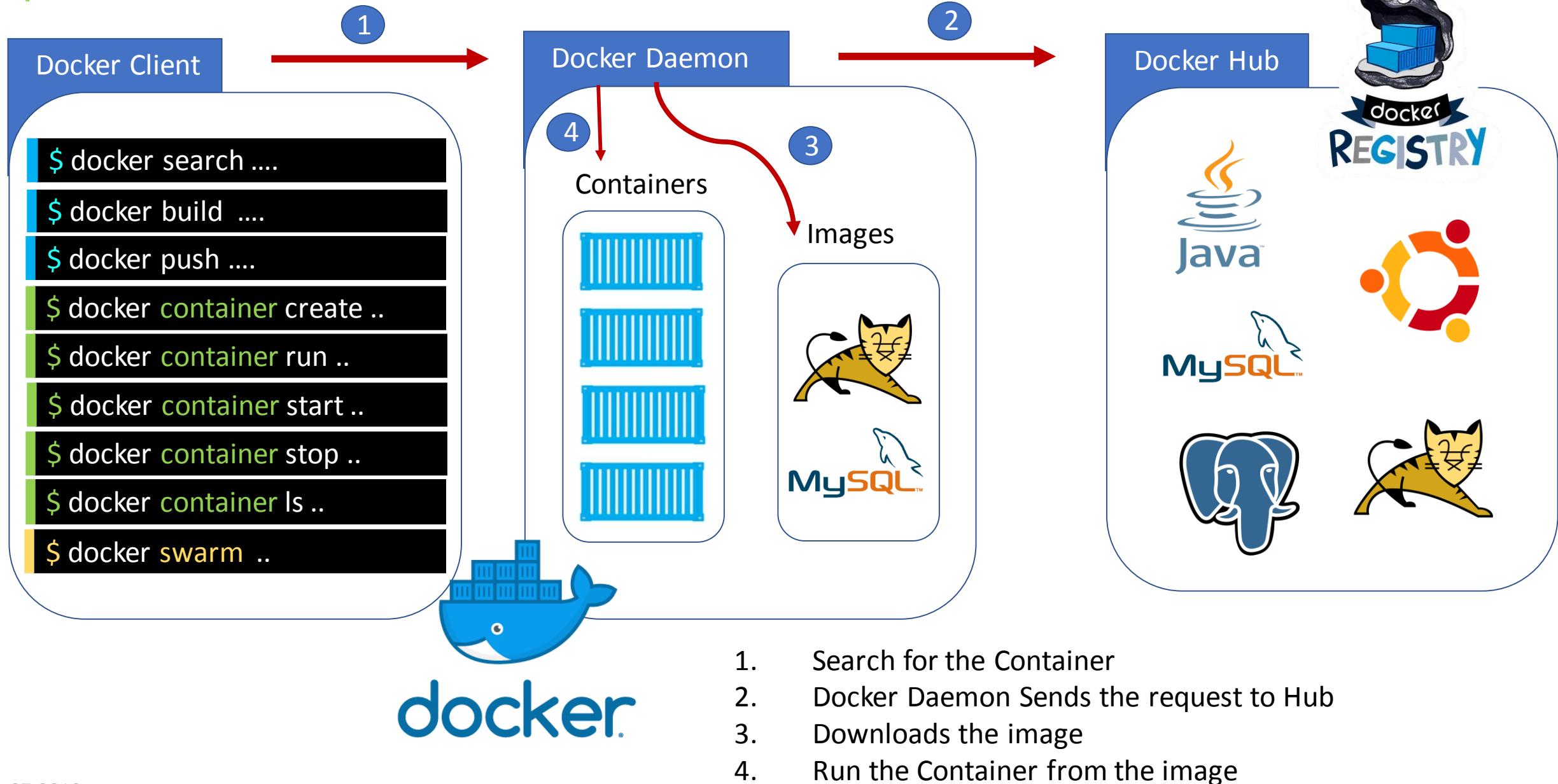
Images



Containers



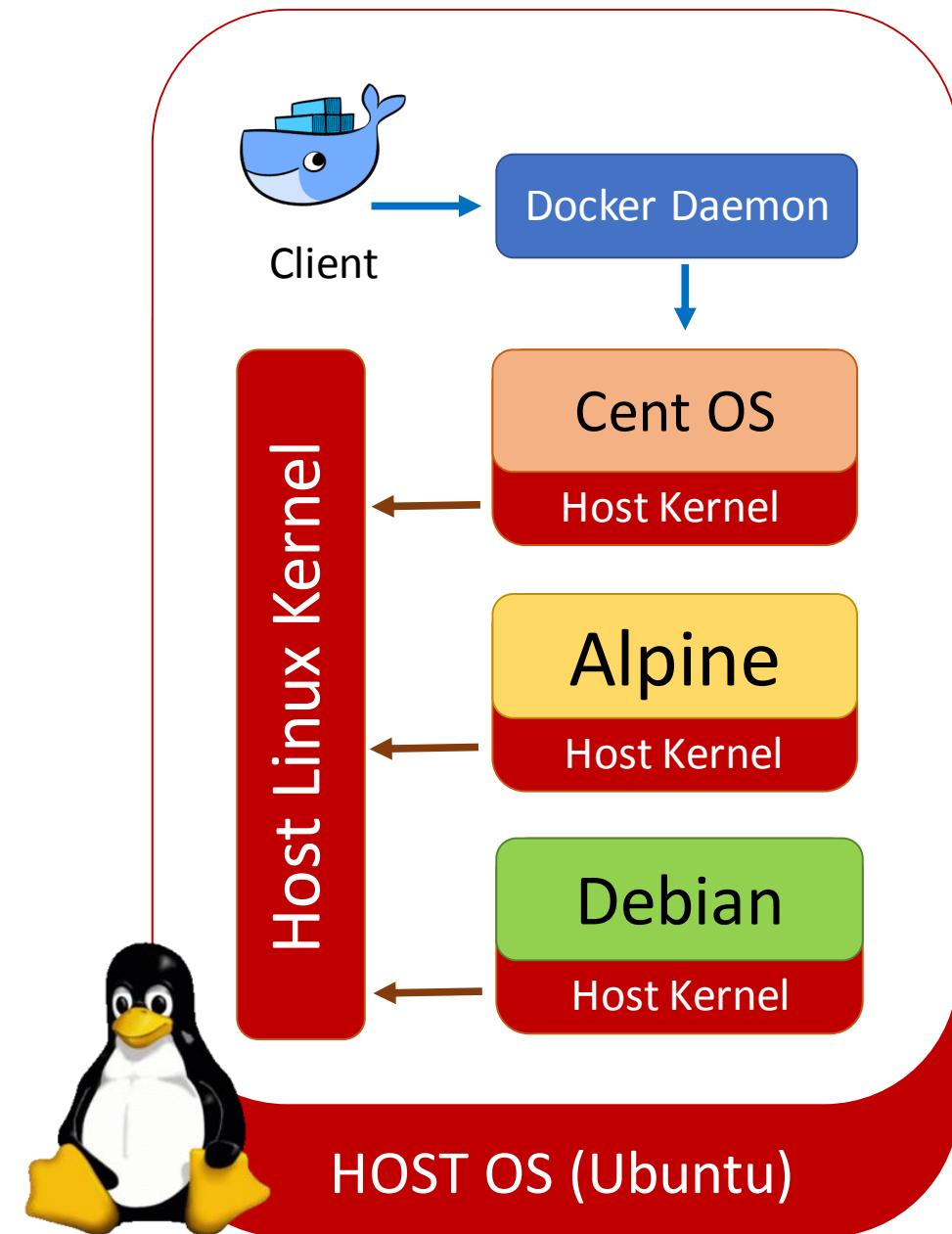
How Docker works....



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

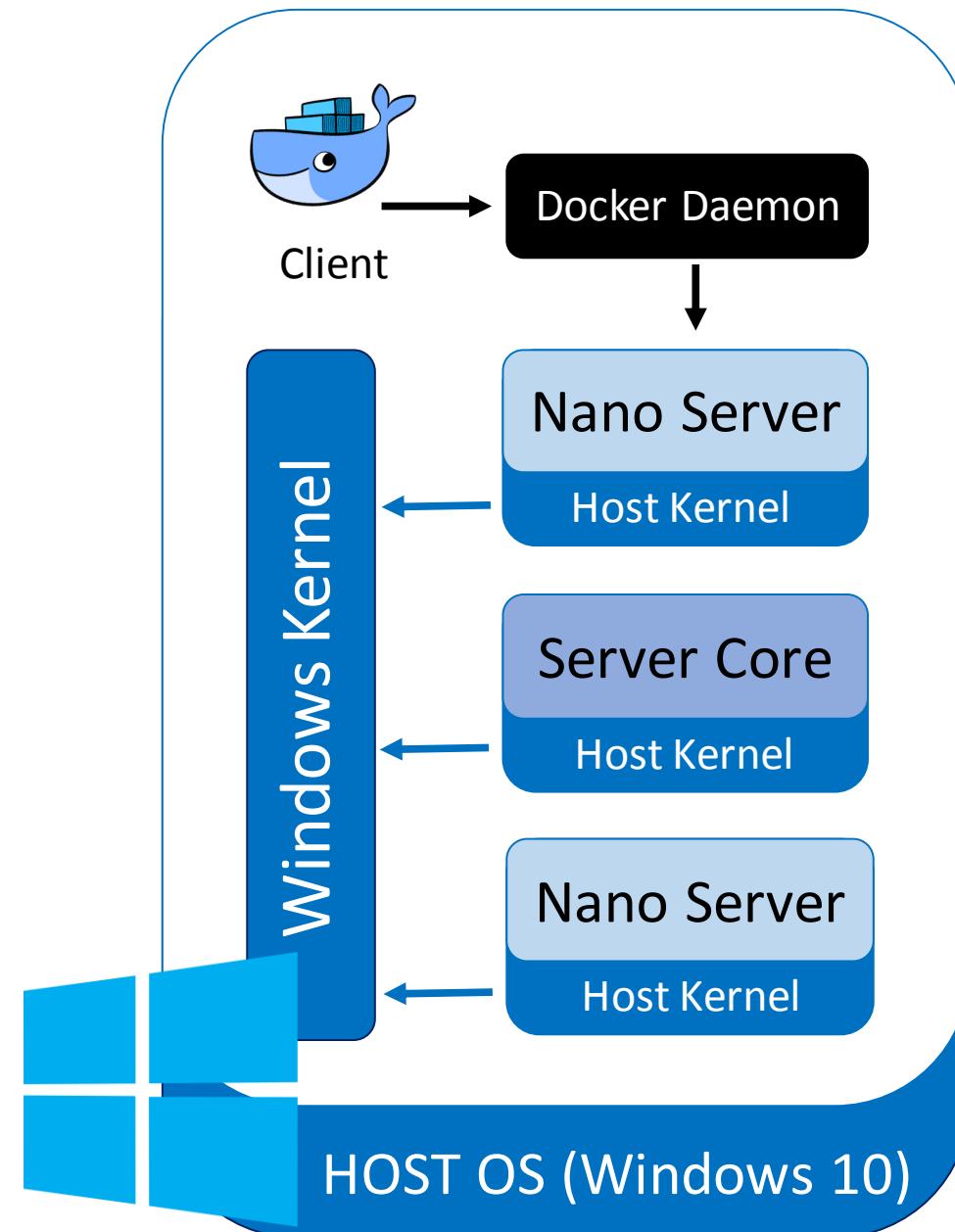
Linux Kernel

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

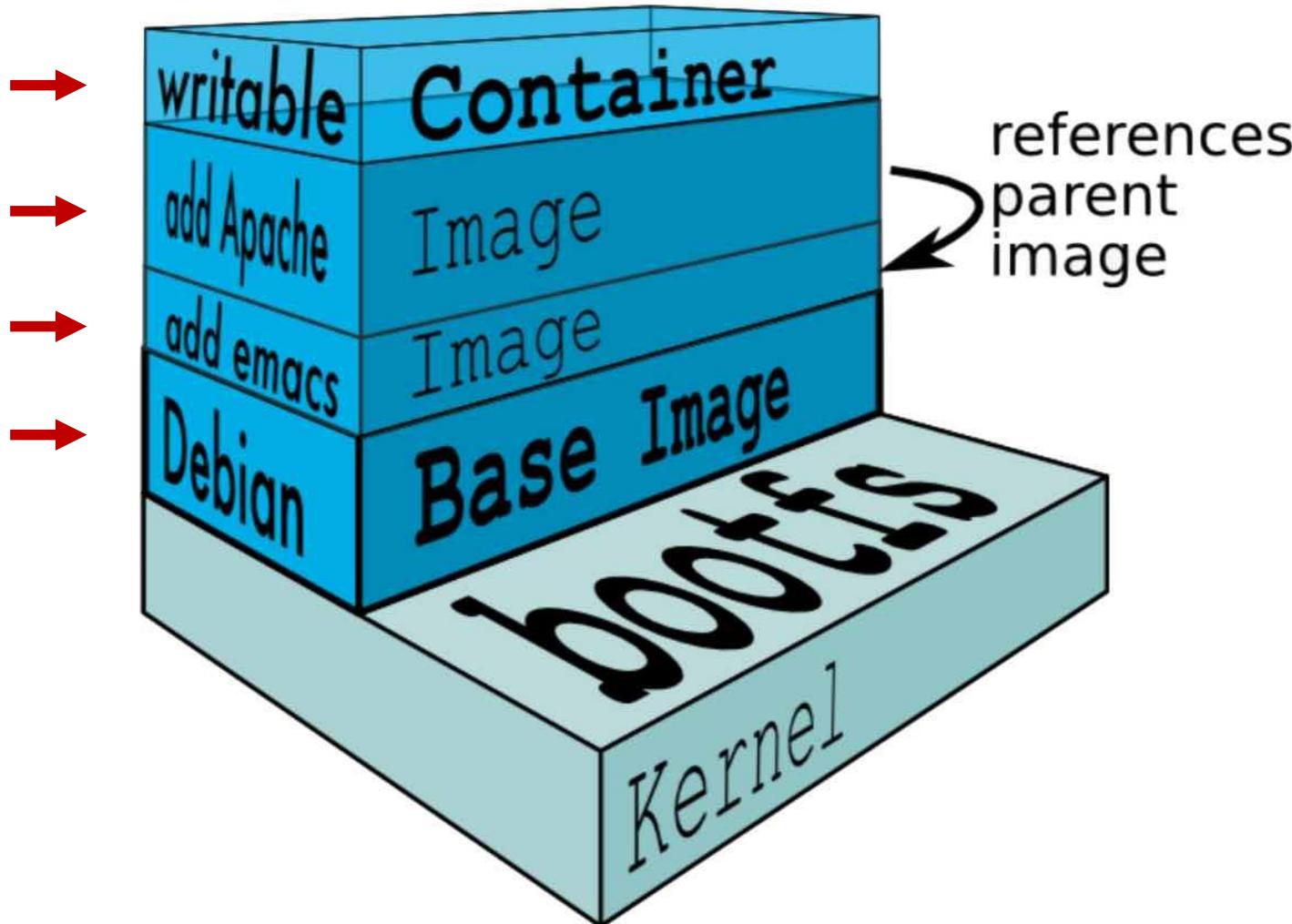


Windows Kernel

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

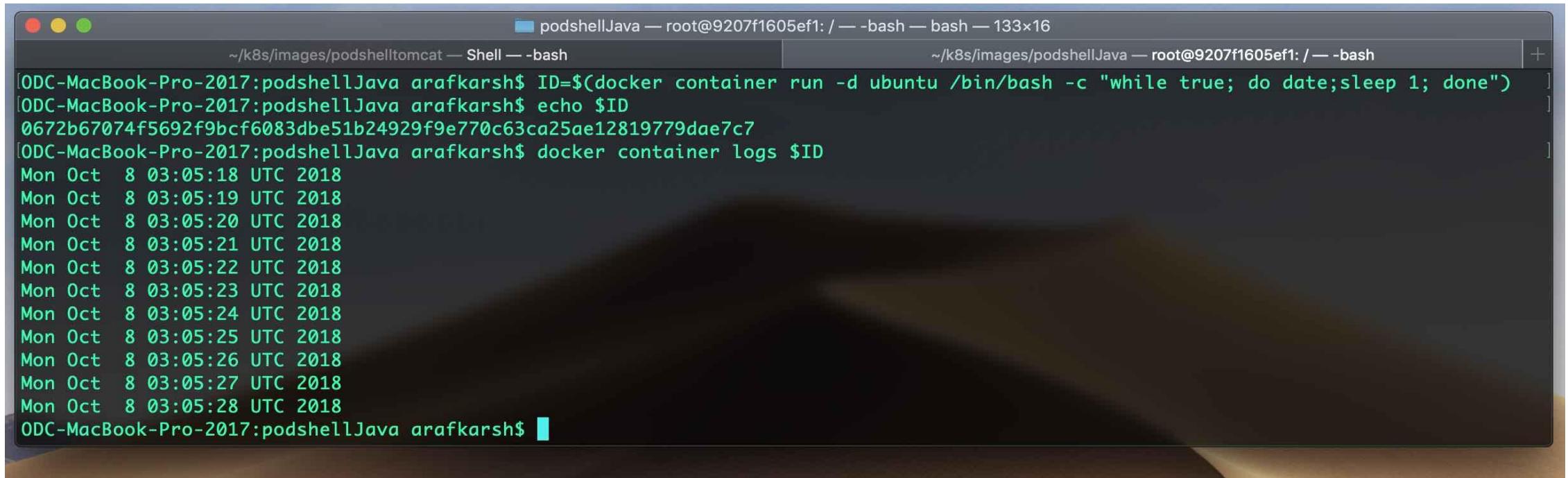
Running a Docker Container

```
$ 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")
```



The screenshot shows a terminal window with two tabs. The left tab is titled 'podshellJava' and shows the command: 'ID=\$(docker container run -d ubuntu /bin/bash -c "while true; do date; sleep 1; done")'. The right tab is also titled 'podshellJava' and shows the output of the command: 'echo \$ID' followed by a long string of characters. Below these tabs, the terminal displays the output of the 'date' command every second, starting from 'Mon Oct 8 03:05:18 UTC 2018' and continuing sequentially up to 'Mon Oct 8 03:05:28 UTC 2018'. The terminal window has a dark background and light-colored text.

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

Shows output from the(bash script) container

```
$ docker container ls
```

List the running Containers

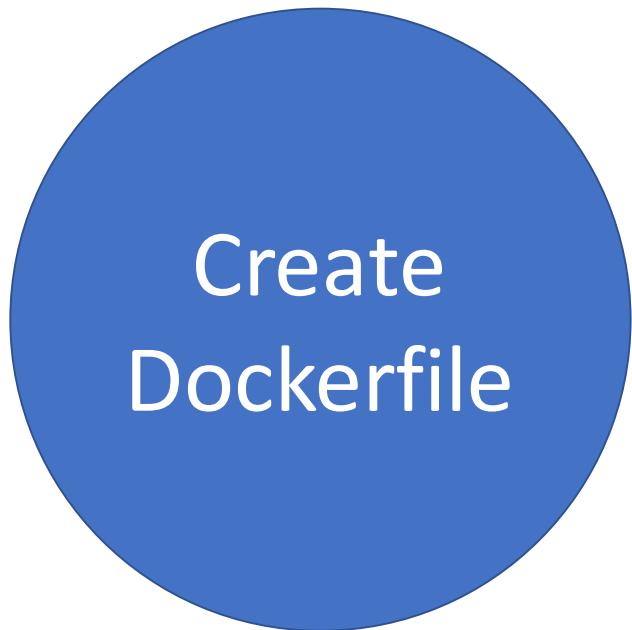
Anatomy of a Dockerfile

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

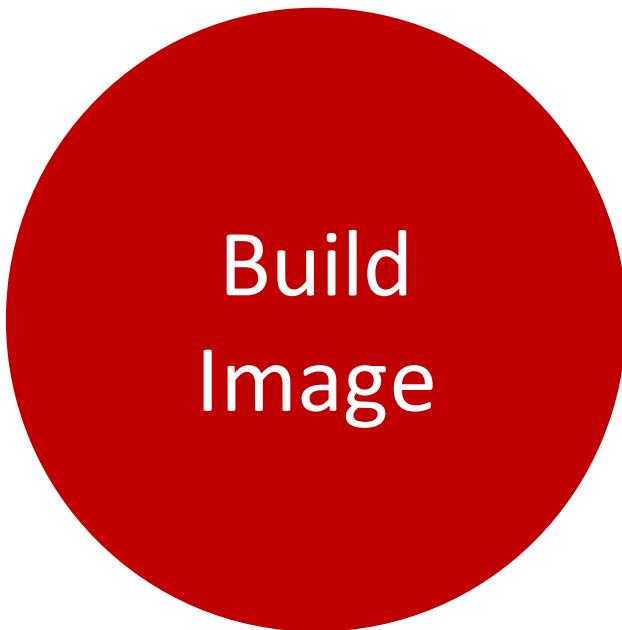
Anatomy of a Dockerfile

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	<p>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.</p> <p>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.</p>	CMD echo "This is a test." wc -
EXPOSE	The EXPOSE instruction 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



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
```

```
podshellJava -- Shell -- docker container run -it metamagic/podshelljava:8 -- bash -- 96x20
...ellJava -- Shell -- docker container run -it metamagic/podshelljava:8 ...8s/images/podshellJava -- root@9207f1605ef1: / -- vi Dockerfile + 
[ODC-MacBook-Pro-2017:podshellJava arafkarsh$ docker container run -it metamagic/podshelljava:8
Filesystem      Size   Used  Available Use% Mounted on
overlay        62.7G  13.8G   45.8G  23% /
tmpfs          64.0M     0    64.0M  0% /dev
tmpfs          999.3M     0   999.3M  0% /sys/fs/cgroup
/dev/sda1       62.7G  13.8G   45.8G  23% /etc/resolv.conf
/dev/sda1       62.7G  13.8G   45.8G  23% /etc/hostname
/dev/sda1       62.7G  13.8G   45.8G  23% /etc/hosts
shm            64.0M     0    64.0M  0% /dev/shm
tmpfs          999.3M     0   999.3M  0% /proc/acpi
tmpfs          64.0M     0    64.0M  0% /proc/kcore
tmpfs          64.0M     0    64.0M  0% /proc/keys
tmpfs          64.0M     0    64.0M  0% /proc/timer_list
tmpfs          64.0M     0    64.0M  0% /proc/sched_debug
tmpfs          999.3M     0   999.3M  0% /sys/firmware
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
/home/podadmin
bash-4.4$
```

```
podshellJava -- root@9207f1605ef1: / -- vi Dockerfile -- bash -- 79x35
~/k8s/images/podshelltomcat -- Shell -- bash
#####
#
# PodShell - Developer Tool box to test Pods in Kubernetes
#
# (C) MetaMagic Global Inc, NJ, USA, 2018
#
# Desire3D Micro Services Containers
#
# Version 0.1
#####
# Base Version
FROM anapsix/alpine-java

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

# Install wget, curl and nano
RUN apk update && apk add wget && apk add curl && apk add nano && apk add vim

# Create Directories
RUN mkdir -p /Home/podadmin/Softwares
RUN chmod -R 750 /Home/podadmin/Softwares

# Add user
RUN adduser -D -u 2048 -s /bin/bash podadmin

ADD .bashrc /home/podadmin/
RUN chown podadmin:podadmin /home/podadmin/.bashrc

WORKDIR /home/podadmin

# Run the Container as podadmin
USER podadmin

ENTRYPOINT /bin/bash
```

Docker Container Management

```
$ ID=$(docker container run -d 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

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.

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
```

ODC-MacBook-Pro-2017:podshellapache arafkarsh\$ docker container run -d -p 80:80 metamagic/podshellapache
e6fcbb1ae5481c5645b042af2d3b5c23e96ea903b98365778c6ed0baa4c03567b
ODC-MacBook-Pro-2017:podshellapache arafkarsh\$ curl localhost
<html>
<title>Apache 2 Home</title>
<body>
<h1> Apache 2 Web Server Running!</h1>

<h2> Hello World! </h2>
</body>
</html>
ODC-MacBook-Pro-2017:podshellapache arafkarsh\$

```
podshellapache — Shell — vi Dockerfile — bash — 81x26
~/k8s/images/podshellapache — Shell — vi Dockerfile
...ges/podshellJava — root@db907a6188d8: / — bash

#####
#
# PodShell - Developer Tool box to test Pods in Kubernetes
#
# (C) MetaMagic Global Inc, NJ, USA, 2018
#
# Desire3D Micro Services Containers - Apache 2
#
# Version 0.1
#####
FROM alpine:3.6

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

RUN apk add --no-cache apache2 && \
    mkdir -p /run/apache2

# Copy Starting page for Apache 2
COPY index.html /var/www/localhost/htdocs/index.html

# Apache 2 on Port 80
EXPOSE 80

# Run Apache2
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

1 Build Docker Tomcat image

1. Create your Dockerfile

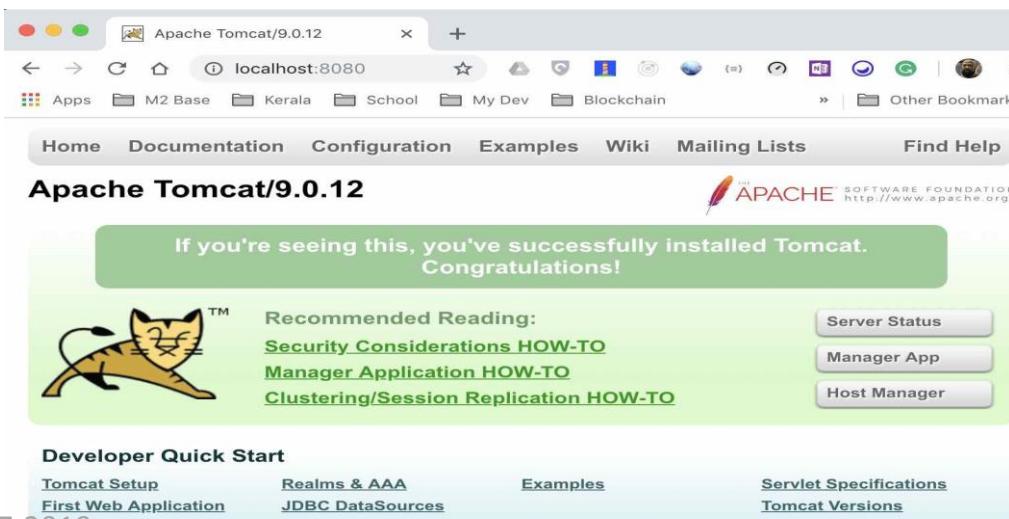
- FROM alpine
- RUN
- COPY
- EXPOSE
- ENTRYPOINT

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 — 70x27
.../images/podshelltomcat — Shell — vi Dockerfile ...hellJava — 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>

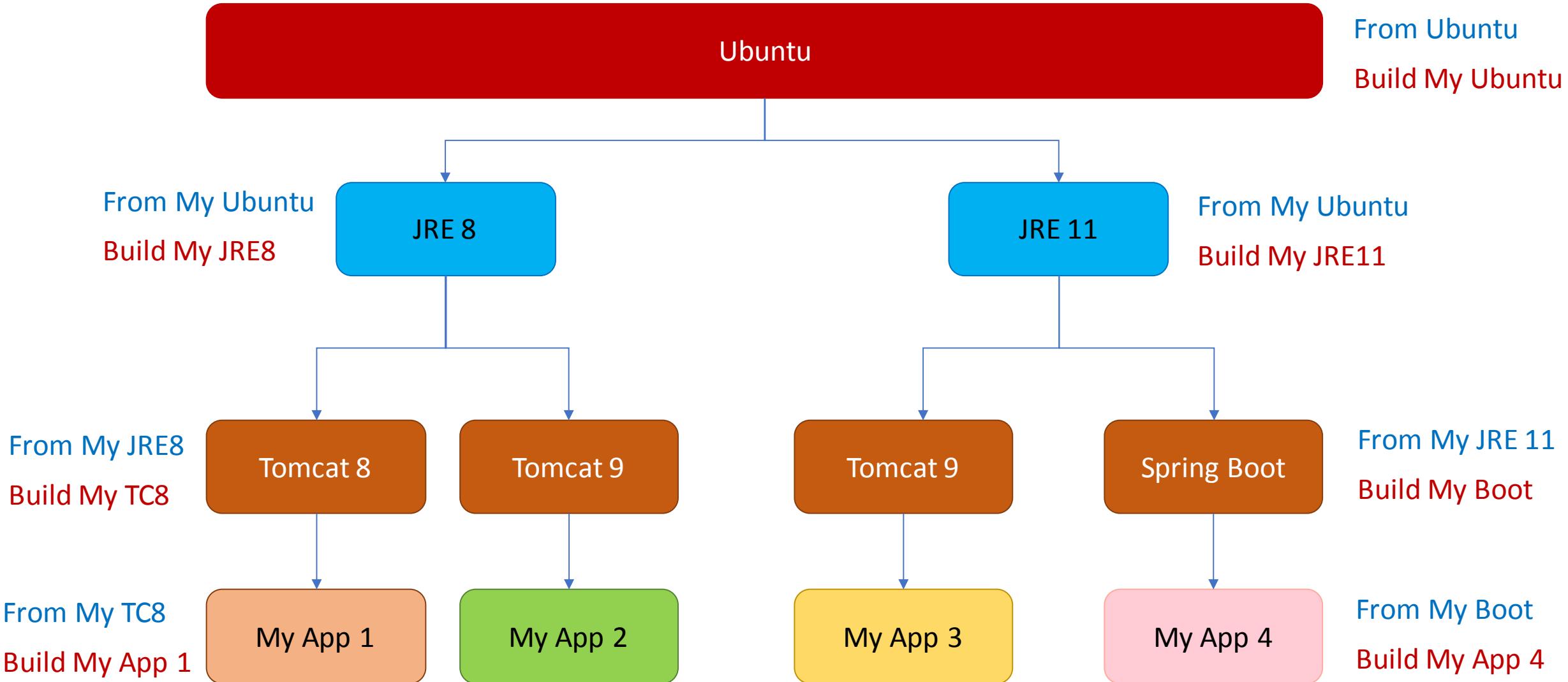
RUN mkdir /Home/Softwares/tomcat/

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

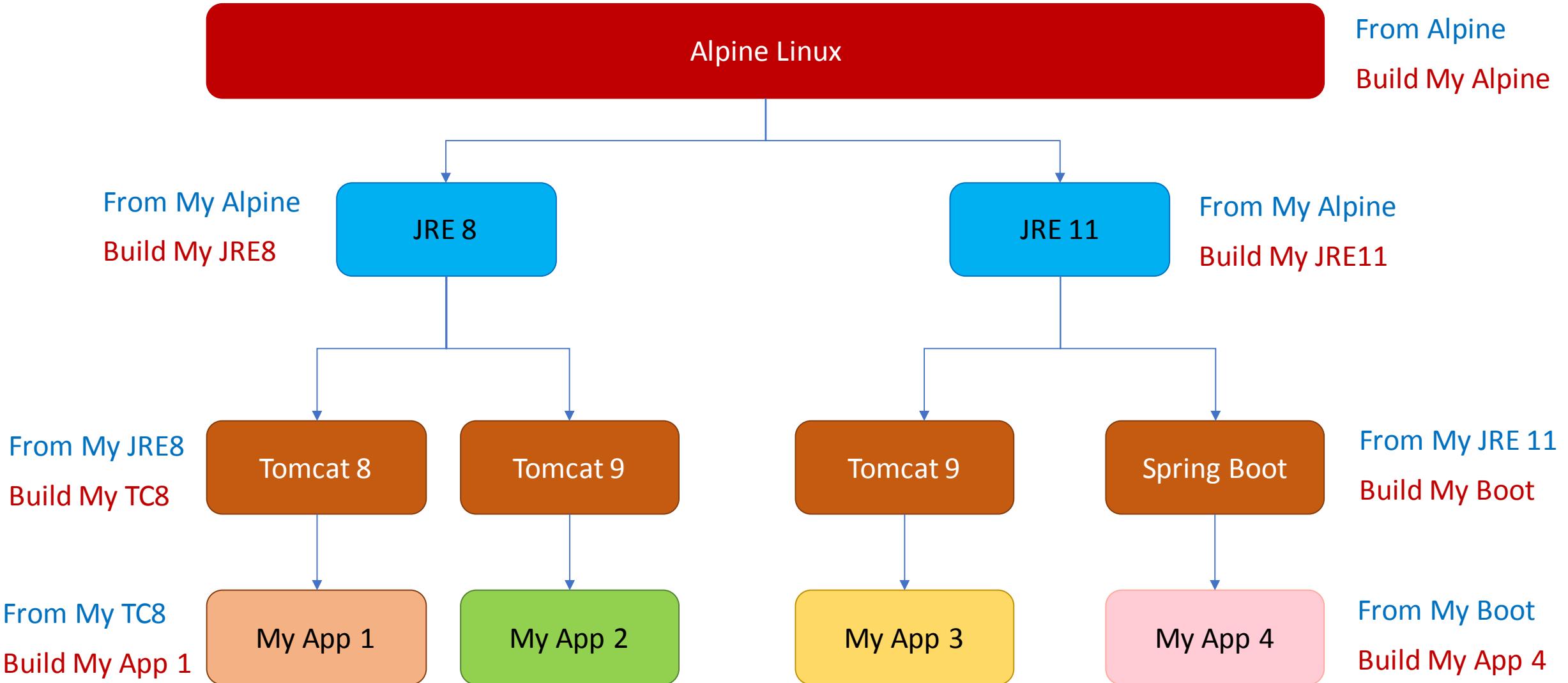
# Expose Tomcat port
EXPOSE 8080

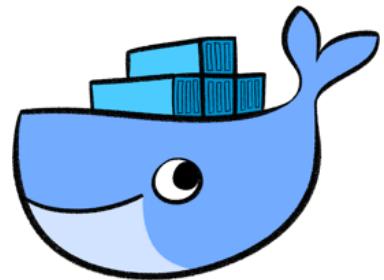
# Start Tomcat
ENTRYPOINT ["/Home/Softwares/tomcat/bin/catalina.sh", "run"]
```

Docker Images in the Github Workshop



Docker Images in the Github Workshop



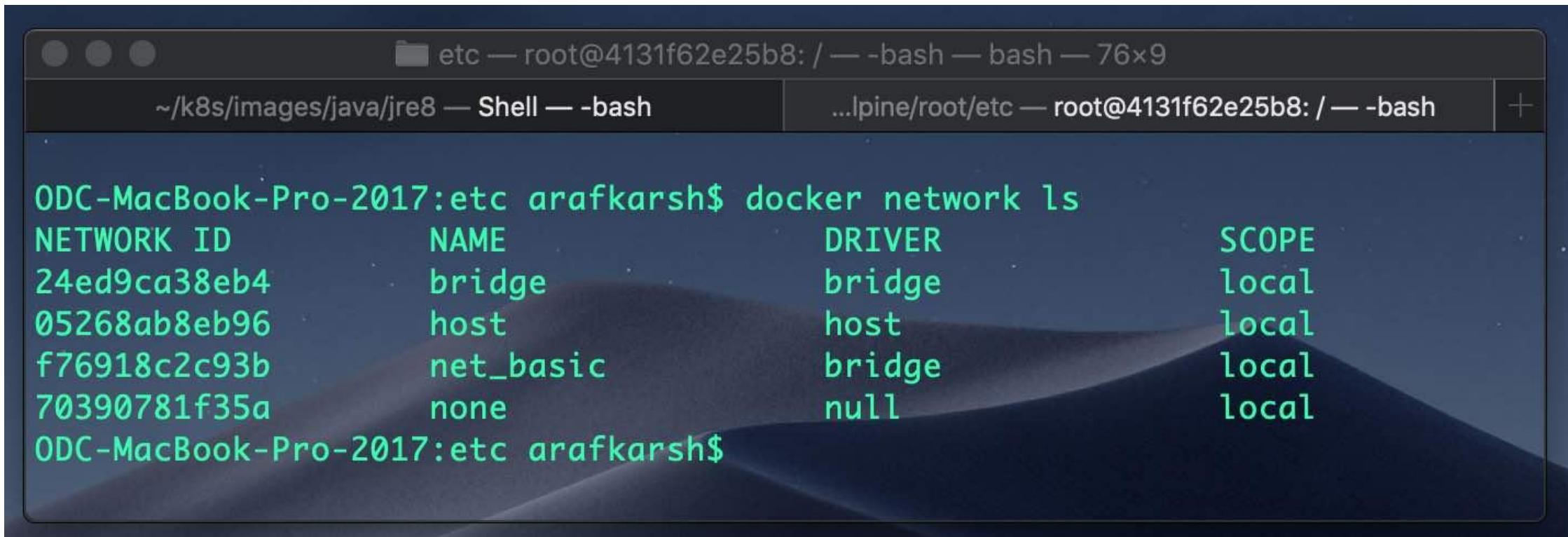


Docker Networking

- 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
```



The screenshot shows a terminal window with three tabs. The active tab is titled 'etc — root@4131f62e25b8: / — -bash — bash — 76x9'. The other tabs are '~/k8s/images/java/jre8 — Shell — -bash' and '...pine/root/etc — root@4131f62e25b8: / — -bash'. The terminal displays the command 'docker network ls' followed by a table of network configurations:

NETWORK ID	NAME	DRIVER	SCOPE
24ed9ca38eb4	bridge	bridge	local
05268ab8eb96	host	host	local
f76918c2c93b	net_basic	bridge	local
70390781f35a	none	null	local

```
$ 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 bash
~/k8s/images/java/jre8 — Shell — vi Dockerfile Dockerfile ...4131f62e25b8: / — docker container run -it metamagic/alpine — bash — 92x19

[...]
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 brd 00:00:00:00:00:00
4: 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=host alpine ip address

```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -it --net=host metamagic/alpine bash
~/k8s/images/java/jre8 — Shell — vi Dockerfile Dockerfile ...b8: / — docker container run -it --net=host metamagic/alpine — bash — 92x41

[...]
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 02:50:00:00:01 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:fe01:1/64 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 brd 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:effff: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 master 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 master 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 container run --rm --net=none alpine ip address

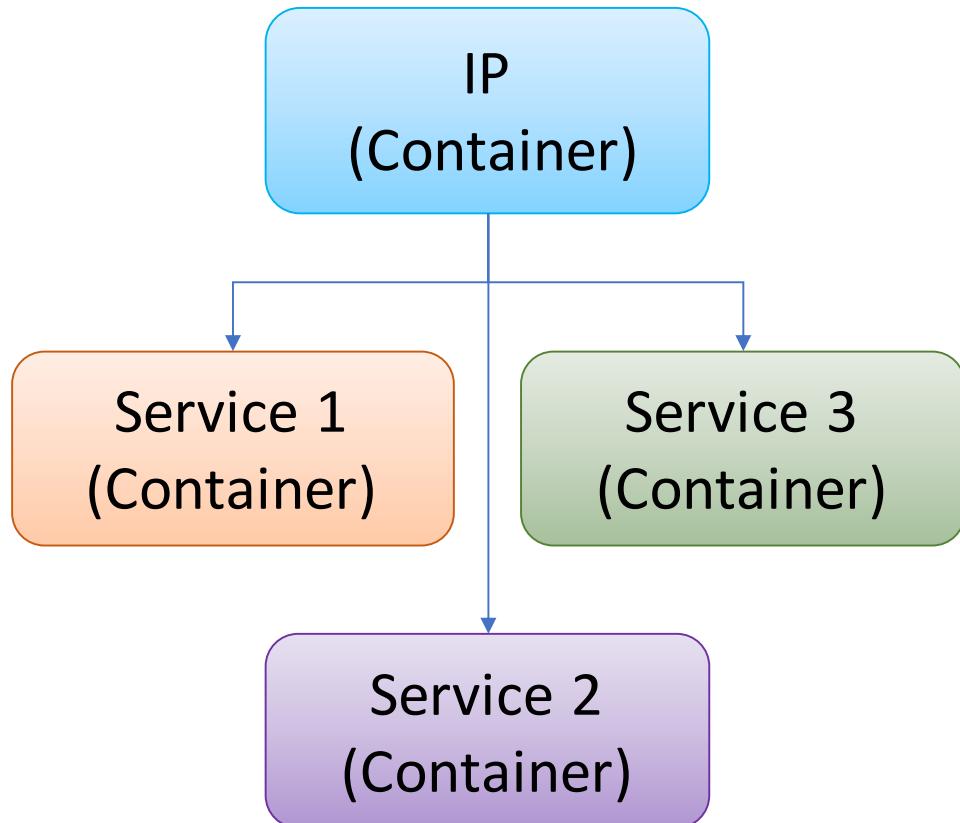
```
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -it --net=none metamagic/alpine bash
~/k8s/images/java/jre8 — Shell — bash ...b8: / — docker container run -it --net=none metamagic/alpine — bash — 92x14

[...]
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 brd 00:00:00:00:00:00
/ #
```

No Network Stack

Docker Containers

Sharing IP Address



```
$ docker container run --name ipctr -itd alpine
```

```
$ docker container exec ipctr ip address
```

```

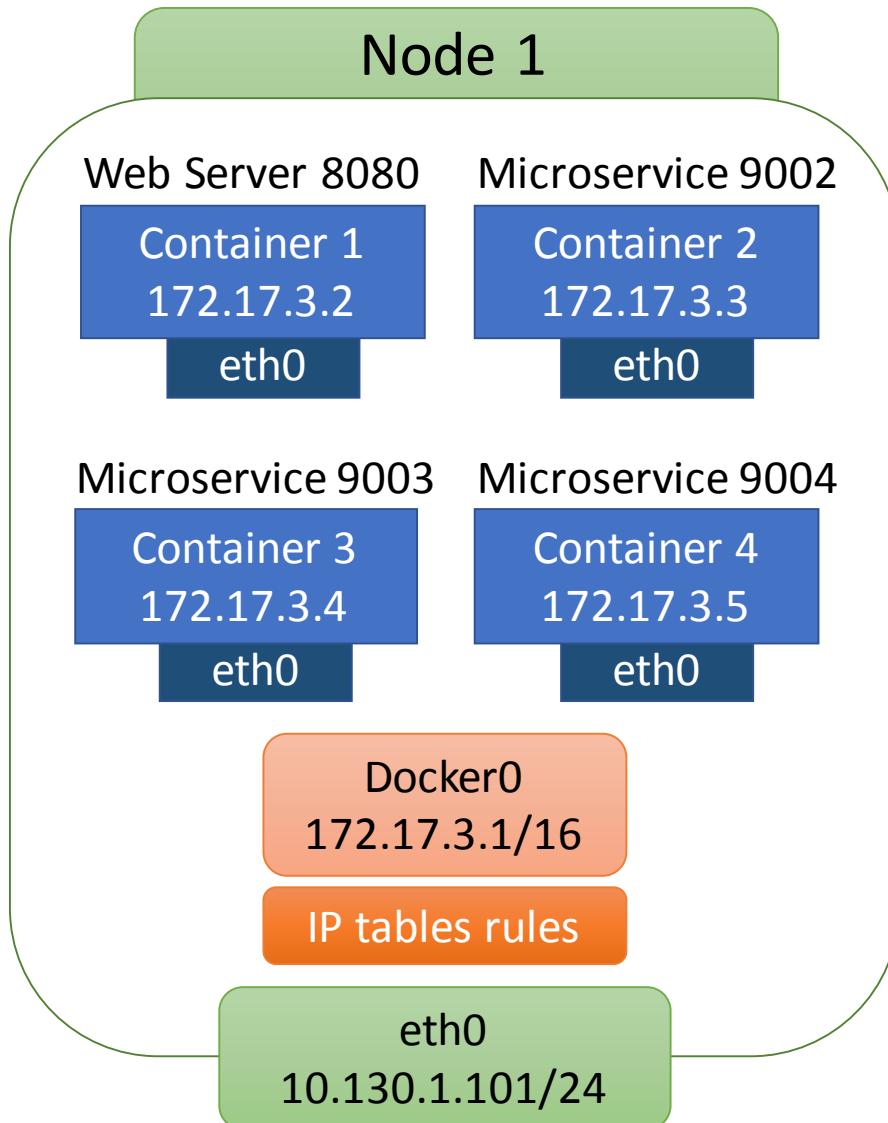
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run -itd --name ipctr metamagic/alpine
142cdc73df5427e013feafa996e705804134d837d39aa0688393f438c9deedb0
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container exec ipctr ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 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 brd 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
```

```

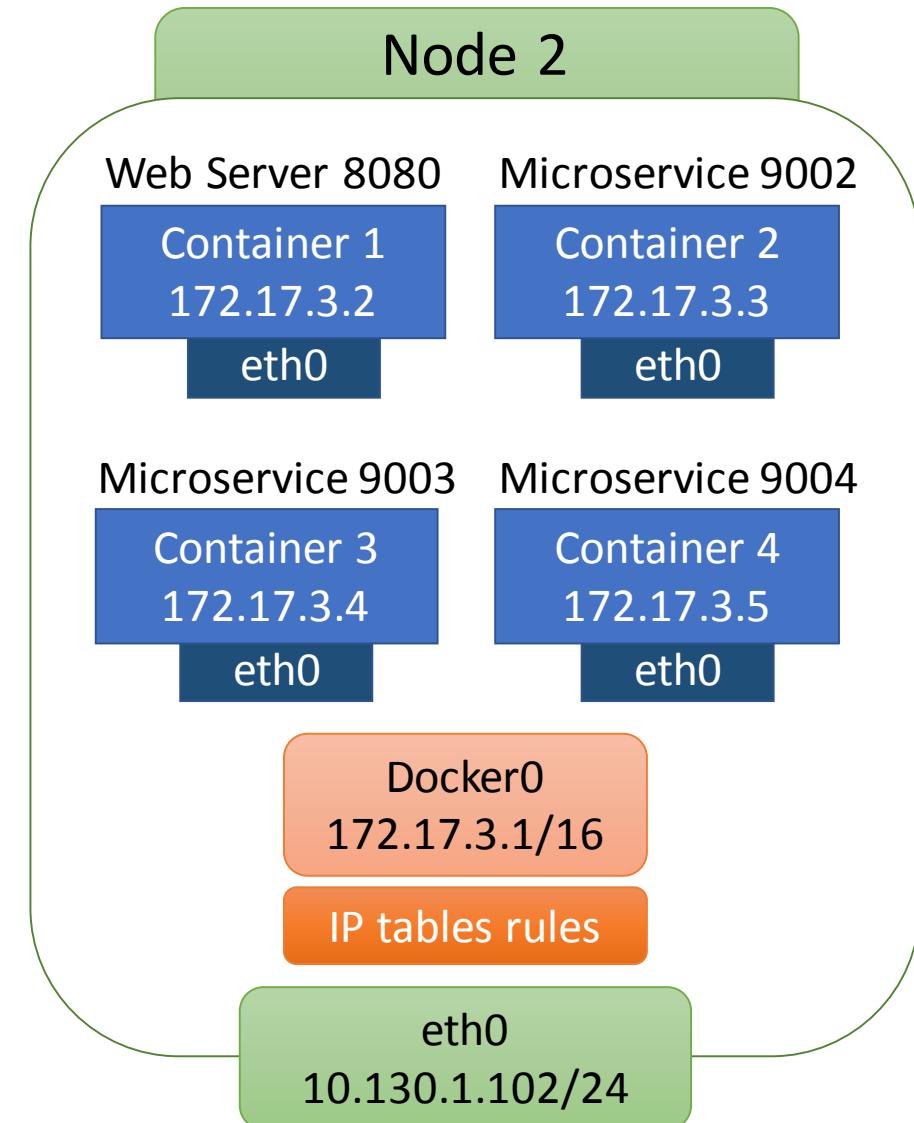
ODC-MacBook-Pro-2017:etc arafkarsh$ docker container run --rm --net container:ipctr metamagi
c/alpine ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft 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 brd 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.



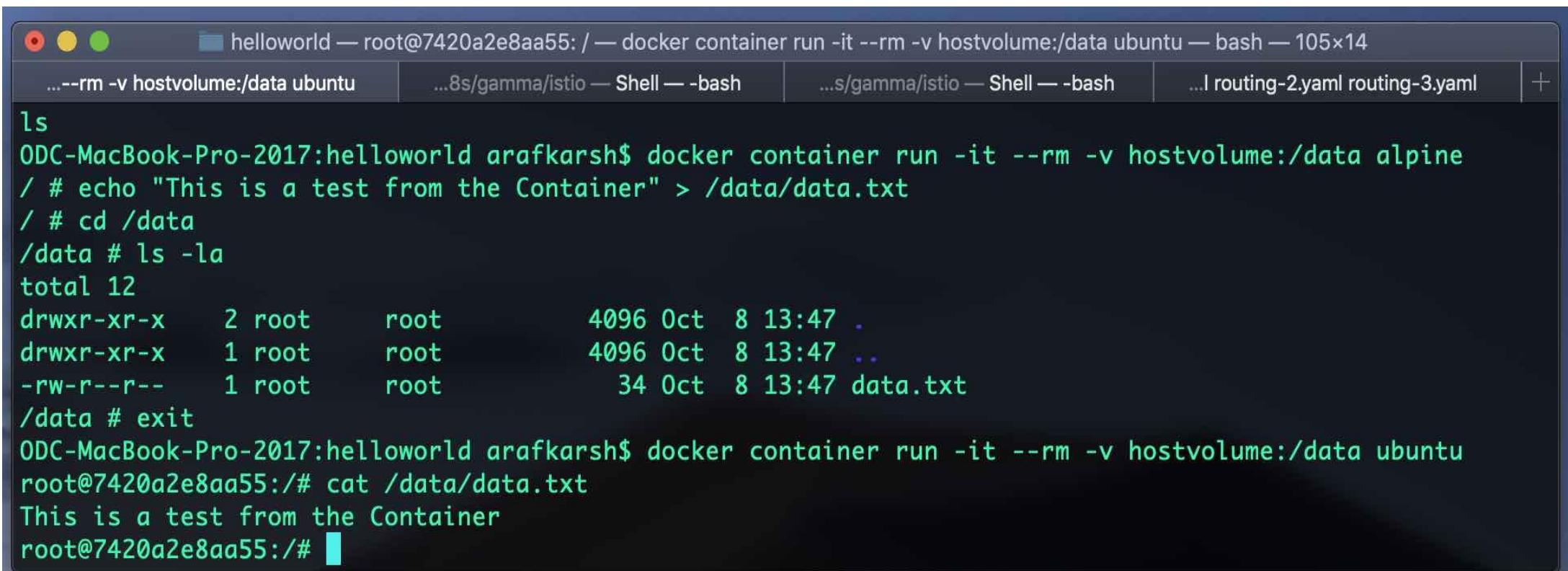
Docker Volumes

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
```



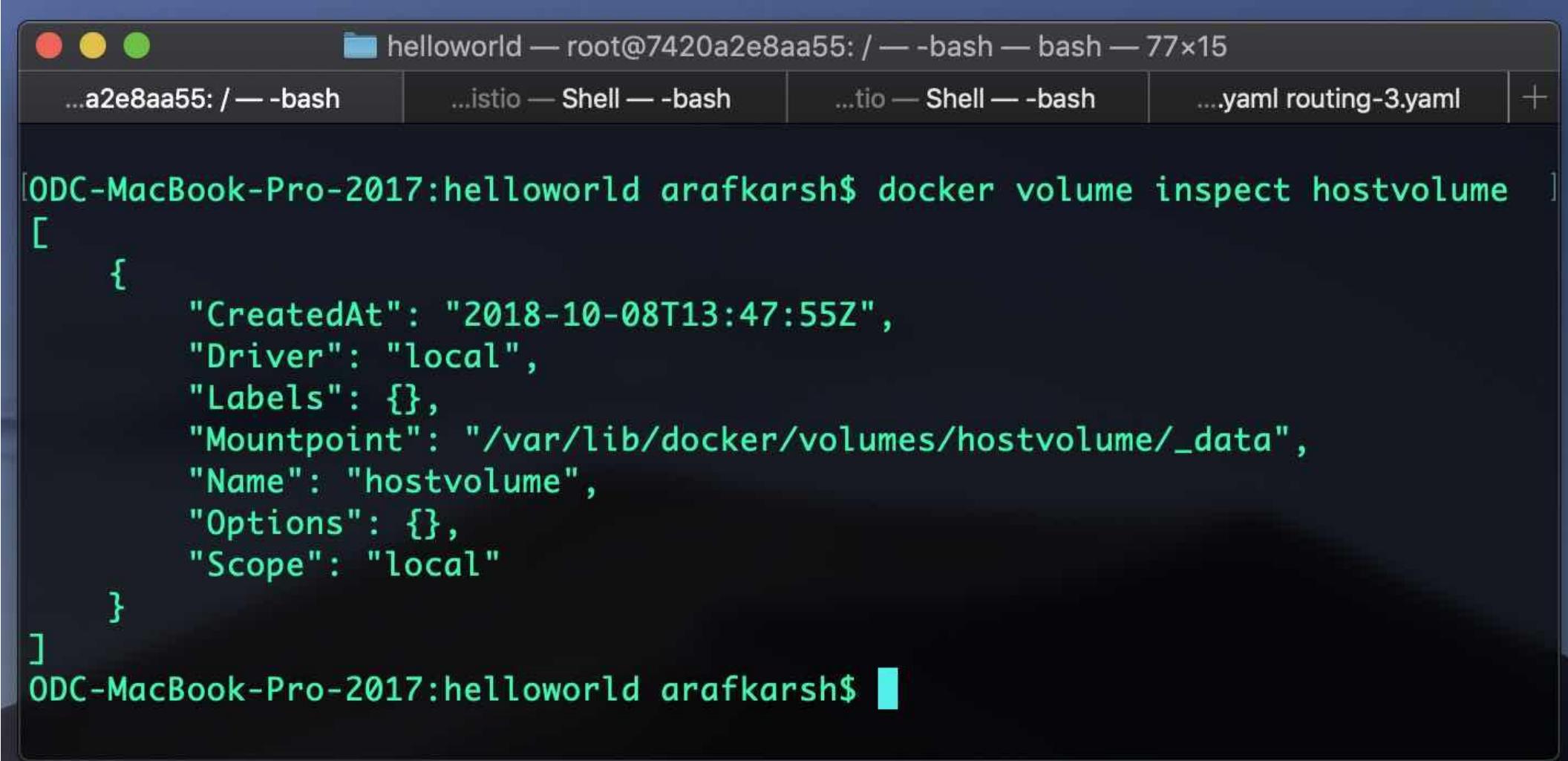
The screenshot shows a terminal window with several tabs at the top. The active tab is titled 'helloworld — root@7420a2e8aa55: / — docker container run -it --rm -v hostvolume:/data ubuntu — bash — 105x14'. Below the tabs, there are four smaller windows showing other processes: '...--rm -v hostvolume:/data ubuntu', '...8s/gamma/istio — Shell — bash', '...s/gamma/istio — Shell — bash', and '...l routing-2.yaml routing-3.yaml'. The main terminal area displays the following command sequence:

```
ls  
ODC-MacBook-Pro-2017:helloworld arafkarsh$ docker container run -it --rm -v hostvolume:/data alpine  
/ # echo "This is a test from the Container" > /data/data.txt  
/ # cd /data  
/data # ls -la  
total 12  
drwxr-xr-x  2 root      root          4096 Oct  8 13:47 .  
drwxr-xr-x  1 root      root          4096 Oct  8 13:47 ..  
-rw-r--r--  1 root      root          34 Oct  8 13:47 data.txt  
/data # exit  
ODC-MacBook-Pro-2017:helloworld arafkarsh$ docker container run -it --rm -v hostvolume:/data ubuntu  
root@7420a2e8aa55:/# cat /data/data.txt  
This is a test from the Container  
root@7420a2e8aa55:/#
```

Docker Volumes

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

Mount Specific File Path



The screenshot shows a macOS terminal window with four tabs at the top: '...a2e8aa55: / — -bash', '...istio — Shell — -bash', '...tio — Shell — -bash', and '....yaml routing-3.yaml'. The active tab displays the command:

```
[ODC-MacBook-Pro-2017:helloworld arafkarsh$ docker volume inspect hostvolume ]
```

Below the command, the output of the `docker volume inspect` command is shown as a JSON object:

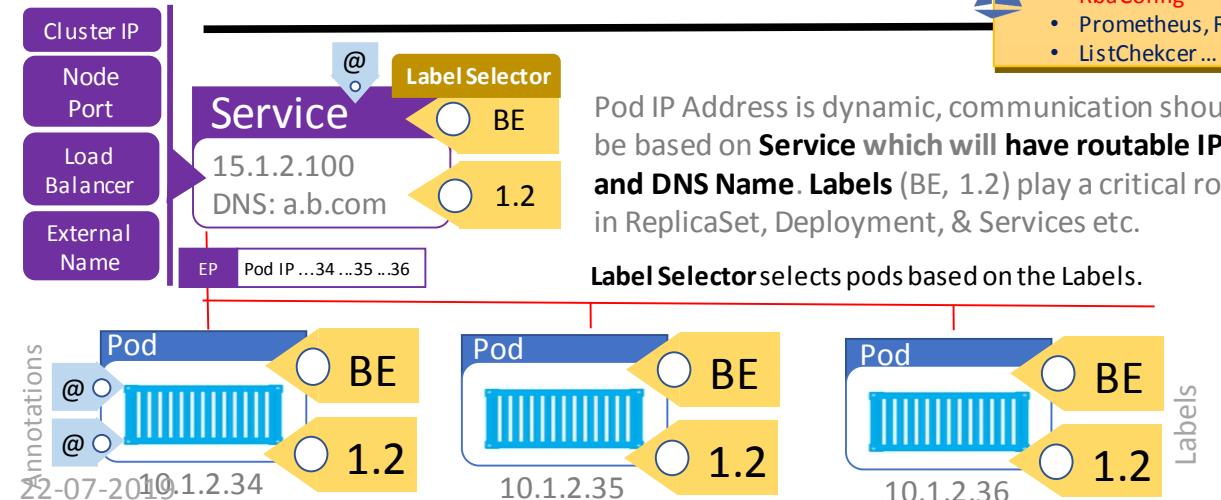
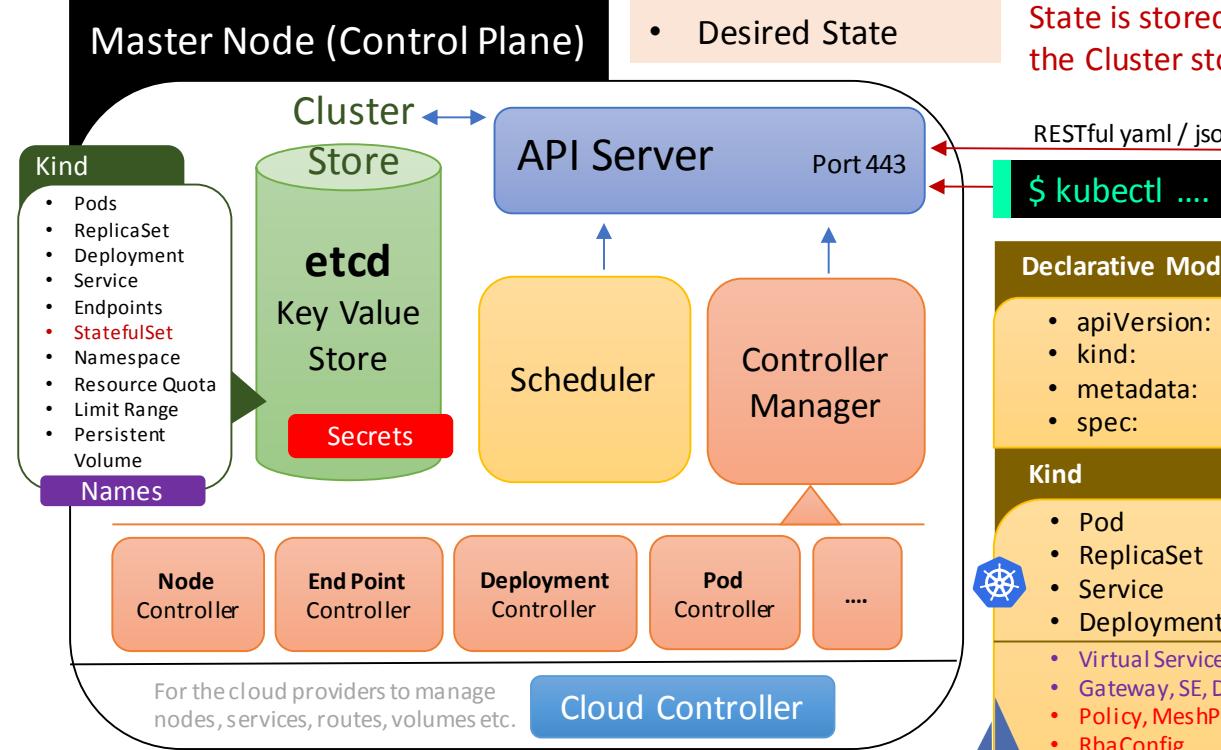
```
[  
  {  
    "CreatedAt": "2018-10-08T13:47:55Z",  
    "Driver": "local",  
    "Labels": {},  
    "Mountpoint": "/var/lib/docker/volumes/hostvolume/_data",  
    "Name": "hostvolume",  
    "Options": {},  
    "Scope": "local"  
  }]  
ODC-MacBook-Pro-2017:helloworld arafkarsh$
```



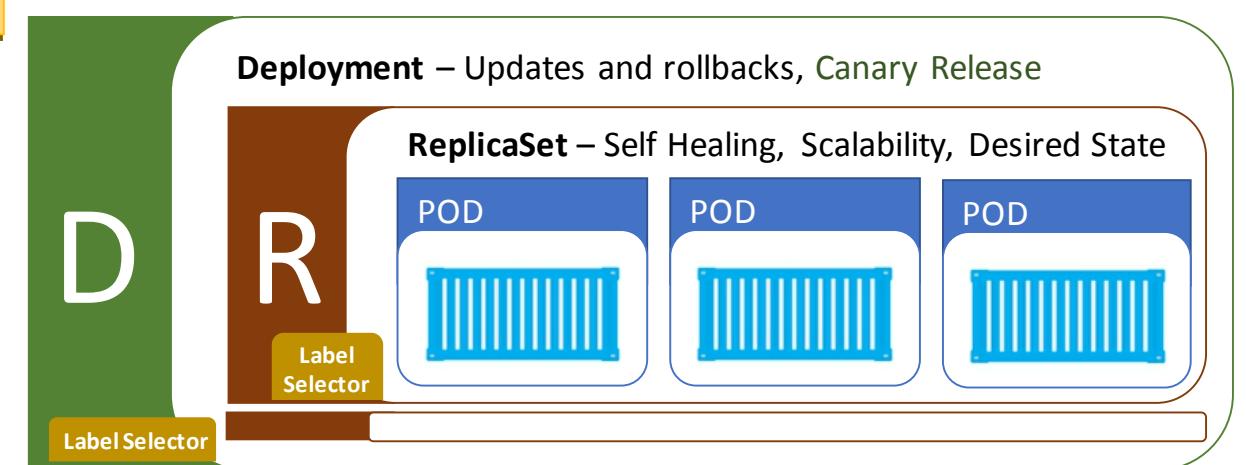
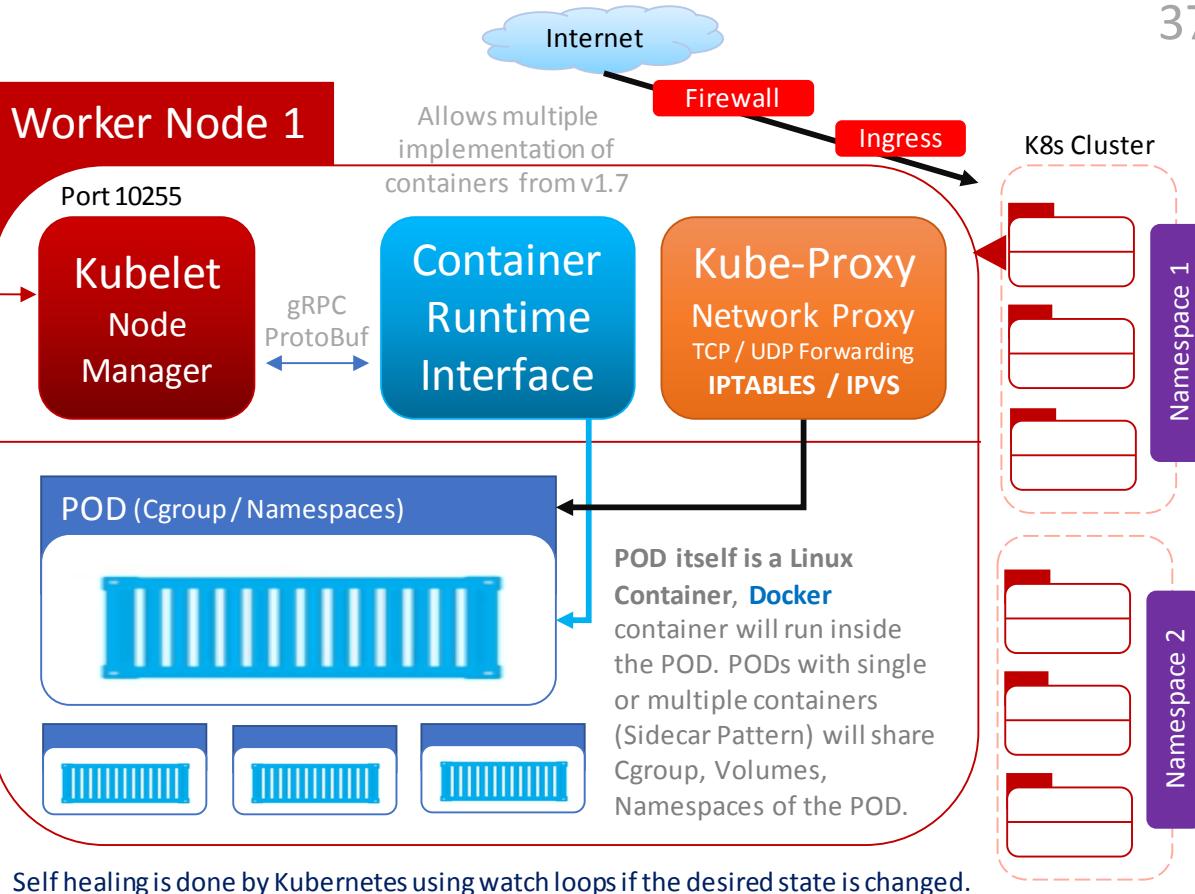
Kubernetes



Kubernetes Architecture



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





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/
```



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 replicaset
```

List all the ReplicaSet

```
$ kubectl describe replicaset
```

Describe the ReplicaSet

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

List the Service Hello-World-Service with Cluster 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



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.

```
$ 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

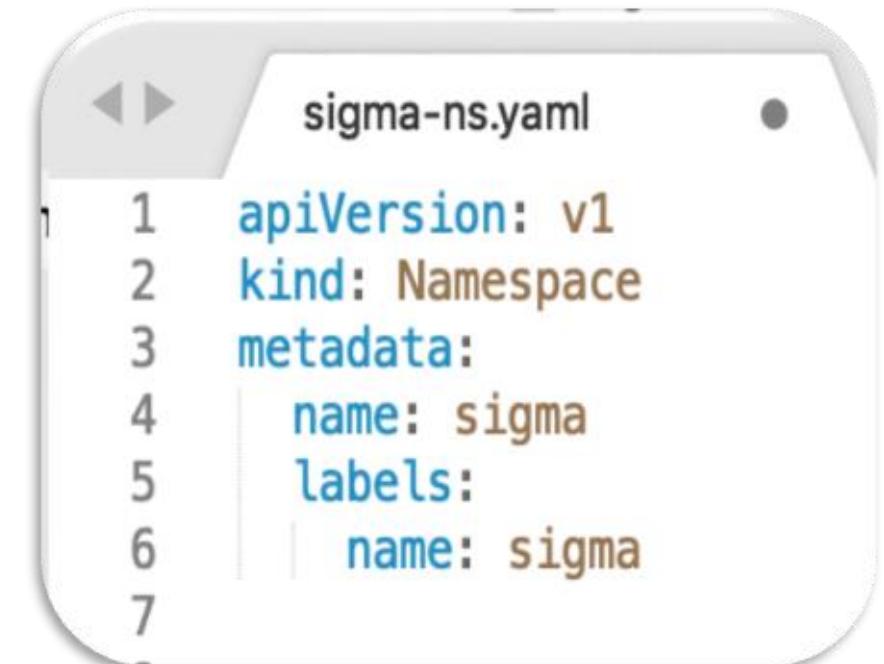
```
$ kubectl create -f app-ns.yml
```

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
```

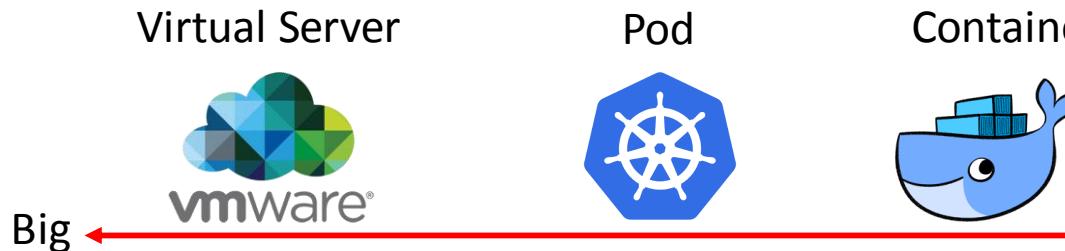


The above command will let you switch the namespace to your namespace (your-ns).



Kubernetes Pods

Atomic Unit



- Pod is a shared environment for one or 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.yaml
$ 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

```

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: app1pod
5   namespace: sigma
6   labels:
7     desire3d.io/name: app1pod
8     desire3d.io/version: "1.0.0"
9     desire3d.io/release: stable
10    desire3d.io/tier: fe
11    desire3d.io/zone: prod
12    desire3d.io/managed-by: m2
13 spec:
14   containers:
15     - name: app1pod-ctr
16       image: metamagicglobal/omega
17       imagePullPolicy: Always
18       ports:
19         - containerPort: 8080
20   resources:
21     requests:
22       memory: "64Mi"
23       cpu: "200m"
24     limits:
25       memory: "96Mi"
26       cpu: "250m"
```



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 exec pod-name ps aux
```

Execute commands in the first Container in the Pod

```
$ 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
```

```
$ 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 -it pod-name sh
```

Log into the Container Shell



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
1 apiVersion: apps/v1beta2
2 kind: ReplicaSet
3 metadata:
4   name: app1pod-rs
5   namespace: sigma
6   labels:
7     desire3d.io/name: app1rs
8 spec:
9   replicas: 5
10  selector:
11    matchLabels:
12      desire3d.io/name: app1pod
13      desire3d.io/zone: prod
14      desire3d.io/managed-by: m2
15      desire3d.io/release: stable
16 template:
17  metadata:
18    labels:
19      desire3d.io/name: app1pod
20      desire3d.io/version: "1.0.0"
21      desire3d.io/release: stable
22      desire3d.io/tier: fe
23      desire3d.io/zone: prod
24      desire3d.io/managed-by: m2
25 spec:
26   containers:
27     - name: app1pod-ctr
28       image: metamagicglobal/omega
29       imagePullPolicy: IfNotPresent
30       ports:
31         - containerPort: 8080
32       resources:
33         requests:
34           memory: "64Mi"
35           cpu: "200m"
36         limits:
37           memory: "96Mi"
38           cpu: "250m"
```



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
1 apiVersion: apps/v1beta2
2 kind: ReplicaSet
3 metadata:
4   name: app1pod-rs
5   namespace: sigma
6   labels:
7     desire3d.io/name: app1rs
8 spec:
9   replicas: 5
10  selector:
11    matchLabels:
12      desire3d.io/name: app1pod
13      desire3d.io/zone: prod
14      desire3d.io/managed-by: m2
15      desire3d.io/release: stable
16 template:
17 metadata:
18   labels:
19     desire3d.io/name: app1pod
20     desire3d.io/version: "1.0.0"
21     desire3d.io/release: stable
22     desire3d.io/tier: fe
23     desire3d.io/zone: prod
24     desire3d.io/managed-by: m2
25 spec:
26   containers:
27     - name: app1pod-ctr
28       image: metamagicglobal/omega
29       imagePullPolicy: IfNotPresent
30       ports:
31         - containerPort: 8080
32       resources:
33         requests:
34           memory: "64Mi"
35           cpu: "200m"
36         limits:
37           memory: "96Mi"
38           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  x
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: app1pod-deploy
5 spec:
6   replicas: 5
7   selector:
8     matchLabels:
9       desire3d.io/name: app1pod
10      desire3d.io/zone: prod
11 minReadySeconds: 10
12 strategy:
13   type: RollingUpdate
14   rollingUpdate:
15     maxUnavailable: 1
16     maxSurge: 1
17 template:
18   metadata:
19     labels:
20       desire3d.io/name: app1pod
21       desire3d.io/version: "1.0.0"
22       desire3d.io/release: stable
23       desire3d.io/tier: fe
24       desire3d.io/zone: prod
25       desire3d.io/managed-by: m2
26 spec:
27   containers:
28     - name: app1pod-ctr
29       image: metamagicglobal/omega
30       imagePullPolicy: IfNotPresent
31       ports:
32         - containerPort: 80
33       resources:
34         requests:
35           memory: "64Mi"
36             cpu: "200m"
37         limits:
38           memory: "96Mi"
             cpu: "250m"
```

```
app1-dep-v2.yaml  x
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: app1pod-deploy
5 spec:
6   replicas: 5
7   selector:
8     matchLabels:
9       desire3d.io/name: app1pod
10      desire3d.io/zone: prod
11 minReadySeconds: 10
12 strategy:
13   type: RollingUpdate
14   rollingUpdate:
15     maxUnavailable: 1
16     maxSurge: 1
17 template:
18   metadata:
19     labels:
20       desire3d.io/name: app1pod
21       desire3d.io/version: "1.0.2"
22       desire3d.io/release: stable
23       desire3d.io/tier: fe
24       desire3d.io/zone: prod
25       desire3d.io/managed-by: m2
26 spec:
27   containers:
28     - name: app1pod-ctr
29       image: metamagicglobal/omega:v2
30       imagePullPolicy: IfNotPresent
31       ports:
32         - containerPort: 80
33       resources:
34         requests:
35           memory: "64Mi"
36             cpu: "200m"
37         limits:
38           memory: "96Mi"
             cpu: "250m"
```



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.



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
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: app1pod-svc
5   namespace: sigma
6   labels:
7     desire3d.io/name: app1svc
8 spec:
9   type: NodePort
10  selector:
11    desire3d.io/name: app1pod
12    desire3d.io/zone: prod
13    desire3d.io/release: stable
14  ports:
15    - port: 80
16      targetPort: 80
17      protocol: TCP
18      name: http
```

Service Types

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



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 externalName 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.

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

```
shopping-ingress.yaml x
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: shoppingportal-ingress
5   namespace: shoppingportal
6   annotations:
7     nginx.ingress.kubernetes.io/ssl-redirect: \"false\"
8 spec:
9   rules:
10    - http:
11      paths:
12        - path: /ui
13          backend:
14            serviceName: k8uiworkshopservice
15            servicePort: 80
16        - path: /productms
17          backend:
18            serviceName: productservice
19            servicePort: 80
20        - path: /productreviewms
21          backend:
22            serviceName: productreviewservice
23            servicePort: 80
24
```



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 x
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: workshop-ingress
5   namespace: shoppingportal
6   annotations:
7     kubernetes.io/ingress.class: alb
8     alb.ingress.kubernetes.io/target-type: ip
9     alb.ingress.kubernetes.io/scheme: internet-facing
10    alb.ingress.kubernetes.io/tags: Environment=dev,Team=workshop
11    alb.ingress.kubernetes.io/subnets: subnet-05ea8630b6
12 spec:
13   rules:
14     - http:
15       paths:
16         - path: /ui/*
17           backend:
18             serviceName: k8uiworkshopservice
19             servicePort: 80
20         - path: /productms/*
21           backend:
22             serviceName: productservice
23             servicePort: 80
24         - path: /productreviewms/*
25           backend:
26             serviceName: productreviewservice
27             servicePort: 80
28
```



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 x
1 apiVersion: autoscaling/v1
2 kind: HorizontalPodAutoscaler
3 metadata:
4   name: product-hpa
5   namespace: shoppingportal
6 spec:
7   scaleTargetRef:
8     apiVersion: apps/v1beta2
9     kind: Deployment
10    name: productreview-deploy
11    minReplicas: 1
12    maxReplicas: 10
13    targetCPUUtilizationPercentage: 10
14
```

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
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
app1pod-deploy	Deployment/app1pod-deploy	<unknown>/30%	3	10	3	33s

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
```

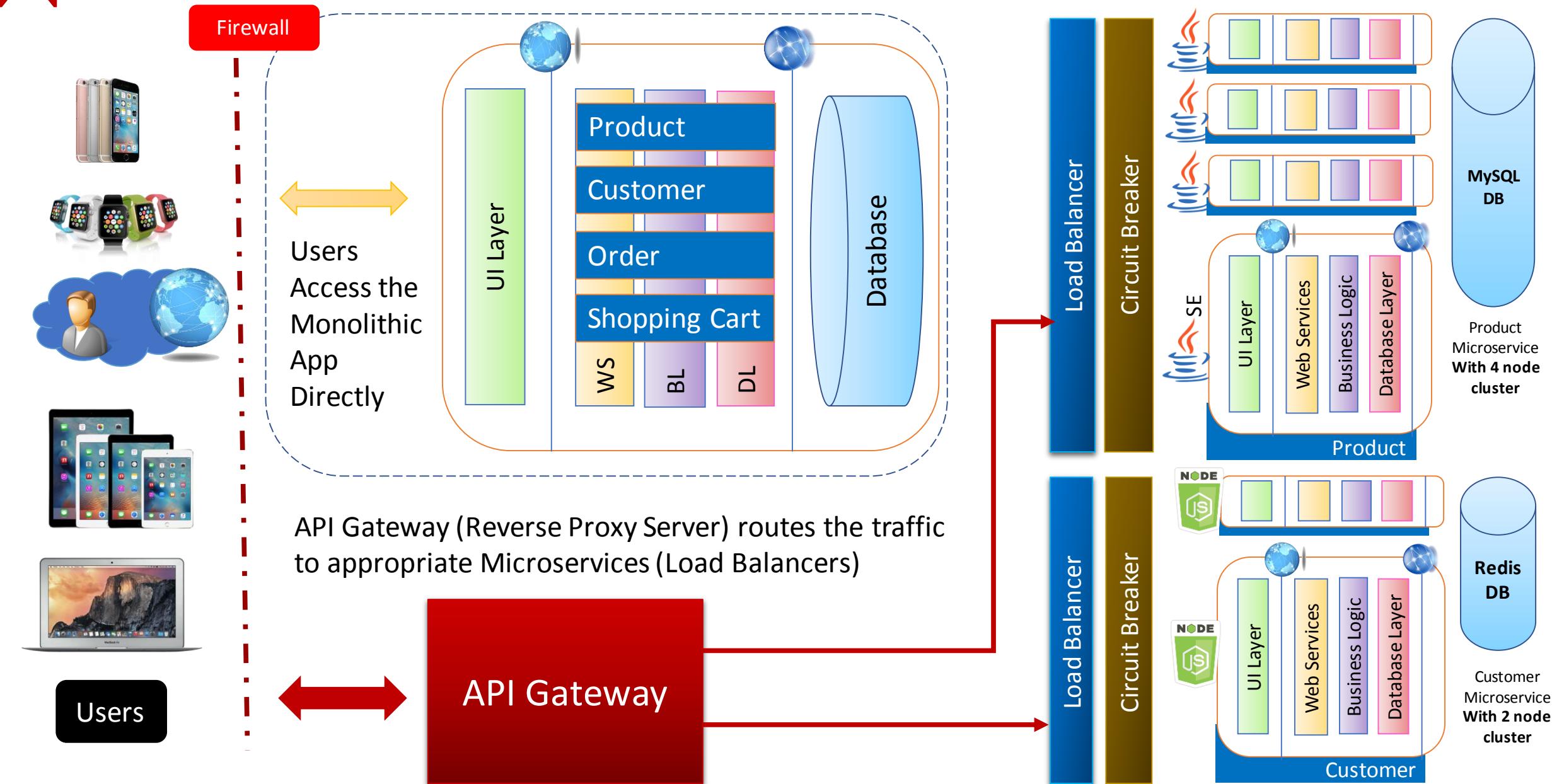
To attach to the running container

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

Infrastructure Design Patterns

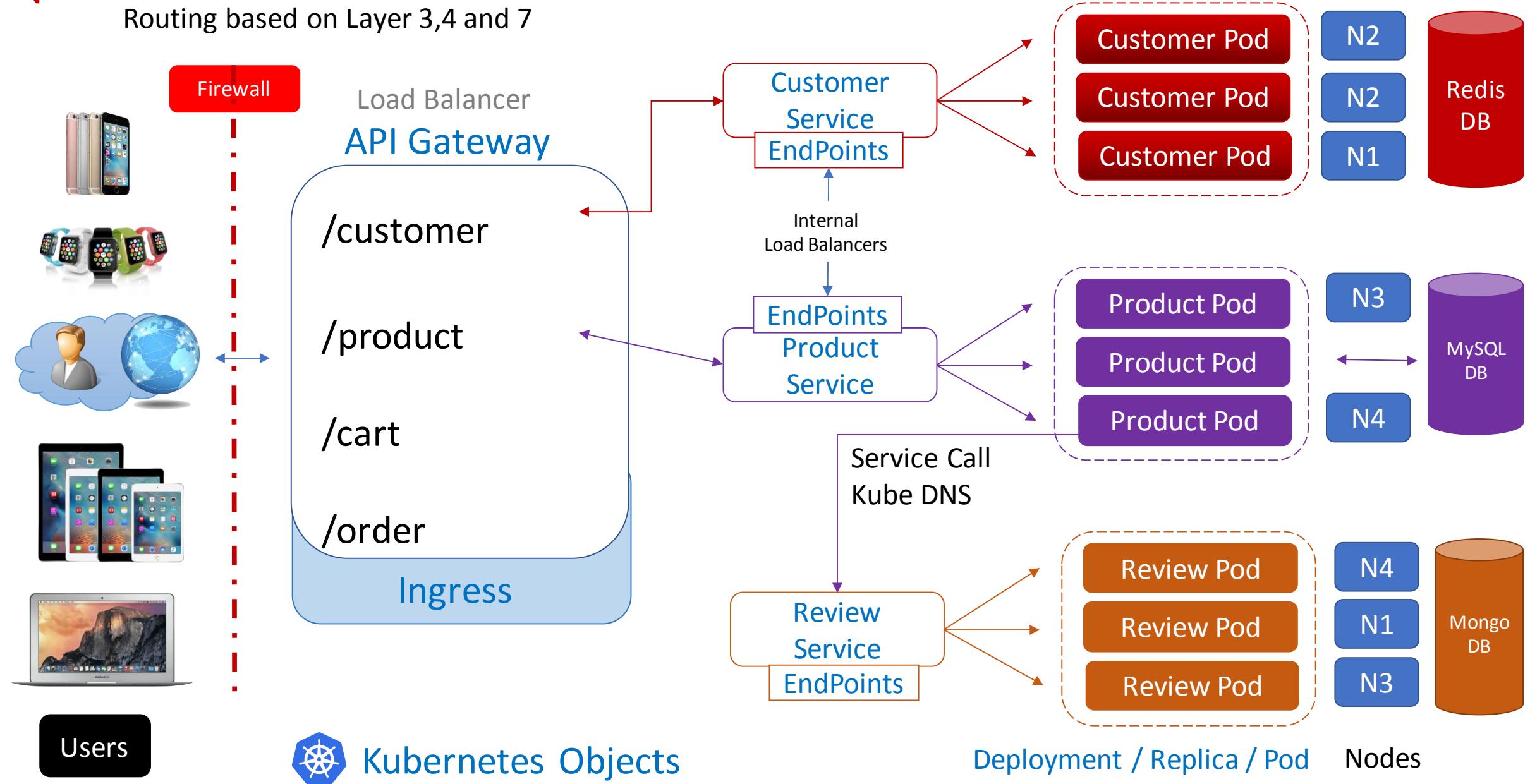
- API Gateway
- Load balancer
- Service discovery
- Circuit breaker
- Service Aggregator
- Let-it crash pattern

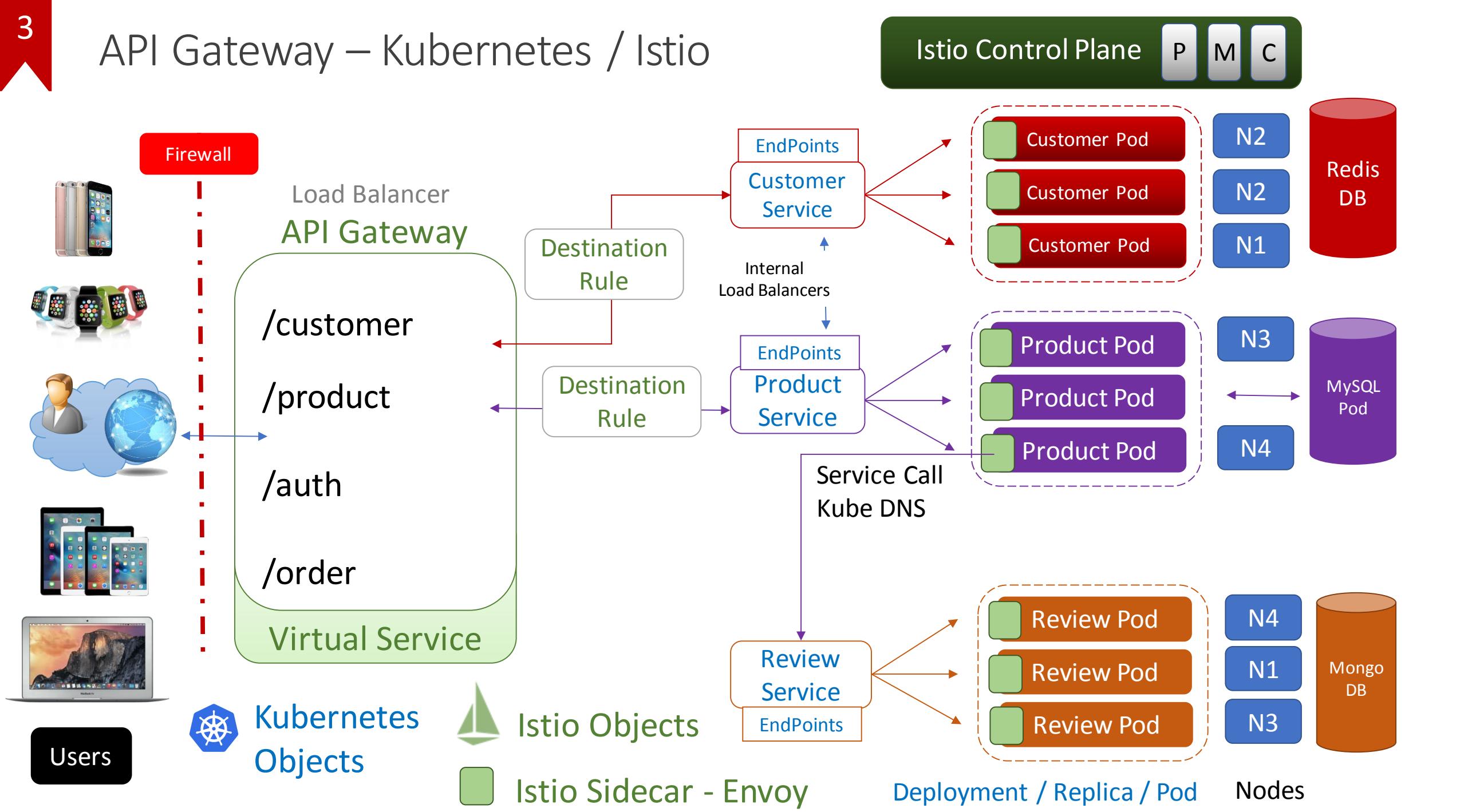
API Gateway Design Pattern – Software Stack



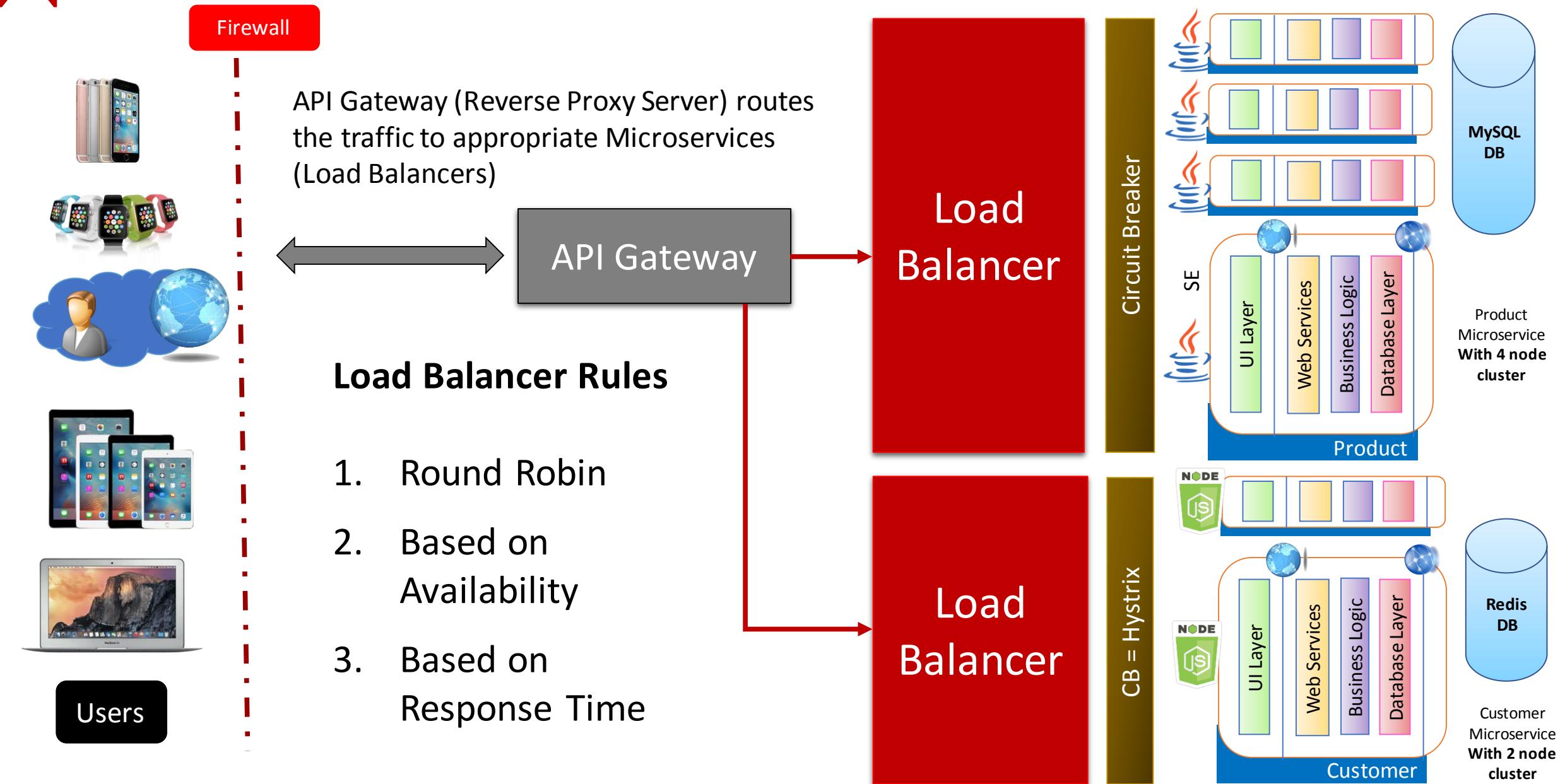
API Gateway – Kubernetes Implementation

Routing based on Layer 3,4 and 7

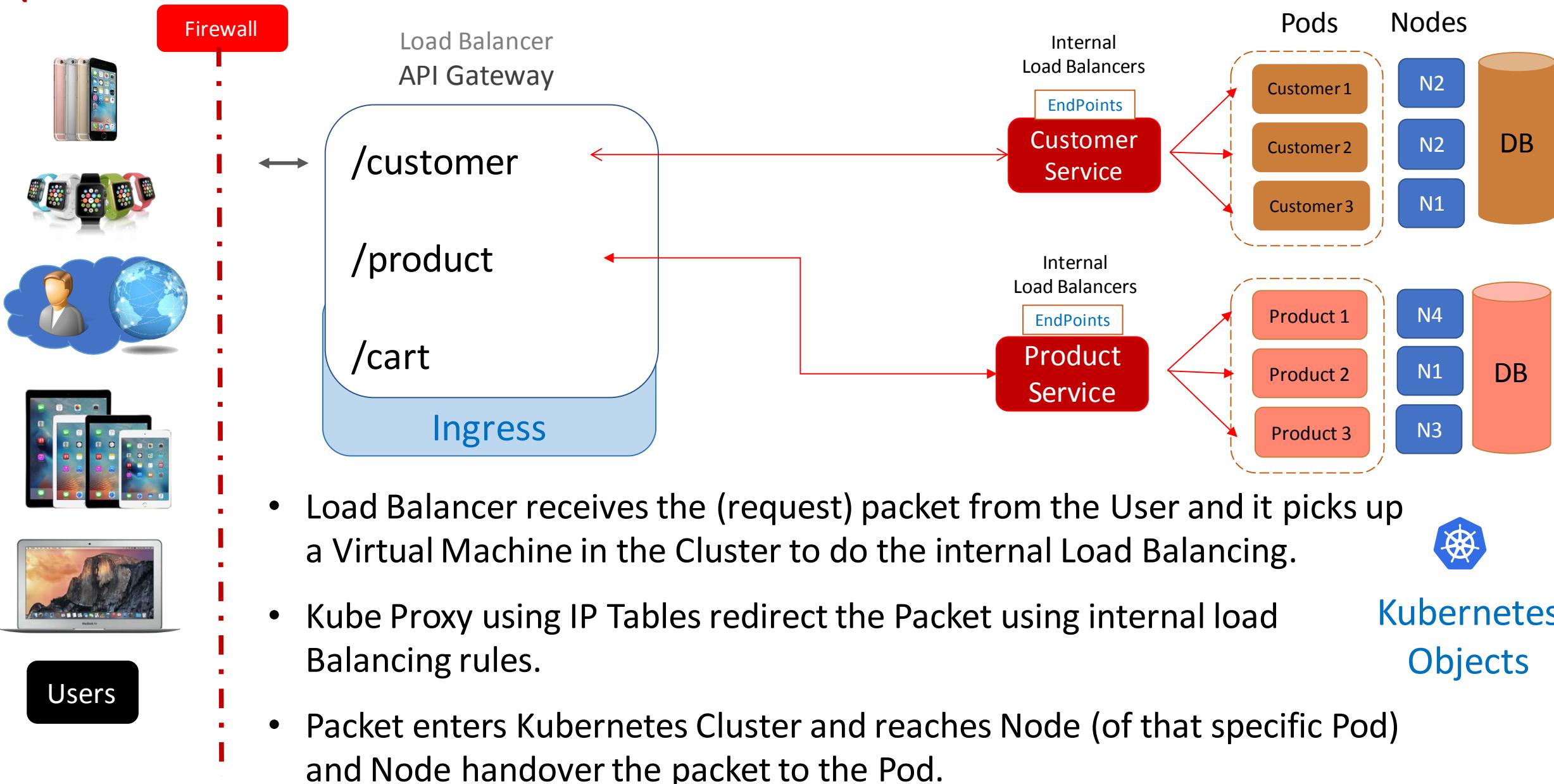




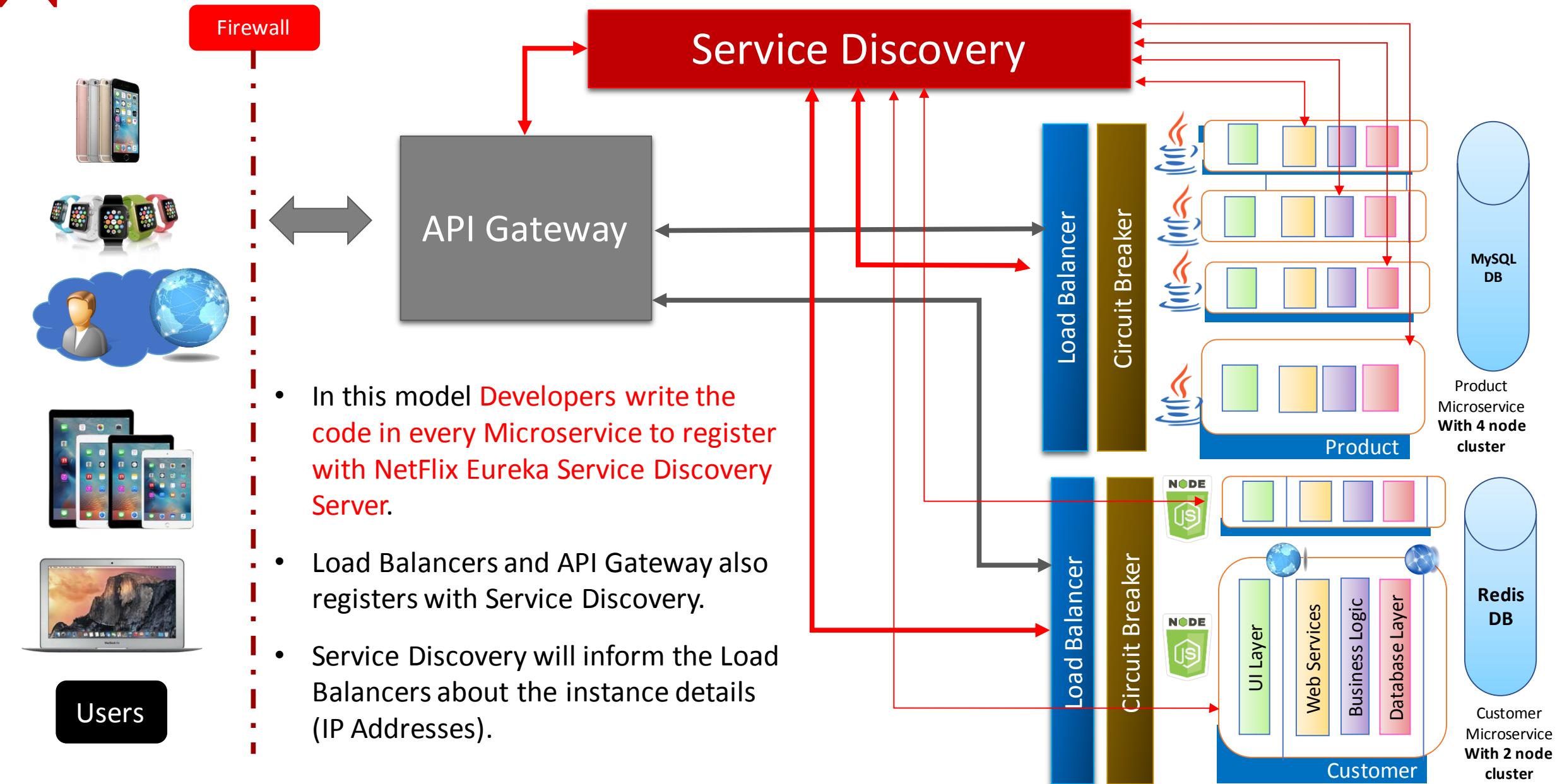
Load Balancer Design Pattern



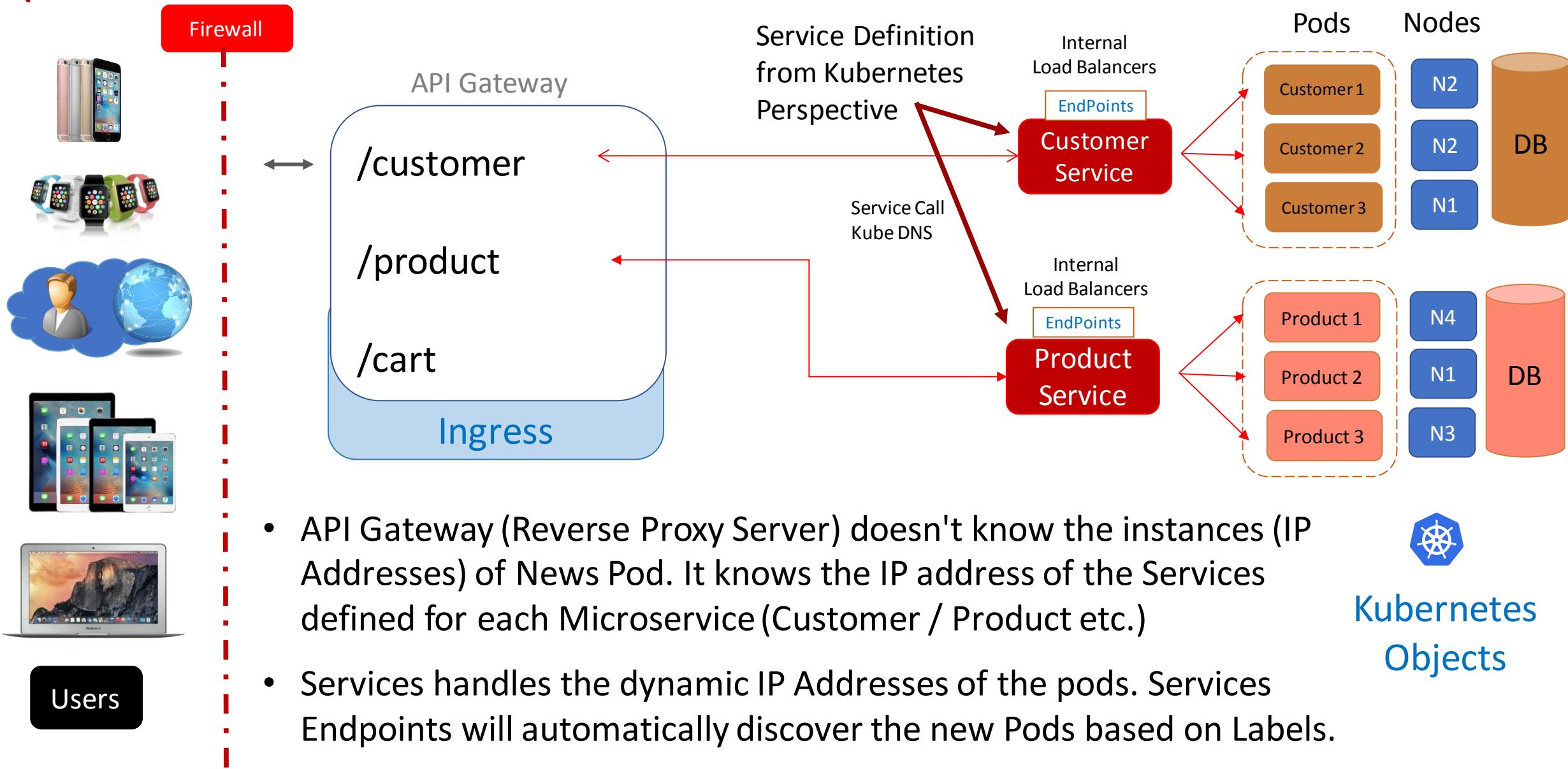
Load Balancer – Kubernetes Model



Service Discovery – NetFlix Network Stack Model

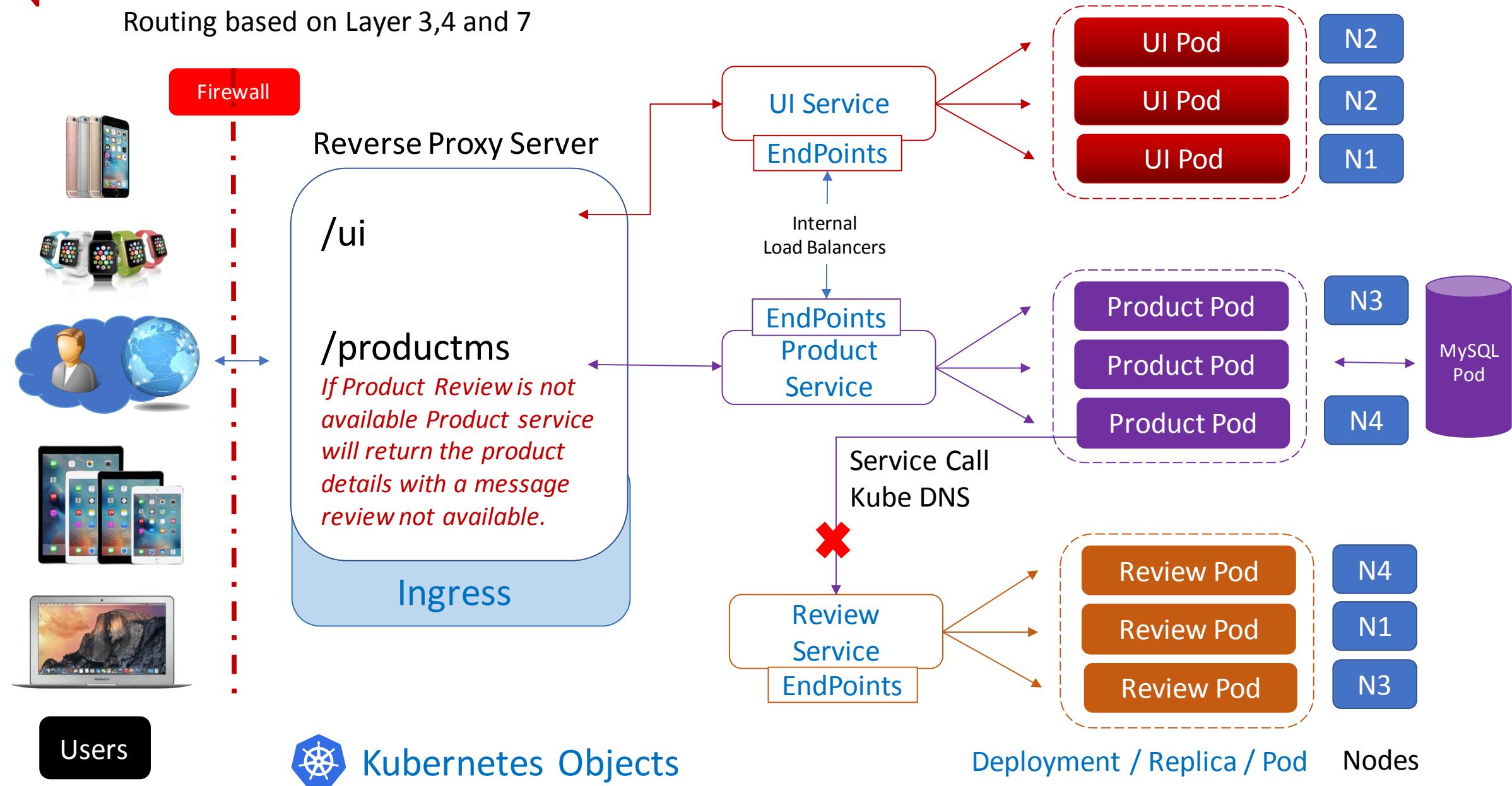


Service Discovery – Kubernetes Model

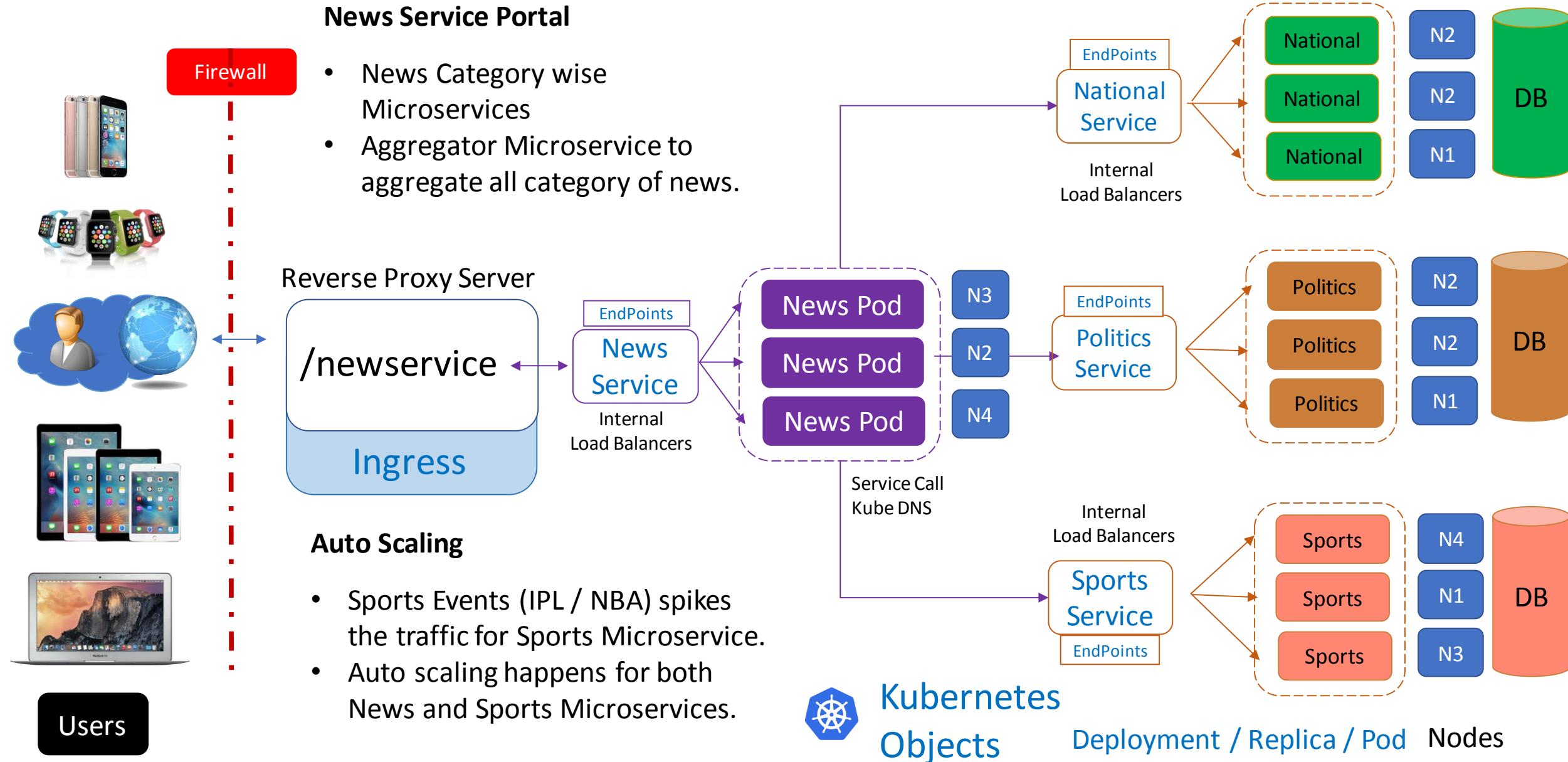


Circuit Breaker Pattern

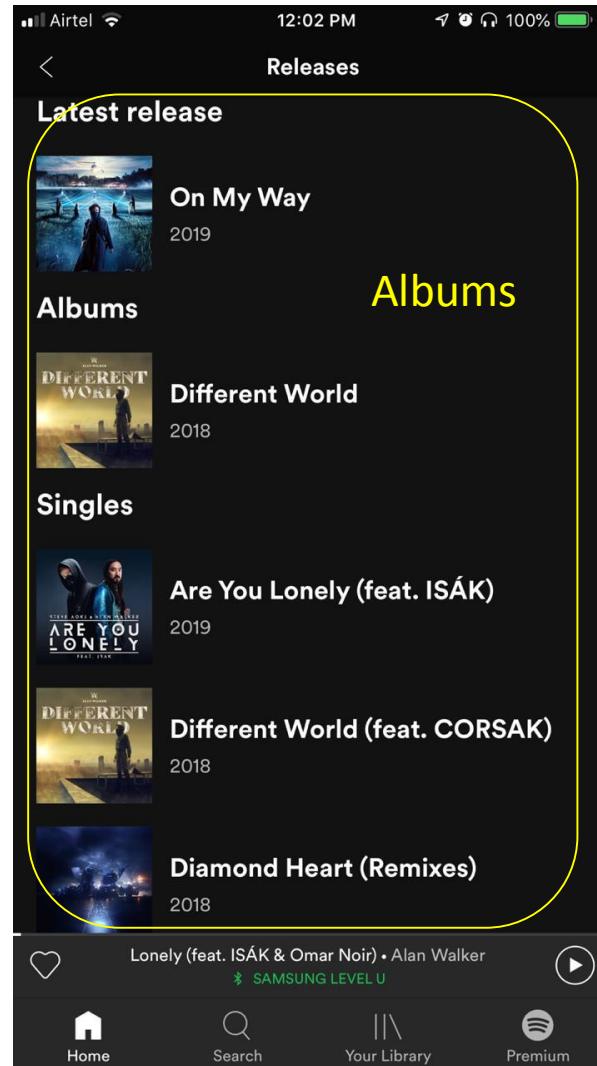
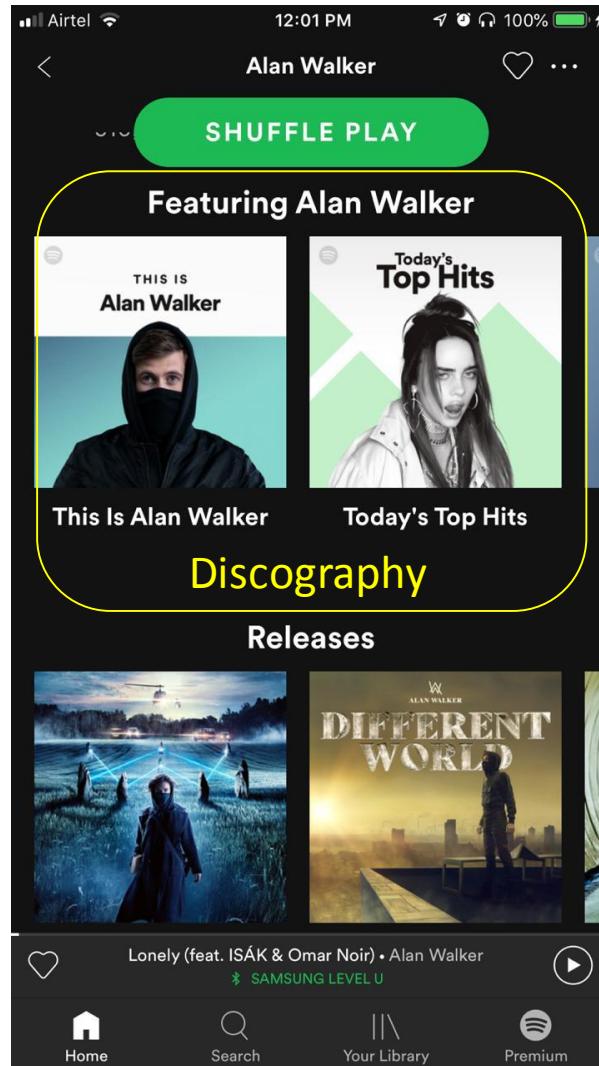
Routing based on Layer 3,4 and 7



Service Aggregator Pattern



Spotify® Music UI





Spotify® Service Aggregator Pattern

3

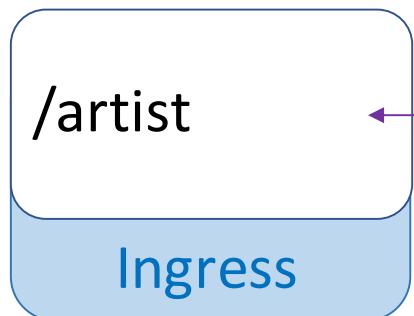


Firewall

Spotify Microservices

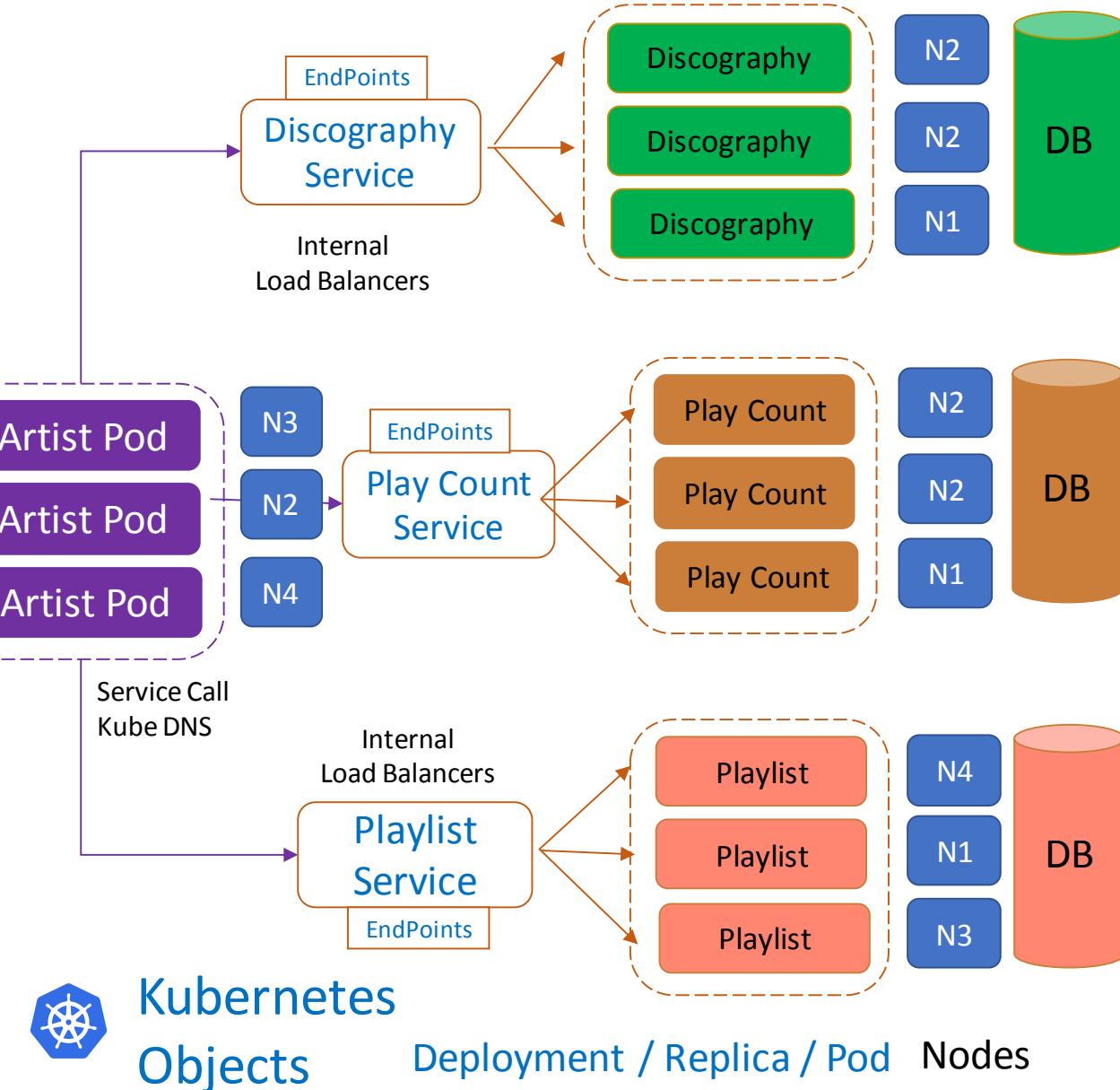
- Artist Microservice combines all the details from Discography, Play count and Playlists.

Reverse Proxy Server

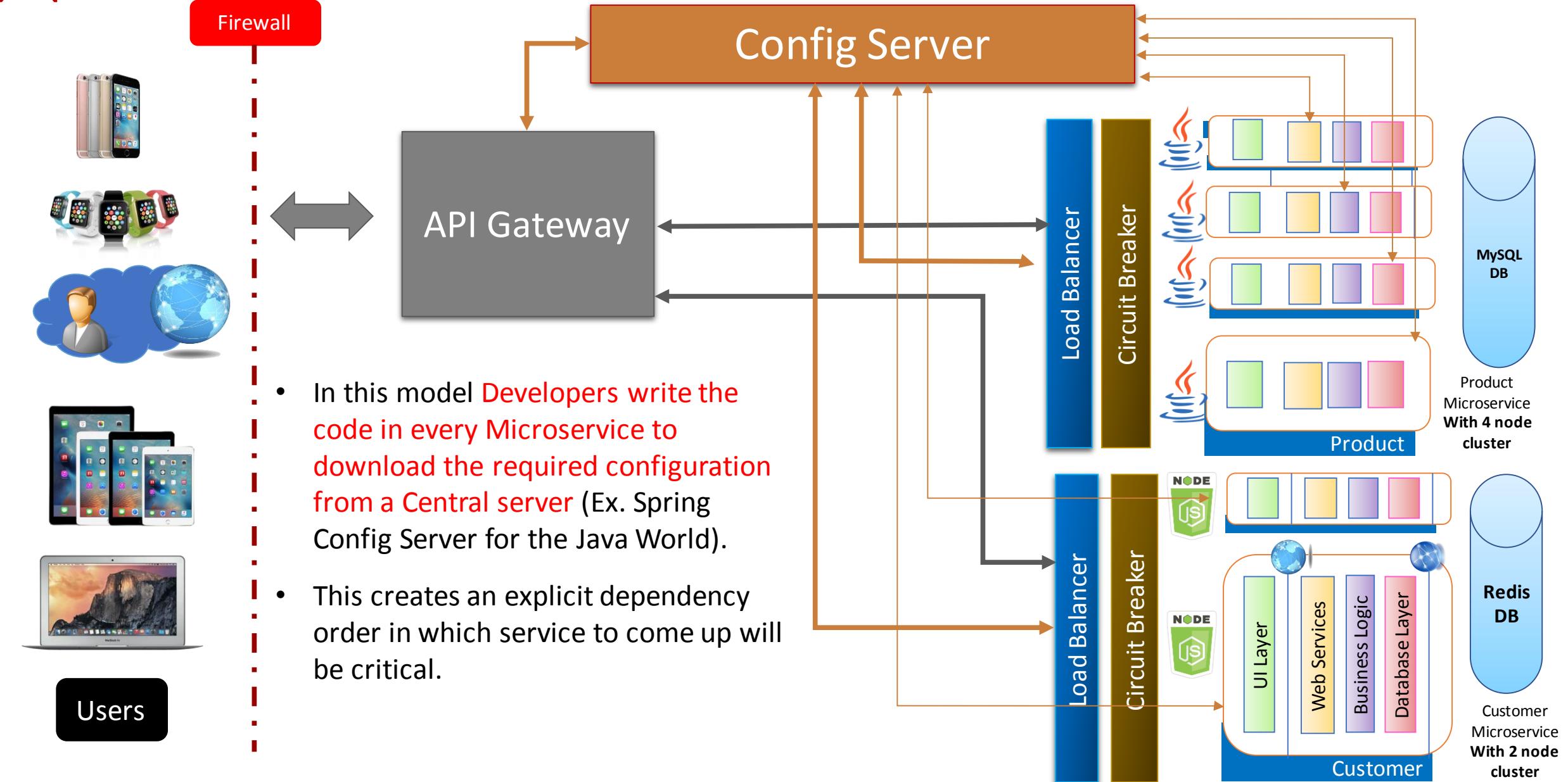


Auto Scaling

- Scaling of Artist and downstream Microservices will automatically scale depends on the load factor.



Config Store – Spring Config Server



Software Network Stack Vs Network Stack

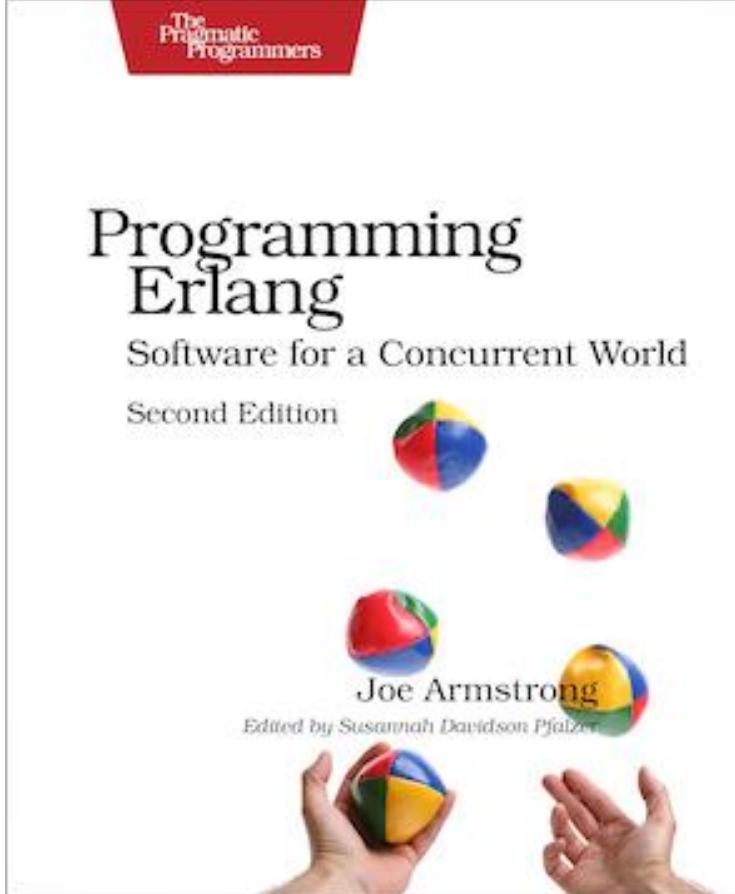
	Pattern	Software Stack Java	Software Stack .NET	Kubernetes
1	API Gateway	Zuul Server	SteelToe	Istio Envoy
2	Service Discovery	Eureka Server	SteelToe	Kube DNS
3	Load Balancer	Ribbon Server	SteelToe	Istio Envoy
4	Circuit Breaker	Hystrix	SteelToe	Istio
5	Config Server	Spring Config	SteelToe	Secrets, Env - K8s Master
	Web Site	https://netflix.github.io/	https://steeltoe.io/	https://kubernetes.io/

The Developer needs to write code to integrate with the Software Stack
 (Programming Language Specific. For Ex. Every microservice needs to subscribe to Service Discovery when the Microservice boots up.

Service Discovery in Kubernetes is based on the Labels assigned to Pod and Services and its Endpoints (IP Address) are dynamically mapped (DNS) based on the Label.

Let-it-Crash Design Pattern – Erlang Philosophy

- The Erlang view of the world is that everything is a process and that processes can interact only by exchanging messages.
- A typical Erlang program might have hundreds, thousands, or even millions of processes.
- Letting processes crash is central to Erlang. It's the equivalent of unplugging your router and plugging it back in – as long as you can get back to a known state, this turns out to be a very good strategy.
- To make that happen, you build supervision trees.
- A supervisor will decide how to deal with a crashed process. It will restart the process, or possibly kill some other processes, or crash and let someone else deal with it.
- Two models of concurrency: Shared State Concurrency, & Message Passing Concurrency. The programming world went one way (toward shared state). The Erlang community went the other way.
- All languages such as C, Java, C++, and so on, have the notion that there is this stuff called state and that we can change it. The moment you share something you need to bring Mutex a Locking Mechanism.
- Erlang has no mutable data structures (that's not quite true, but it's true enough). No mutable data structures = No locks. No mutable data structures = Easy to parallelize.



Let-it-Crash Design Pattern

1. The idea of **Messages as the first class citizens of a system**, has been rediscovered by the Event Sourcing / CQRS community, along with a strong focus on domain models.
2. **Event Sourced Aggregates** are a way to Model the Processes and NOT things.
3. Each component **MUST tolerate a crash and restart** at any point in time.
4. All interaction between the components must tolerate that peers can crash. This mean **ubiquitous use of timeouts and Circuit Breaker**.
5. Each component must be strongly encapsulated so that failures are fully contained and cannot spread.
6. All requests sent to a component **MUST be self describing** as is practical so that processing **can resume with a little recovery cost** as possible after a restart.

Let-it-Crash : Comparison Erlang Vs. Microservices Vs. Monolithic Apps

	Erlang Philosophy	Micro Services Architecture	Monolithic Apps (Java, C++, C#, Node JS ...)
1 Perspective	Everything is a Process	Event Sourced Aggregates are a way to model the Process and NOT things.	Things (defined as Objects) and Behaviors
2 Crash Recovery	Supervisor will decide how to handle the crashed process	Kubernetes Manager monitors all the Pods (Microservices) and its Readiness and Health. K8s terminates the Pod if the health is bad and spawns a new Pod. Circuit Breaker Pattern is used handle the fallback mechanism.	Not available. Most of the monolithic Apps are Stateful and Crash Recovery needs to be handled manually and all languages other than Erlang focuses on defensive programming.
3 Concurrency	Message Passing Concurrency	Domain Events for state changes within a Bounded Context & Integration Events for external Systems.	Mostly Shared State Concurrency
4 State	Stateless : Mostly Immutable Structures	Immutability is handled thru Event Sourcing along with Domain Events and Integration Events.	Predominantly Stateful with Mutable structures and Mutex as a Locking Mechanism
5 Citizen	Messages	Messages are 1 st class citizen by Event Sourcing / CQRS pattern with a strong focus on Domain Models	Mutable Objects and Strong focus on Domain Models and synchronous communication.

Infrastructure Design Patterns Summary

1. API Gateway
2. Service Discovery
3. Load Balancer
4. Circuit Breaker
5. Service Aggregator Pattern
6. Let It Crash Pattern



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.



Kubernetes Networking

3 Networks

Networks

1. Physical Network
2. Pod Network
3. Service Network

CIDR Range (RFC 1918)

1. 10.0.0.0/8
2. 172.0.0.0/11
3. 192.168.0.0/16

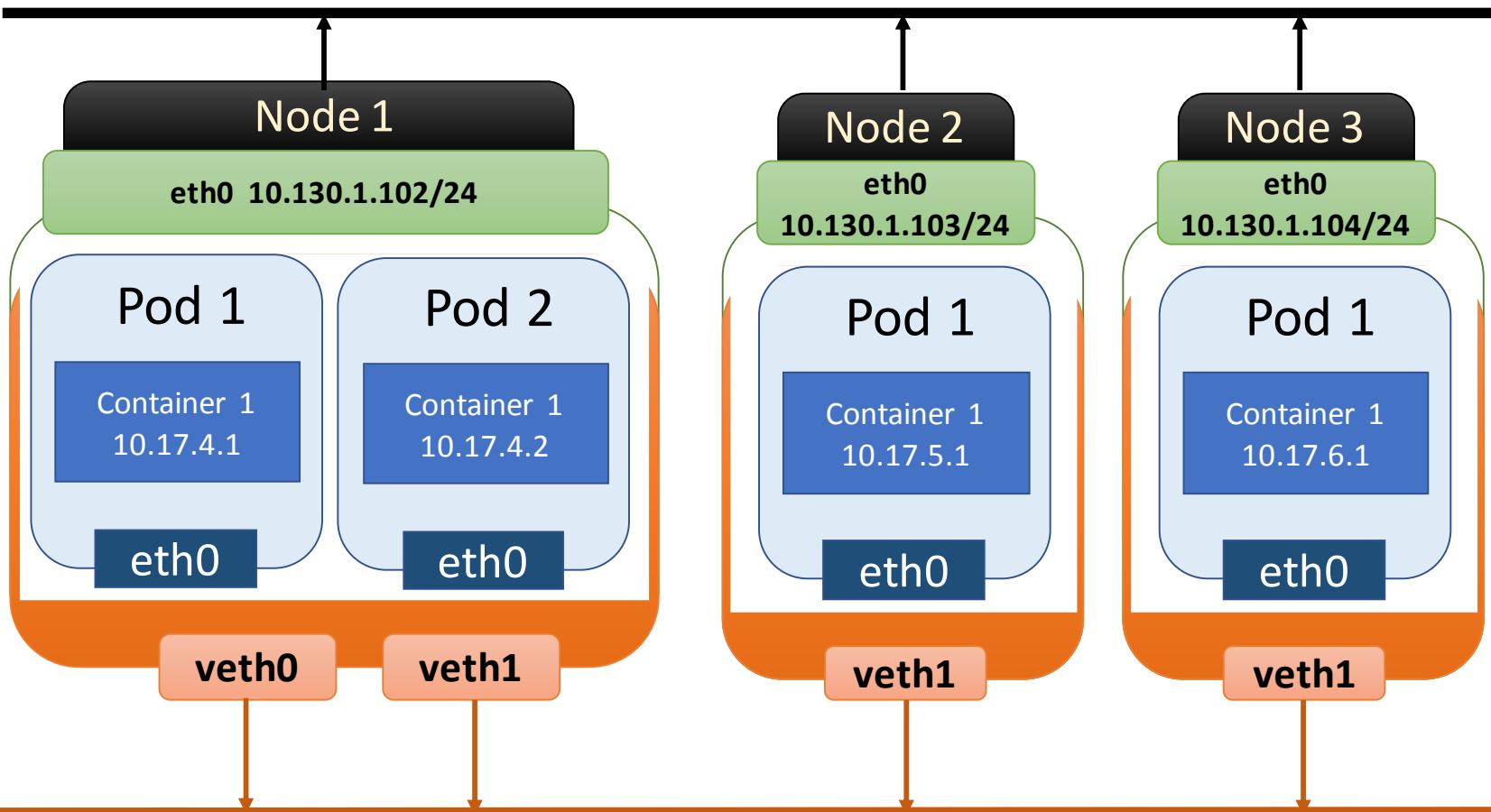
Keep the Address ranges separate.



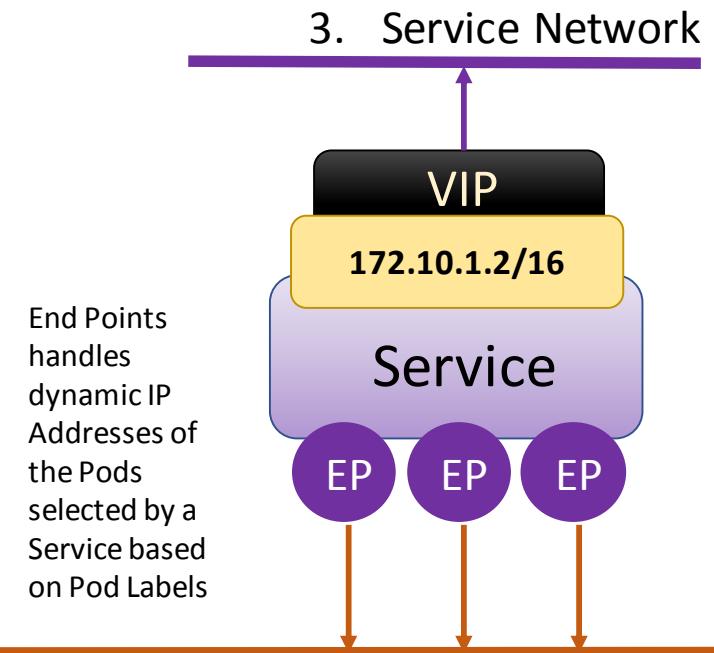
Kubernetes Networking

3 Networks

1. Physical Network



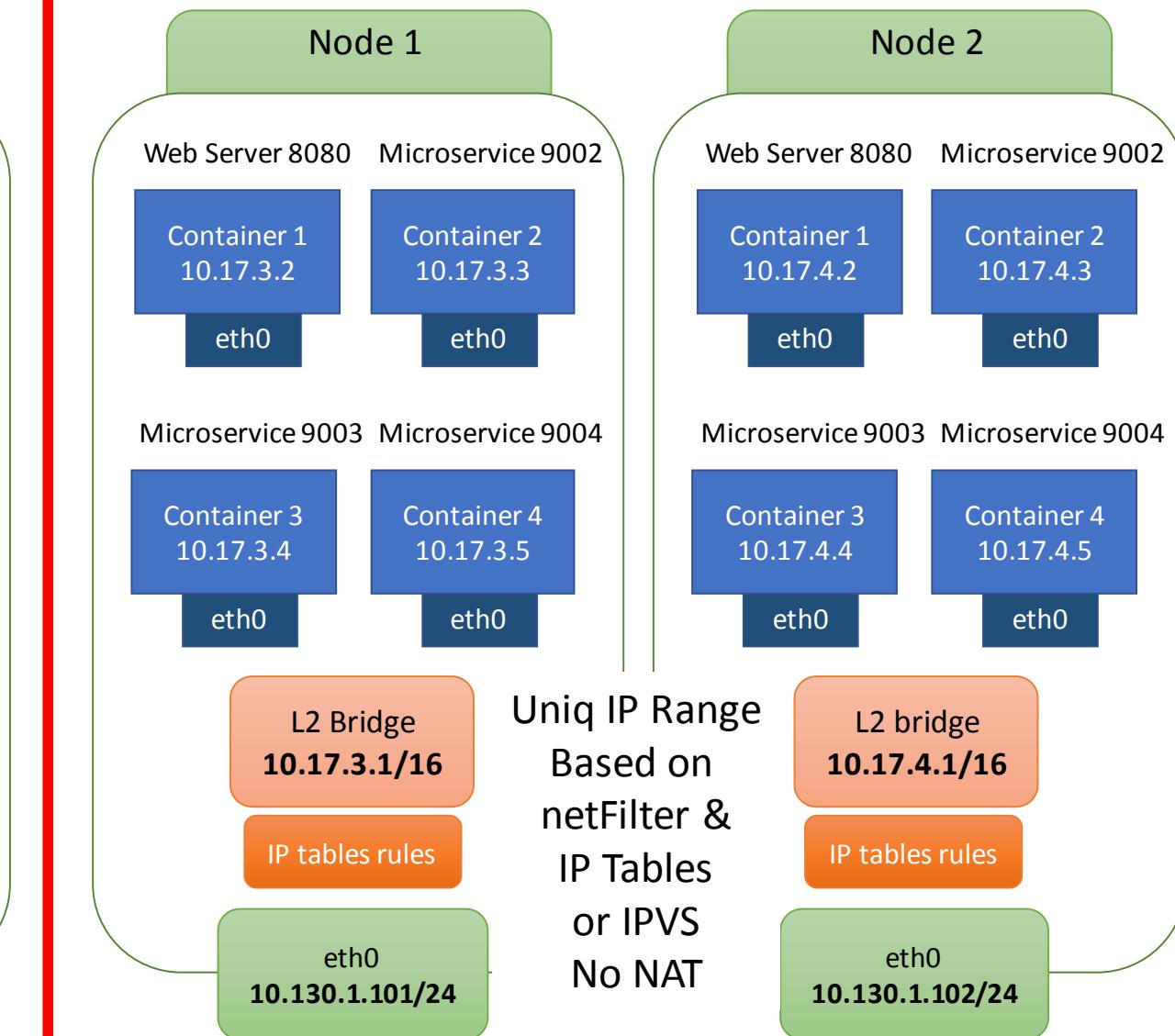
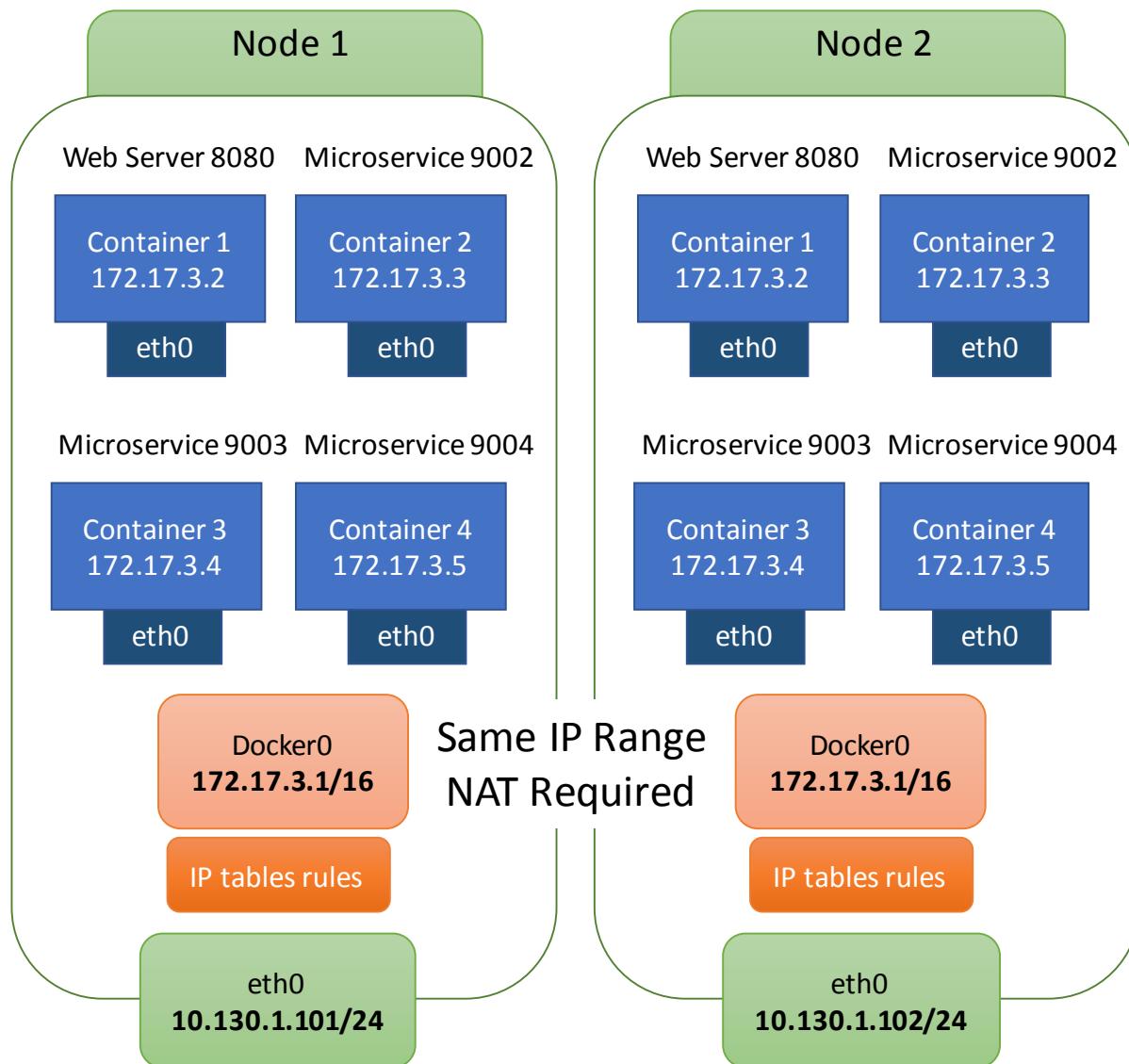
2. Pod Network



End Points handles dynamic IP Addresses of the Pods selected by a Service based on Pod Labels

Docker Networking

Vs. 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.

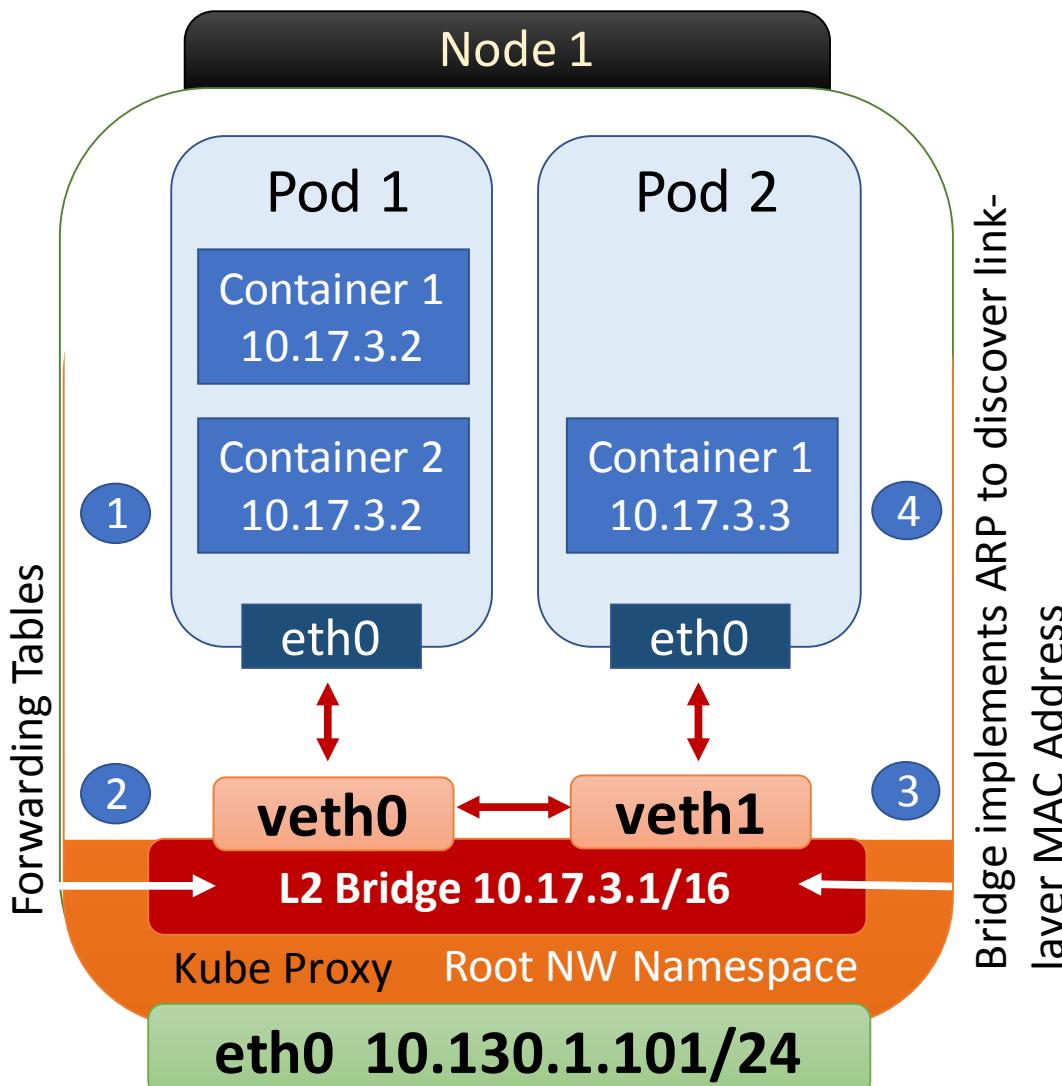
A DNS Pod consists of three separate containers

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.



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

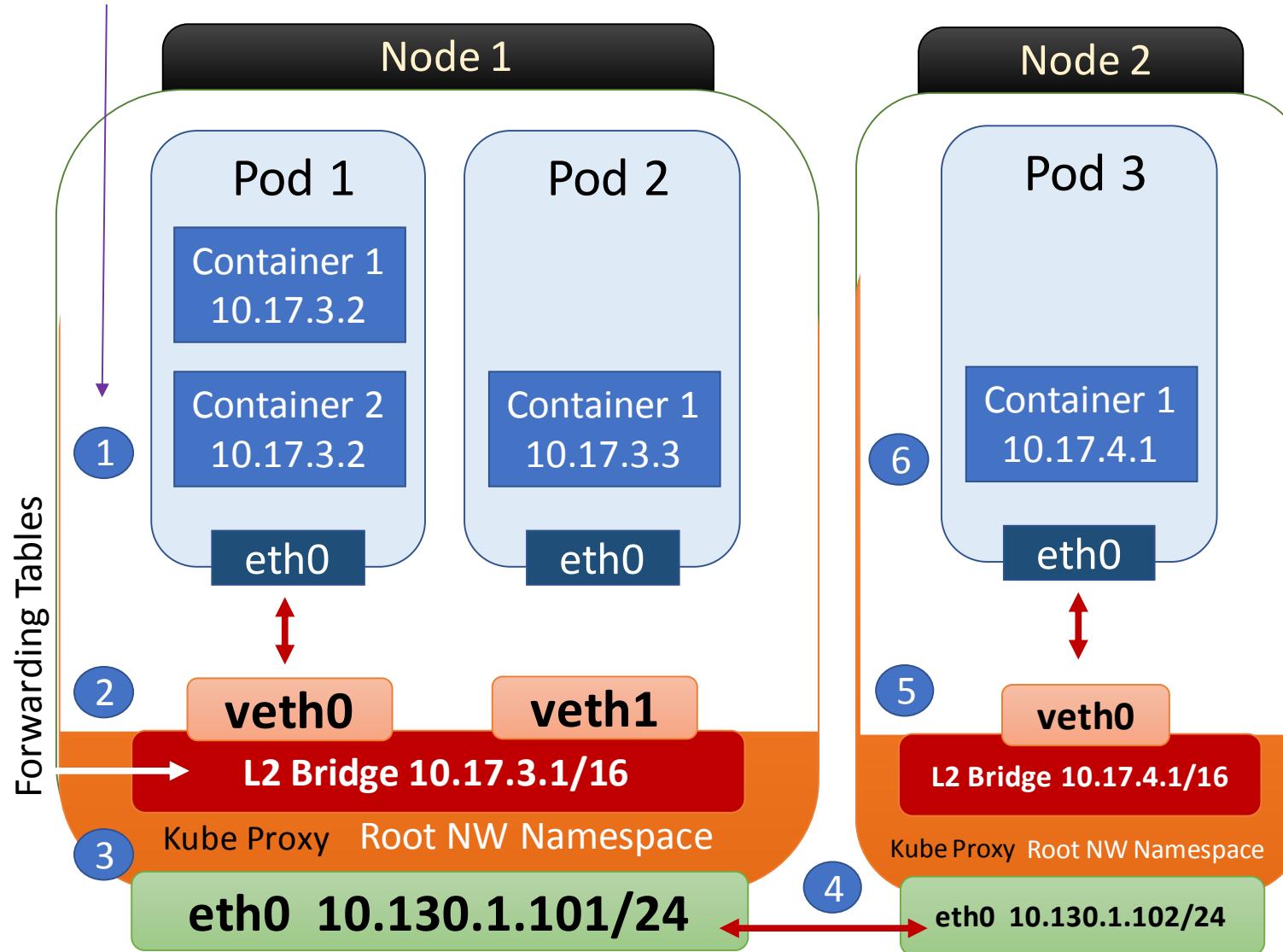
1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge resolves the Destination with ARP protocol and
3. Bridge sends the packet to veth1
4. veth1 forwards the packet directly to Pod 2 thru eth0

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

Src: Pod1 – Dst: Pod3

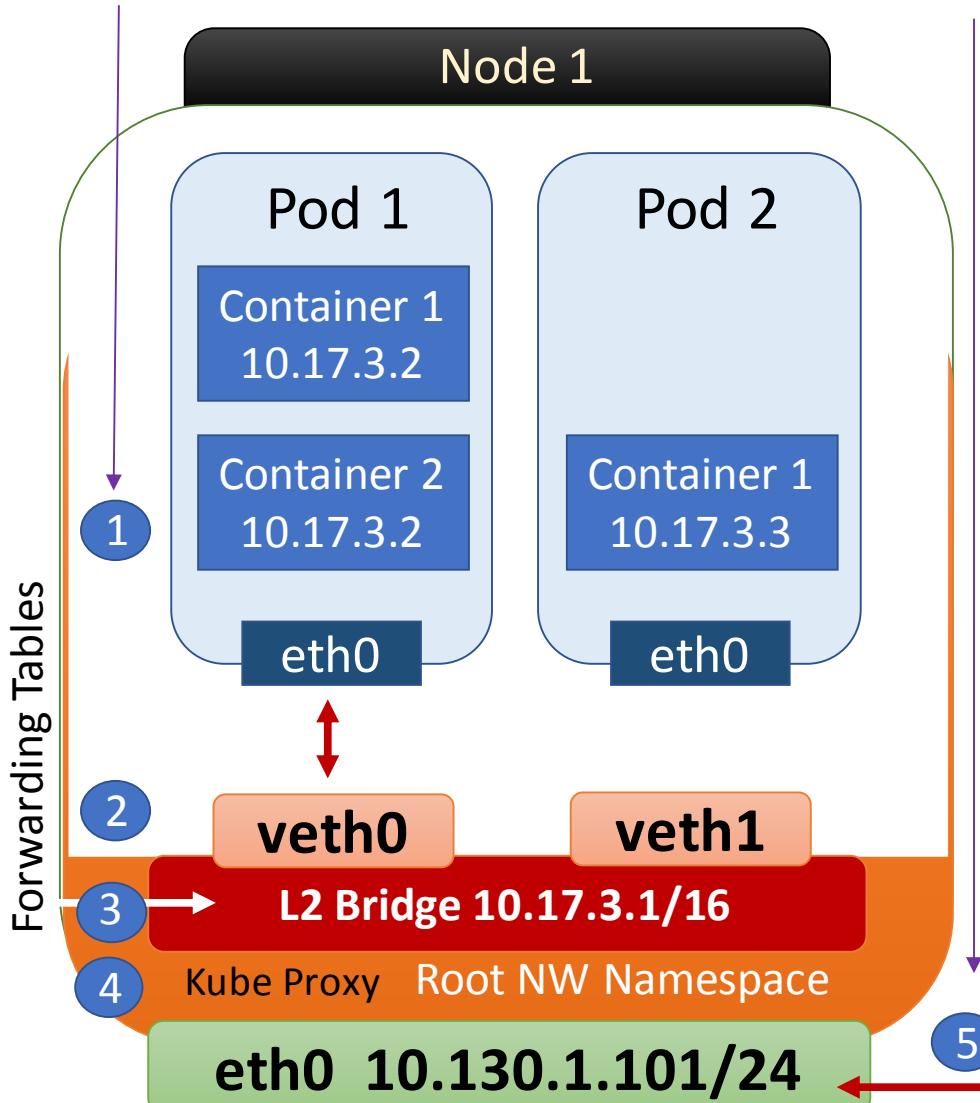


1. Pod 1 sends packet to eth0 – eth0 is connected to veth0
2. Bridge will try to resolve the Destination with ARP protocol and ARP will fail because there is no device connected to that IP.
3. On Failure Bridge will send the packet to eth0 of the Node 1.
4. At this point packet leaves eth0 and enters 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

Src: Pod1 – Dst: Service1

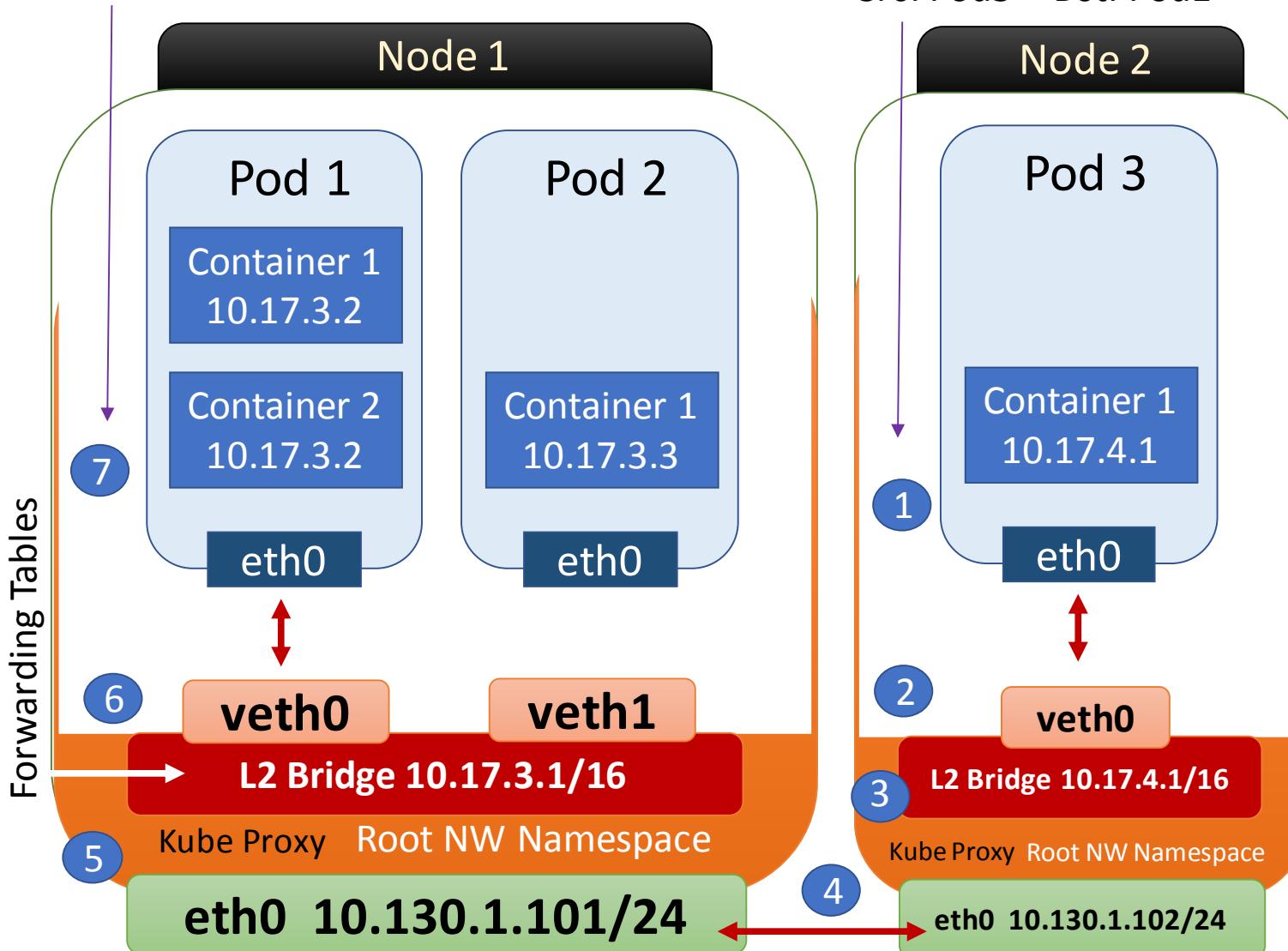


Src: Pod1 – Dst: Pod3

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

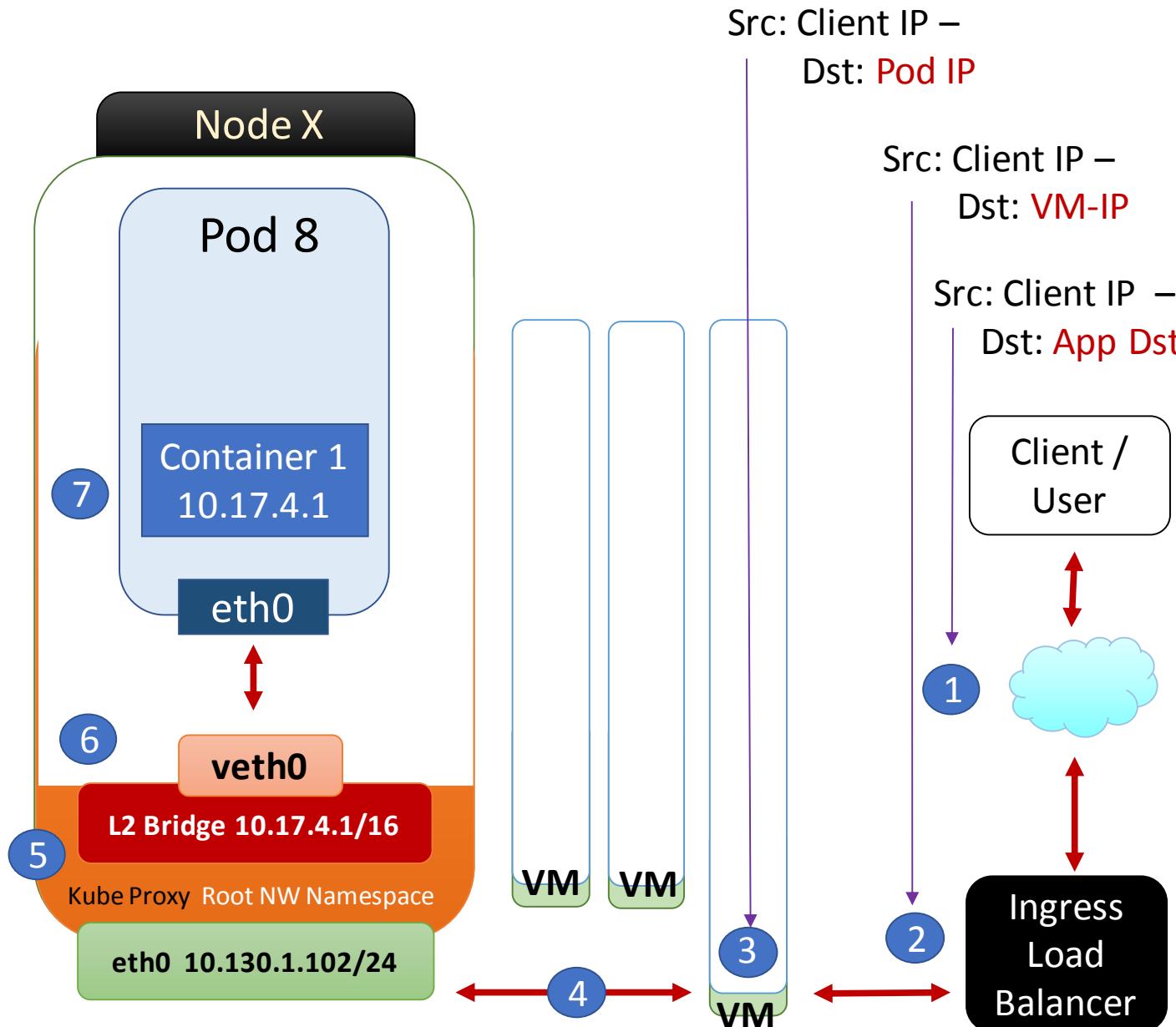
Src: Service1 – Dst: Pod1



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: Internet to Pod

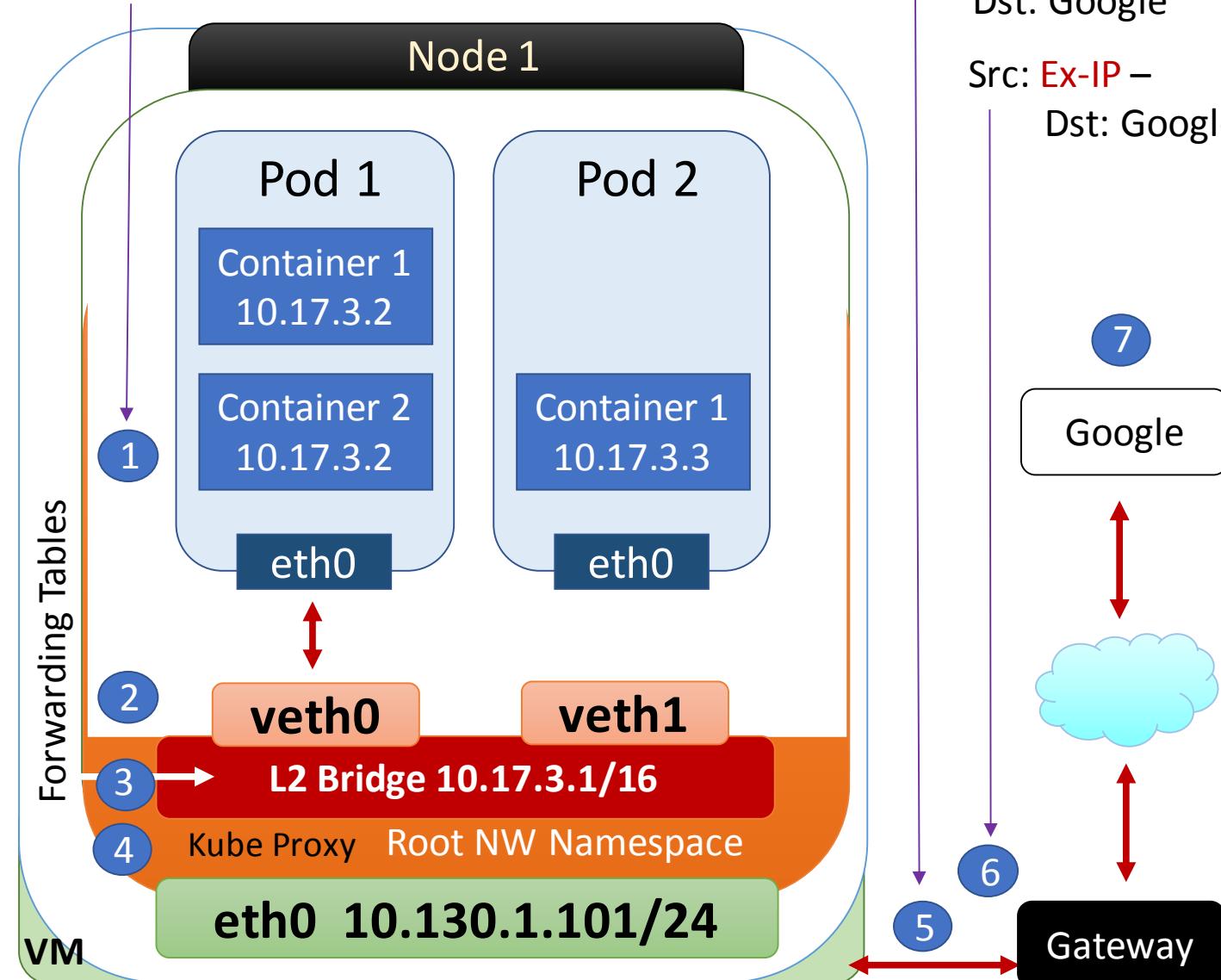


1. Client Connects to App published Domain.
2. Once the Load Balancer receives the packet it picks a VM.
3. Once inside the VM IP Tables knows how to redirect the packet to the Pod using internal load Balancing rules installed into the cluster using Kube Proxy.
4. Traffic enters Kubernetes cluster and 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



Kubernetes: Pod to Internet

Src: Pod1 – Dst: Google



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.



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 closest 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.



Kubernetes Network Policies

a4-Ingress-Allow-All.yaml x

```

1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: allow-all
5   namespace: sigma
6 spec:
7   podSelector: {}
8   ingress:
9     - {}
10

```

a4-Egress-Allow-All.yaml x

```

1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: allow-all
5   namespace: sigma
6 spec:
7   podSelector: {}
8   egress:
9     - {}
10  policyTypes:
11    - Egress
12

```

a4-Ingress-Deny-All.yaml x

```

1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: default-deny
5   namespace: sigma
6 spec:
7   podSelector: {}
8   policyTypes:
9     - Ingress
10

```

a4-Egress-Deny-All.yaml x

```

1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: default-deny
5   namespace: sigma
6 spec:
7   podSelector: {}
8   policyTypes:
9     - Egress
10

```

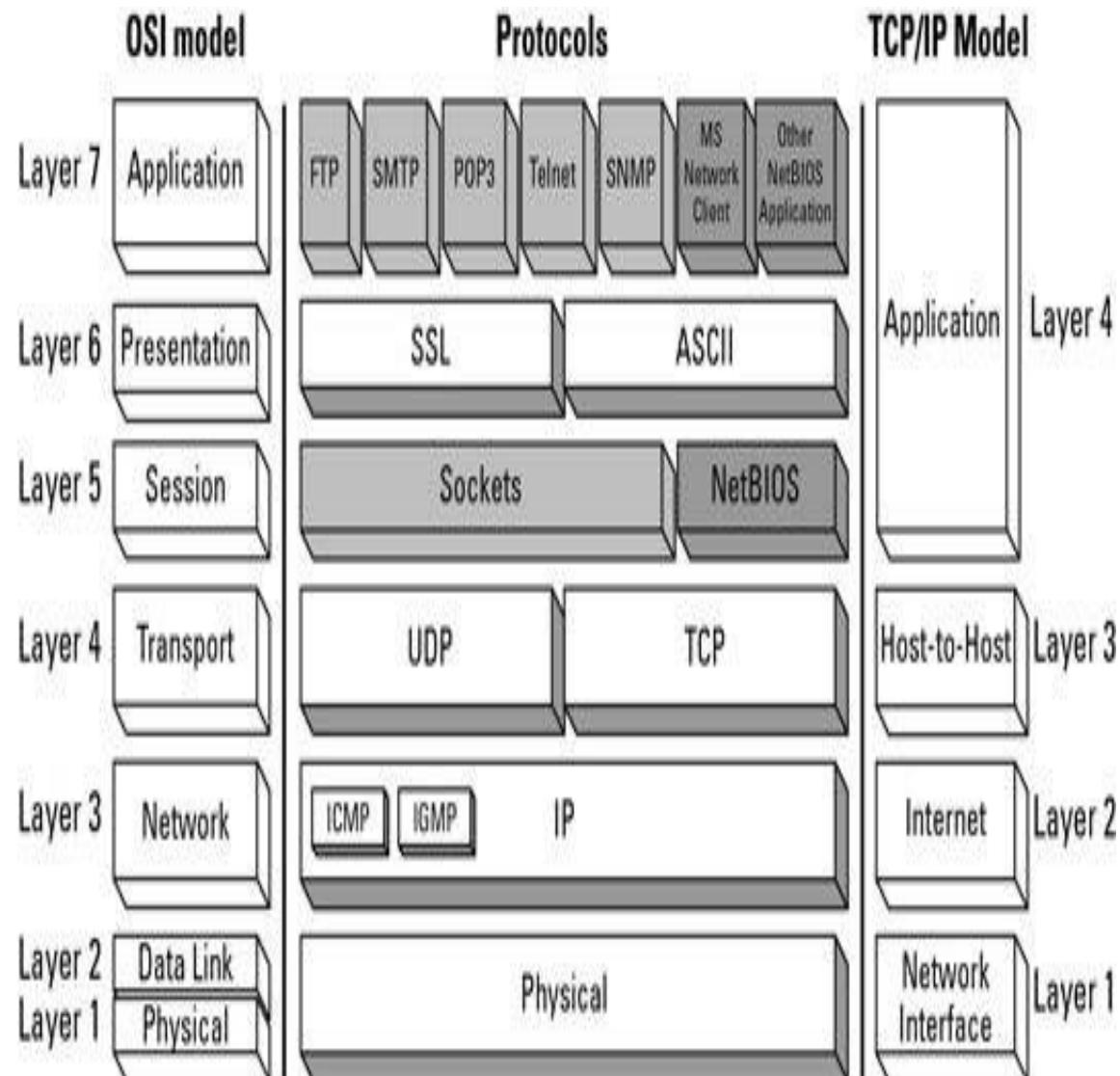
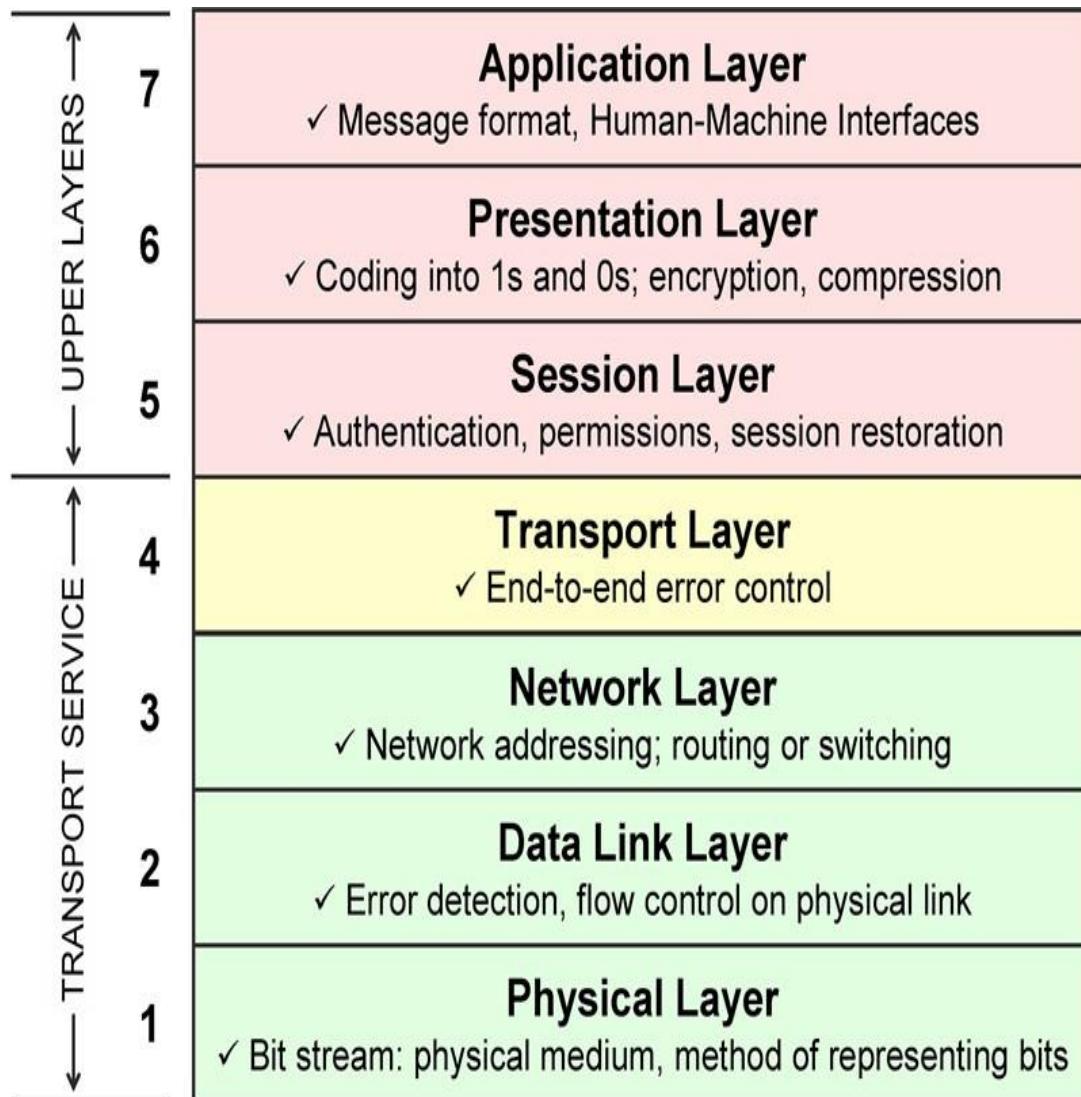
a4-networkpolicy.yaml x

```

1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: test-network-policy
5   namespace: sigma
6 spec:
7   podSelector:
8     matchLabels:
9       role: db
10    policyTypes:
11      - Ingress
12      - Egress
13    ingress:
14      - from:
15        - ipBlock:
16          cidr: 172.17.0.0/16
17          except:
18            - 172.17.1.0/24
19        - namespaceSelector:
20          matchLabels:
21            project: myproject
22      - podSelector:
23        matchLabels:
24          role: frontend
25    ports:
26      - protocol: TCP
27        port: 6379
28    egress:
29      - to:
30        - ipBlock:
31          cidr: 10.0.0.0/24
32    ports:
33      - protocol: TCP
34        port: 5978
35

```

OSI Layers





Service Mesh: Istio

- Gateway

- Virtual Service

- Destination Rule

- Service Entry

- Service Discovery

- Traffic Routing

- Security



Data 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

Istio Components



Control Plane

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.)

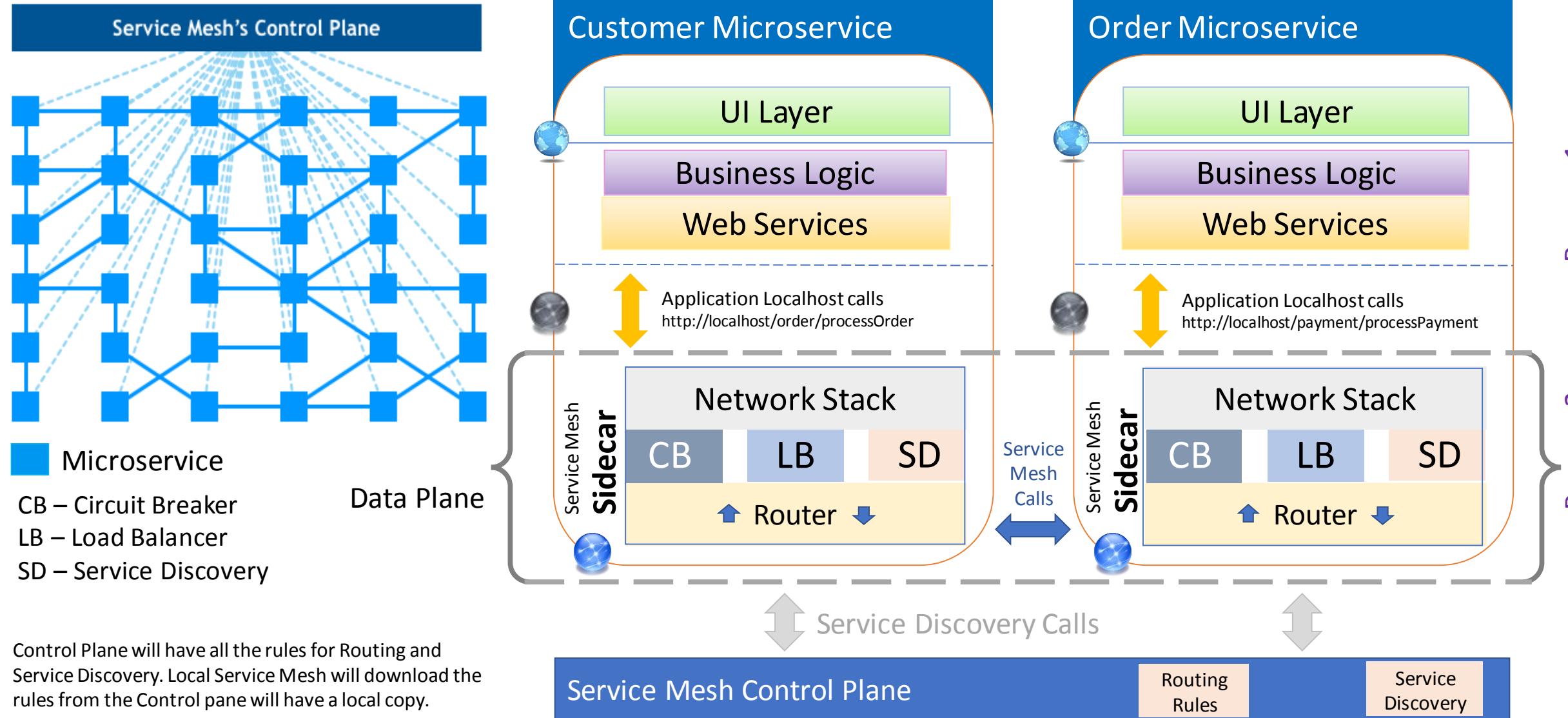
Citadel

- 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.

Galley

- Provides
- Configuration Injection
 - Processing and Distribution
- Component of Istio

Service Mesh – Sidecar Design Pattern

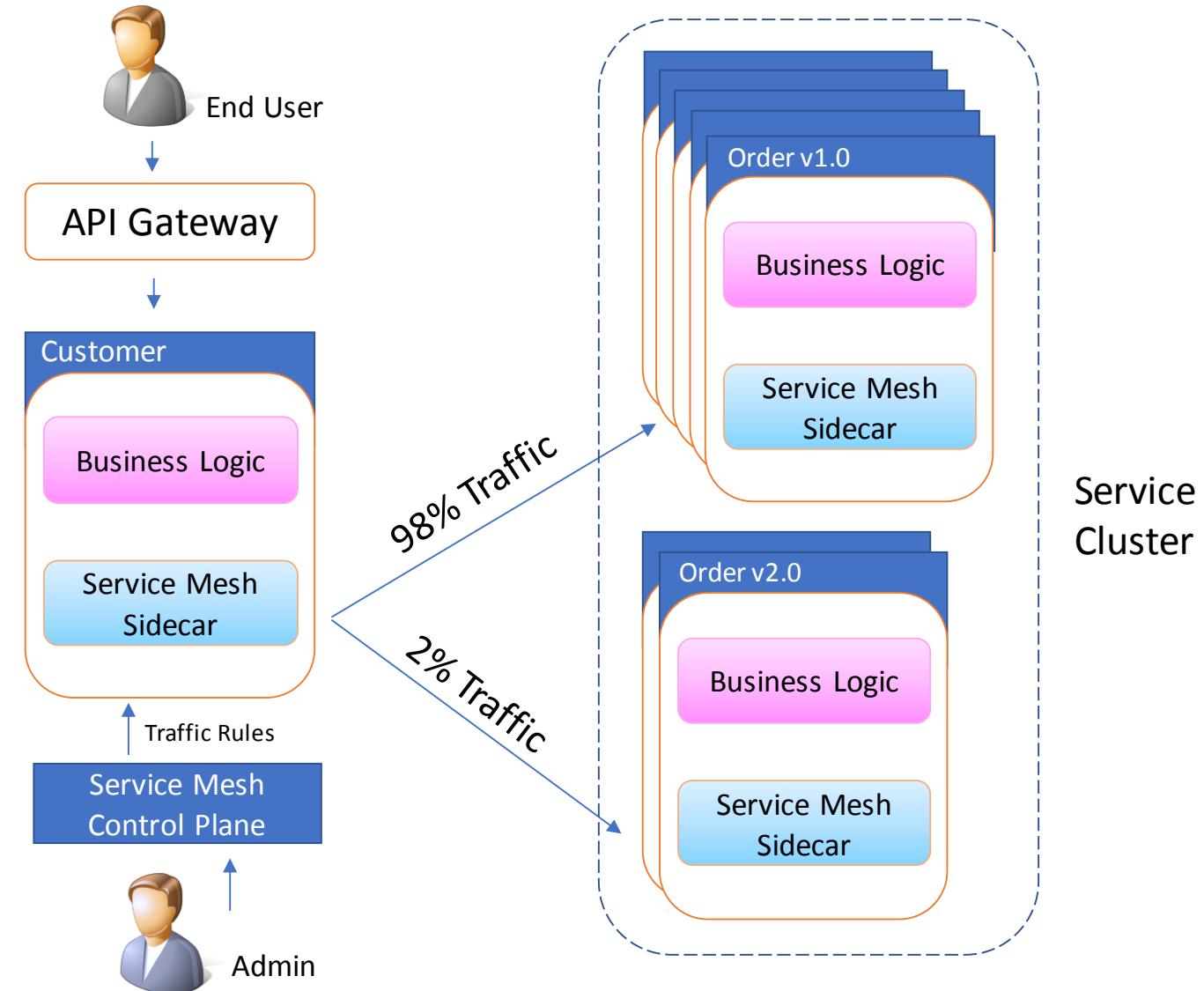


Control Plane will have all the rules for Routing and Service Discovery. Local Service Mesh will download the rules from the Control pane will have a local copy.

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





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



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
```



Kubernetes to Istio - Kinds

#	Kubernetes		
1	Ingress	GateWay	
		Virtual Service	
		Destination Rule	
2	Service	Service Entry	
	Service Account	ClusterRBACConfig	
		Mesh Policy	
		Policy	

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.

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

Gateway

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

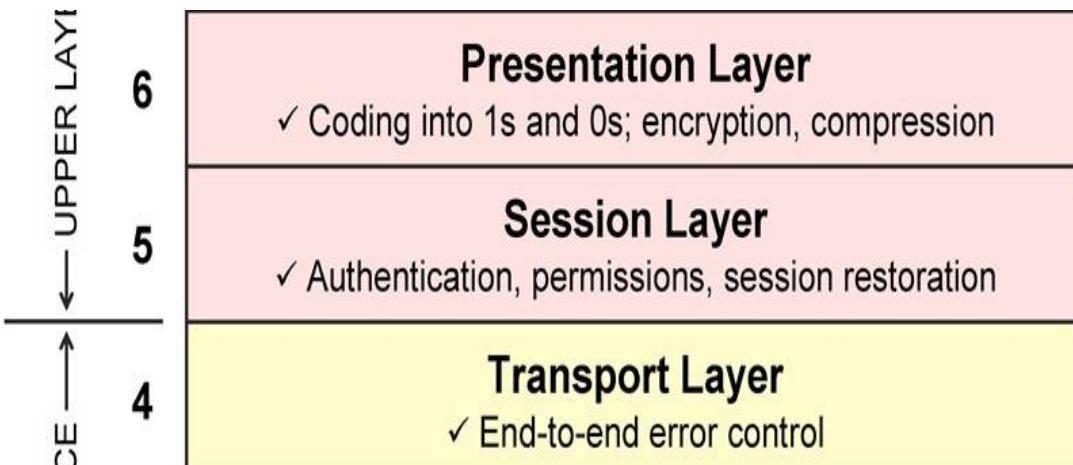
Policies

- Traffic Policies
 - Load Balancer

Istio Gateway

Gateway describes a load balancer operating at the edge of the mesh receiving incoming or outgoing HTTP/TCP connections.

- The Gateway specification above describes the L4-L6 properties of a load balancer.



```
shoppingportal-gw.yaml x
1 apiVersion: networking.istio.io/v1alpha3
2 kind: Gateway
3 metadata:
4   name: shoppingportal-gateway
5   namespace: shoppingportal
6 spec:
7   selector:
8     istio: ingressgateway # use istio default
9   servers:
10    - port:
11      number: 80
12      name: http
13      protocol: HTTP
14      hosts:
15        - "*"
```



Istio Gateway

In this Gateway configuration sets up a proxy to act as a load balancer exposing

- port 80 and
- 9080 (http),
- 443 (https),
- 9443(https)

for ingress.

Multiple Sub-domains are mapped to the single Load Balancer IP Address.

The same rule is also applicable inside the mesh for requests to the “reviews.prod.svc.cluster.local” service. This rule is applicable across ports 443, 9080. Note that http://in.shoppingportal.com gets redirected to https:// in.shoppingportal..com (i.e. 80 redirects to 443).

```
apiVersion: networking.istio.io/v1alpha3 kind: VirtualService metadata:
name: bookinfo-rule namespace: bookinfo-namespace spec: hosts: -
reviews.prod.svc.cluster.local
```

```
1 apiVersion: networking.istio.io/v1alpha3
2 kind: Gateway
3 metadata:
4   name: shoppingportal-gateway
5   namespace: shoppingportal
6 spec:
7   selector:
8     app: my-gateway-controller
9   servers:
10    - port:
11      number: 80
12      name: http
13      protocol: HTTP
14      hosts:
15        - in.shoppingportal.com
16        - us.shoppingportal.com
17      tls:
18        httpsRedirect: true # sends 301 redirect for http requ
19    - port:
20      number: 443
21      name: https-443
22      protocol: HTTPS
23      hosts:
24        - in.shoppingportal.com
25        - us.shoppingportal.com
26      tls:
27        mode: SIMPLE # enables HTTPS on this port
28        serverCertificate: /etc/certs/servercert.pem
29        privateKey: /etc/certs/privatekey.pem
30    - port:
31      number: 9443
32      name: https-9443
33      protocol: HTTPS
34      hosts:
35        - "shoppingportal-namespace/*.shoppingportal.com"
36      tls:
37        mode: SIMPLE # enables HTTPS on this port
38        credentialName: shoppingportal-secret # fetches certs
39    - port:
40      number: 9080
41      name: http-wildcard
42      protocol: HTTP
43      hosts:
44        - "*"
```

Both sub domains mapped to a single IP Address



Istio Virtual Service

The following VirtualService splits traffic for

- `https://in.shoppingportal.com/reviews`,
- `https://us.shoppingportal.com/reviews`,
- `http://in.shoppingportal.com:9080/reviews`,
- `http://in.shoppingportal.com:9080/reviews`
- into two versions (prod and qa) of an internal reviews service on port 9080.

In addition, requests containing the cookie “`user: dev-610`” will be sent to special port 7777 in the qa version

```

istio-Virtual-Service.yaml x istio-Gateway.yaml x
1 apiVersion: networking.istio.io/v1alpha3
2 kind: VirtualService
3 metadata:
4   name: shoppingportal-vs
5   namespace: shoppingportal
6 spec:
7   hosts:
8     - reviews.prod.svc.cluster.local
9     - in.shoppingportal.com
10    - us.shoppingportal.com
11   gateways:
12     - shoppingportal/shoppingportal-gateway
13   http:
14     - match:
15       - headers:
16         cookie:
17           exact: "user=dev-610"
18       route:
19         - destination:
20           port:
21             number: 5129
22             host: reviews.qa.svc.cluster.local
23       - match:
24         - uri:
25           prefix: /reviews/
26         route:
27           - destination:
28             port:
29               number: 9080 # can be omitted if it's the
30               host: reviews.prod.svc.cluster.local
31             weight: 80
32           - destination:
33             host: reviews.qa.svc.cluster.local
34             weight: 2
35
36

```

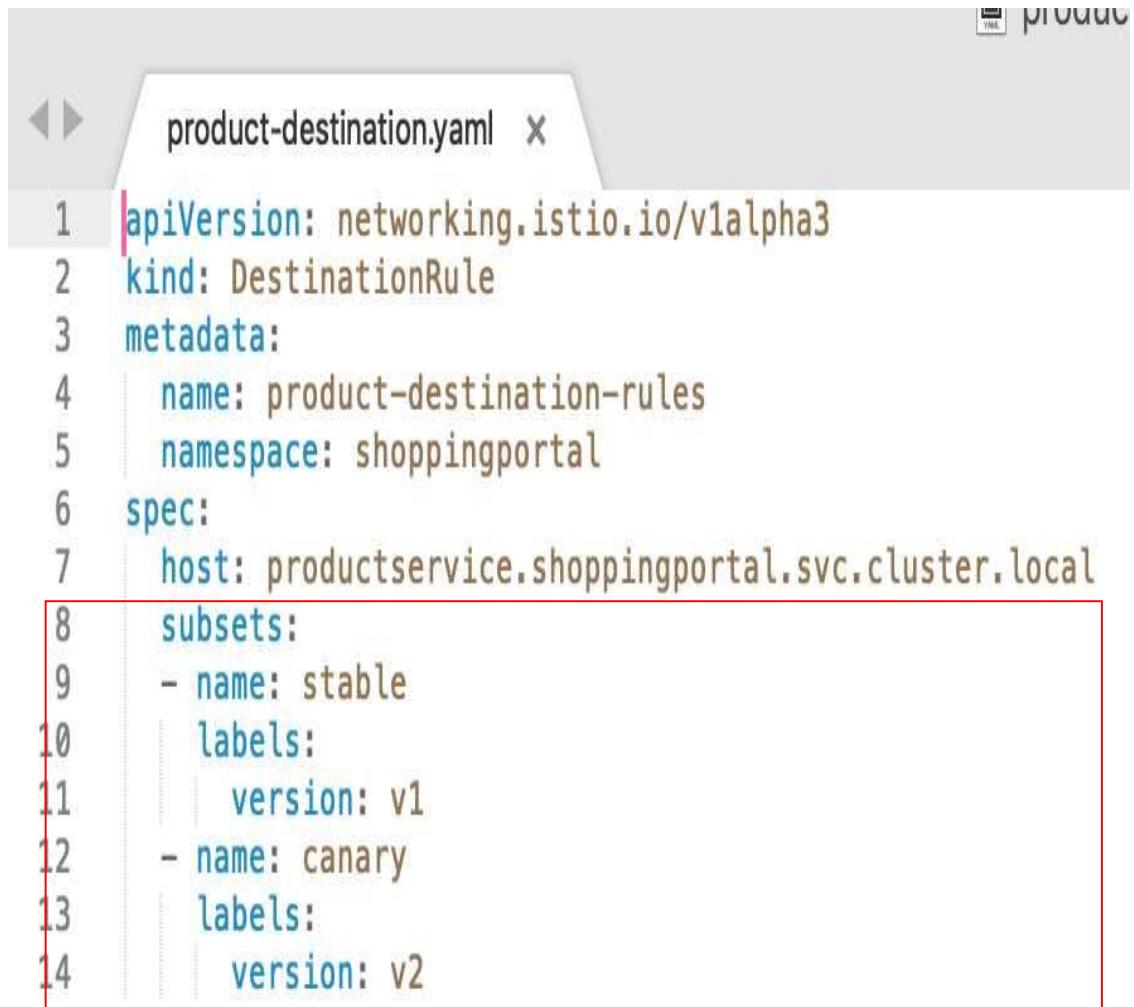
 Istio Virtual Service

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

```
shoppingportal-virtualservice.yaml ×
1 apiVersion: networking.istio.io/v1alpha3
2 kind: VirtualService
3 metadata:
4   name: shoppingportal-vs
5   namespace: shoppingportal
6 spec:
7   gateways:
8     - shoppingportal-gateway
9   hosts:
10    - "*"
11   http:
12     - match:
13       - uri:
14         prefix: /ui
15         headers:
16           user-agent:
17             regex: ".*Firefox.*"
18       route:
19         - destination:
20           host: k8uiworkshopservice.shoppingportal.svc.cluster.local
21           subset: canary # match v2 only
22           port:
23             number: 80
24     - match:
25       - uri:
26         prefix: /ui
27         route:
28           - destination:
29             host: k8uiworkshopservice.shoppingportal.svc.cluster.local
30             subset: stable # match v2 only
31             port:
32               number: 80
33     - match:
34       - uri:
35         prefix: /productms
36         headers:
37           end-user:
38             exact: metamagic
39         route:
40           - destination:
41             host: productservice.shoppingportal.svc.cluster.local
42             subset: canary # match v2 only
43             port:
44               number: 80
45     - match:
46       - uri:
47         prefix: /productms
48         route:
49           - destination:
50             host: productservice.shoppingportal.svc.cluster.local
51             subset: stable # match v2 only
52             port:
53               number: 80
54     - match:
55       - uri:
56         prefix: /productreviewms
57         route:
58           - destination:
59             host: productreviewservice.shoppingportal.svc.cluster.local
60             port:
61               number: 80
62
```

Istio Destination Rule

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



```
product-destination.yaml
1 apiVersion: networking.istio.io/v1alpha3
2 kind: DestinationRule
3 metadata:
4   name: product-destination-rules
5   namespace: shoppingportal
6 spec:
7   host: productservice.shoppingportal.svc.cluster.local
8   subsets:
9     - name: stable
10       labels:
11         version: v1
12     - name: canary
13       labels:
14         version: v2
```

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



Istio ServiceEntry

A service entry describes the properties of a service

- DNS name,
- VIPs (Virtual IPs)
- ports, protocols
- endpoints

For HTTP-based services, it is possible to create a `VirtualService` backed by multiple DNS addressable endpoints. In such a scenario, the application can use the `HTTP_PROXY` environment variable to transparently reroute API calls for the `VirtualService` to a chosen backend.

For example, the following configuration

- creates a non-existent external service called `foo.bar.com` backed by three domains:
 - `us.foo.bar.com:8080`,
 - `uk.foo.bar.com:9080`, and
 - `in.foo.bar.com:7080`

`MESH_EXTERNAL`

Signifies that the service is external to the mesh.

Typically used to indicate external services consumed through APIs.

`MESH_INTERNAL`

Signifies that the service is part of the mesh.

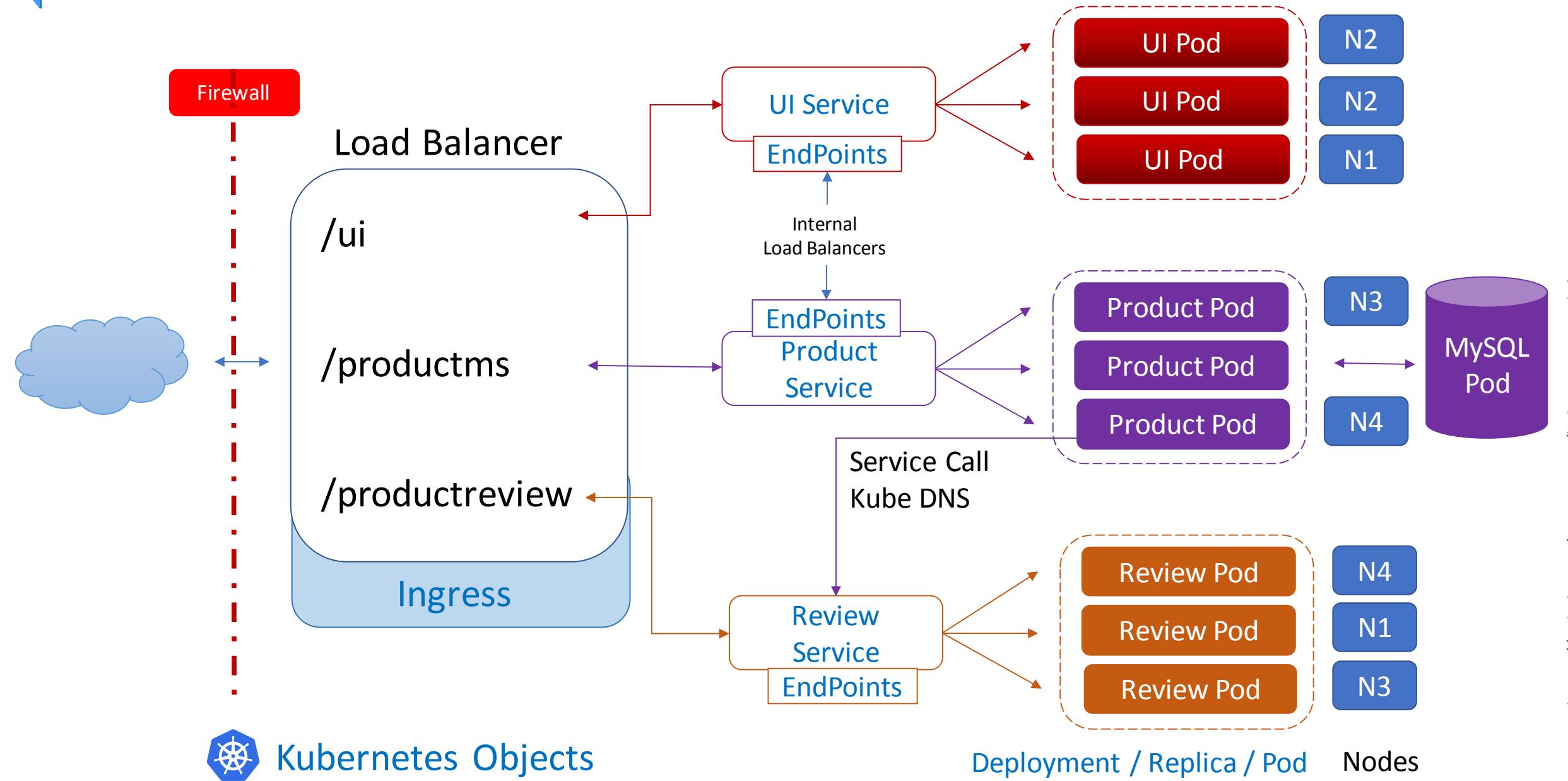
```

istio-ServiceEntry.yaml x
1 apiVersion: networking.istio.io/v1alpha3
2 kind: ServiceEntry
3 metadata:
4   name: external-svc-dns
5 spec:
6   hosts:
7     - foo.bar.com
8   location: MESH_EXTERNAL
9   ports:
10    - number: 80
11      name: http
12      protocol: HTTP
13      resolution: DNS
14   endpoints:
15     - address: us.foo.bar.com
16       ports:
17         https: 8080
18     - address: uk.foo.bar.com
19       ports:
20         https: 9080
21     - address: in.foo.bar.com
22       ports:
23         https: 7080

```

Resolution determines how the proxy will resolve the IP addresses of the network endpoints associated with the service, so that it can route to one of them. Values: DNS : Static : None

Shopping Portal – Docker / Kubernetes



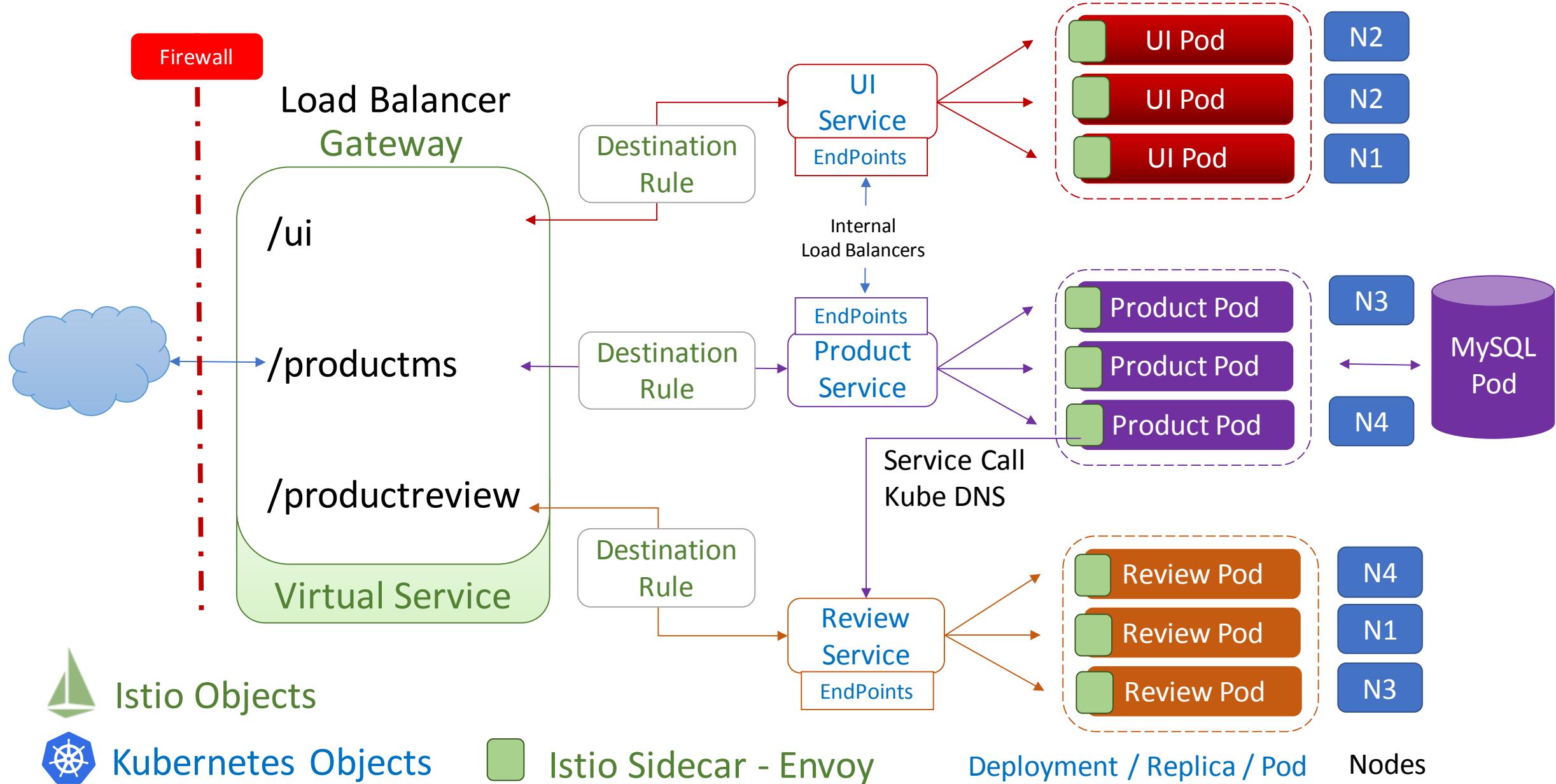
Shopping Portal - Istio

Istio Control Plane

Pilot

Mixer

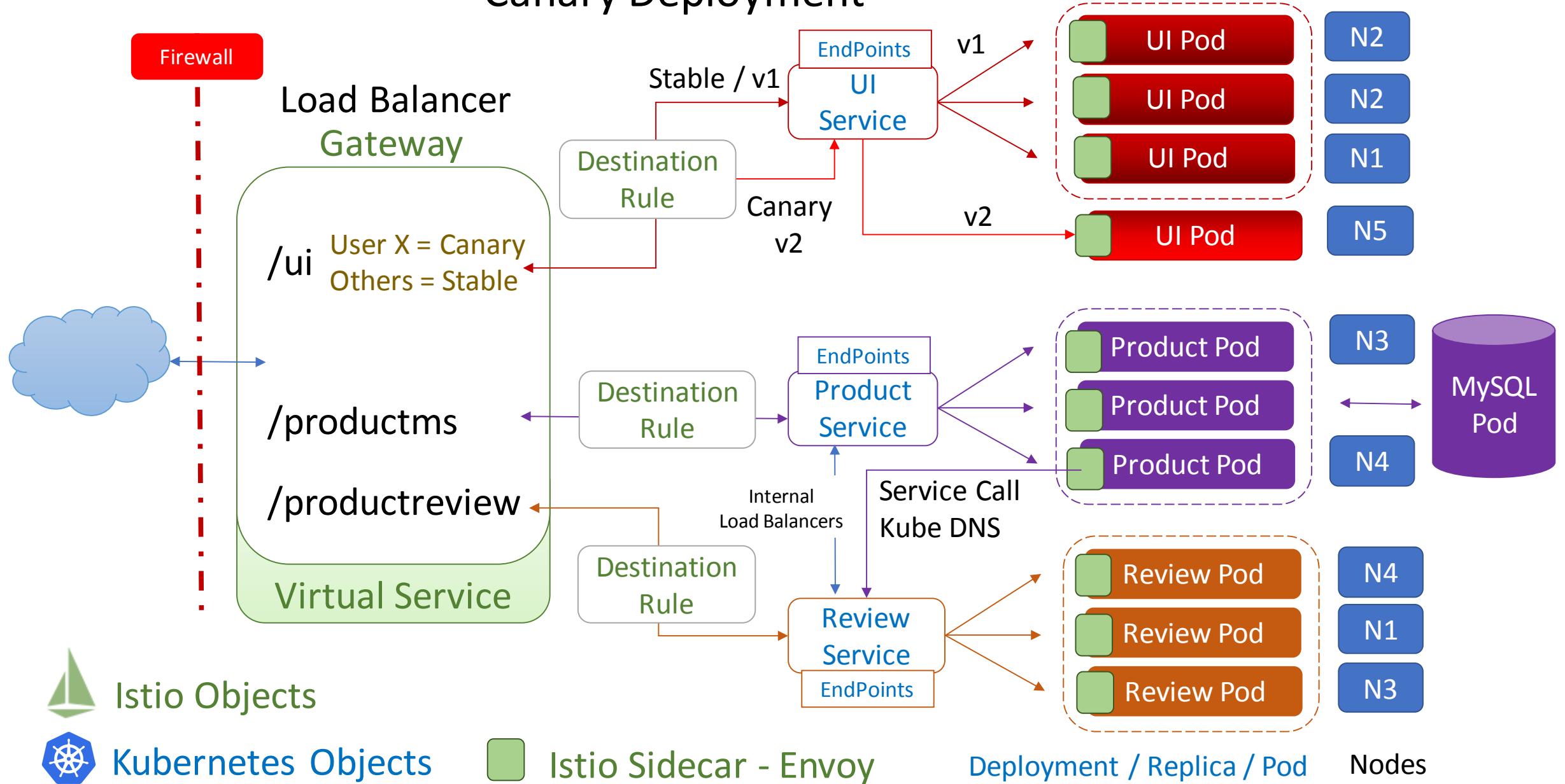
Citadel



Shopping Portal A / B Testing using Canary Deployment

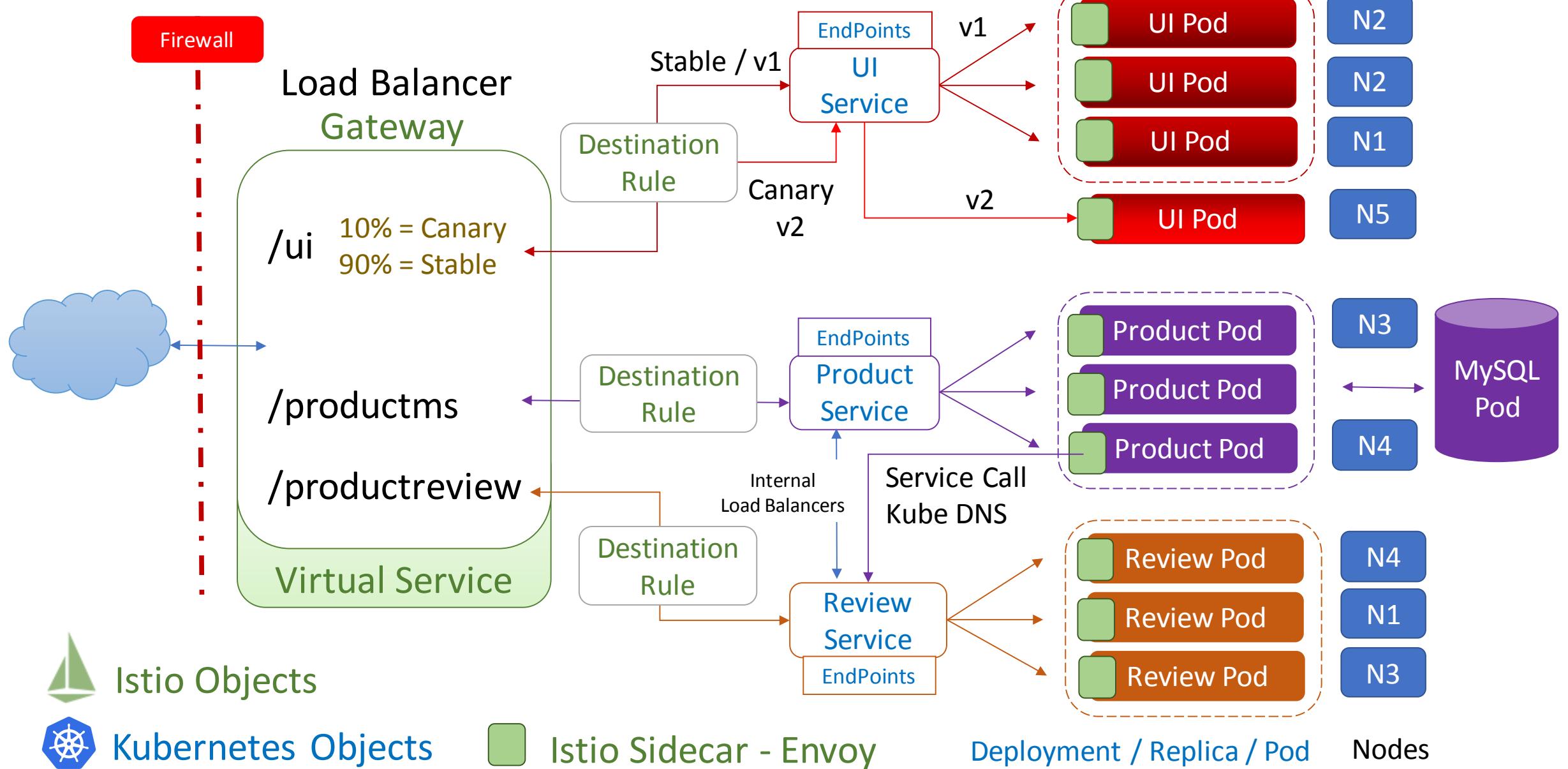
Istio Control Plane

P M C



Shopping Portal Traffic Shifting Canary Deployment

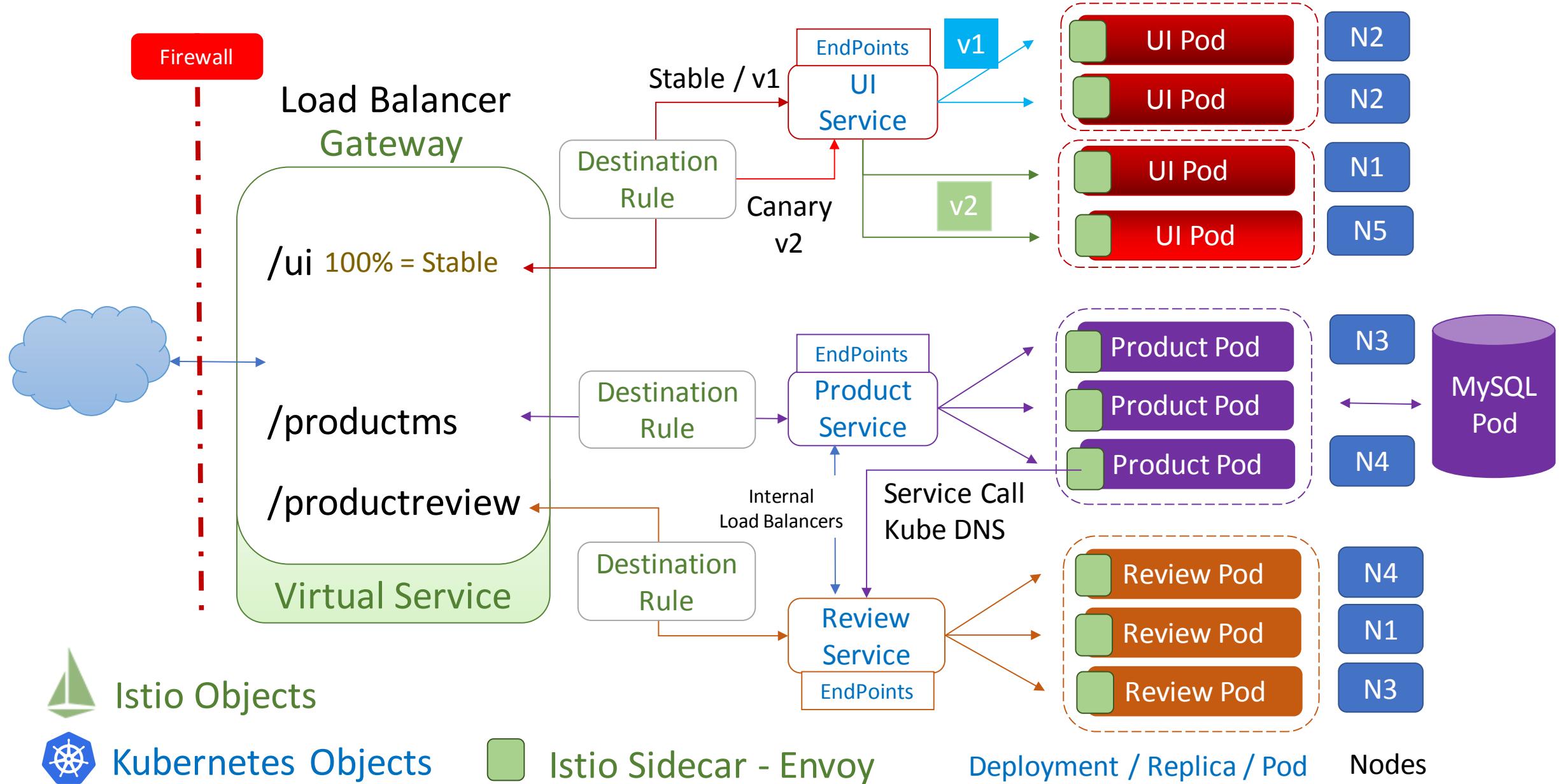
Istio Control Plane P M C



Shopping Portal Blue Green Deployment

Istio Control Plane

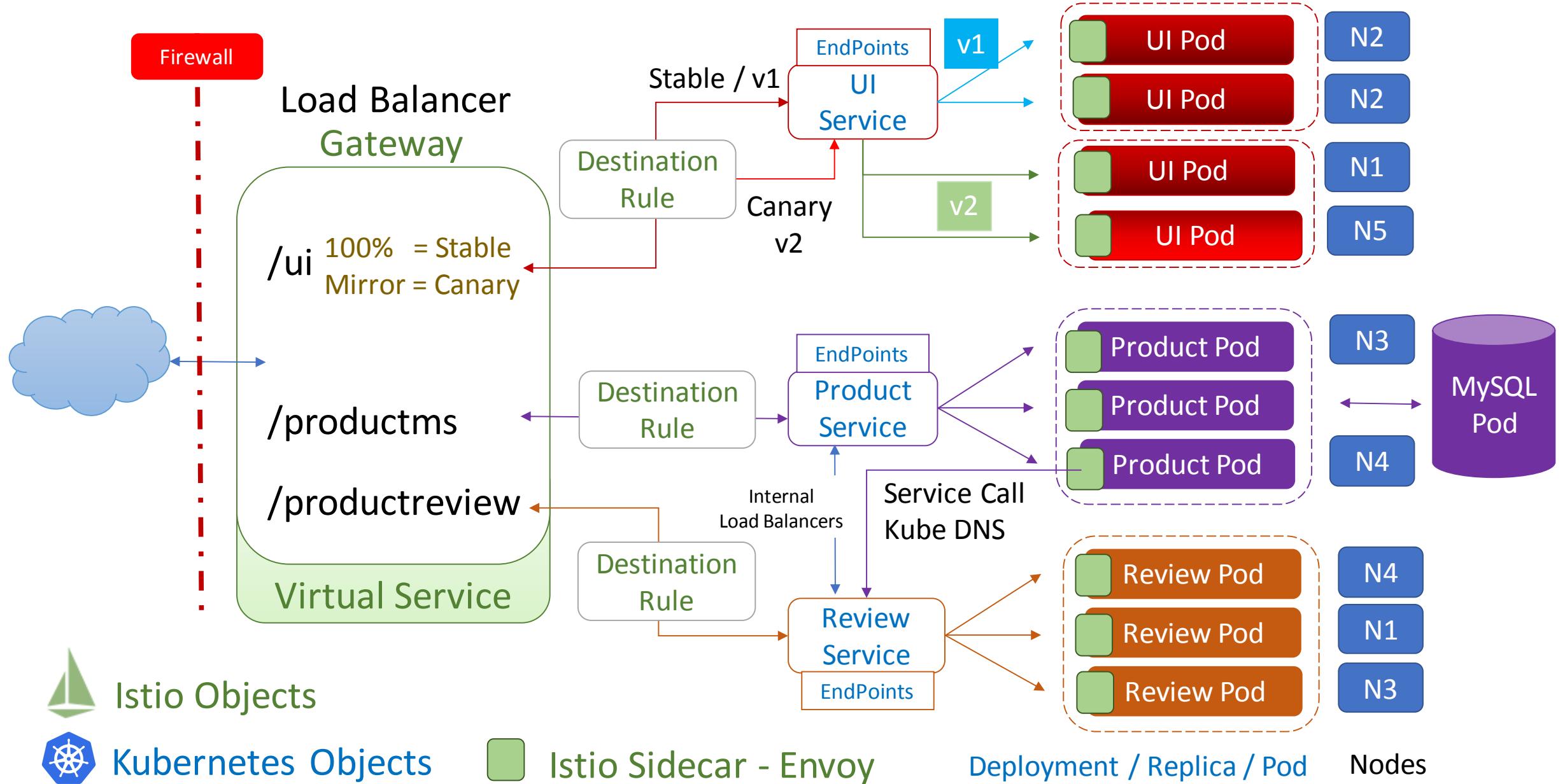
P M C



Shopping Portal Mirror Data

Istio Control Plane

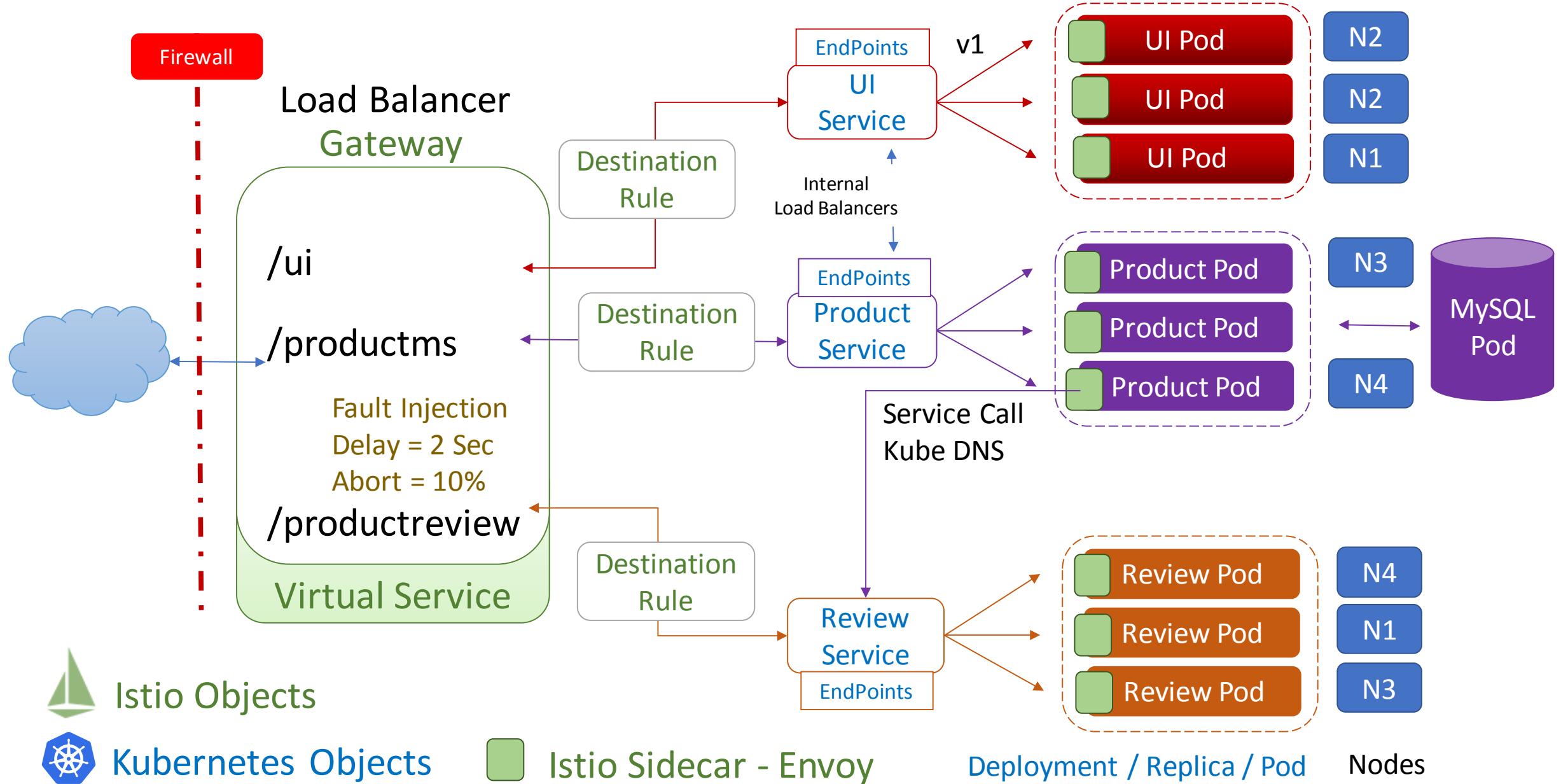
P M C



Shopping Portal Fault Injection

Istio Control Plane

P M C



Istio Security

Network Security

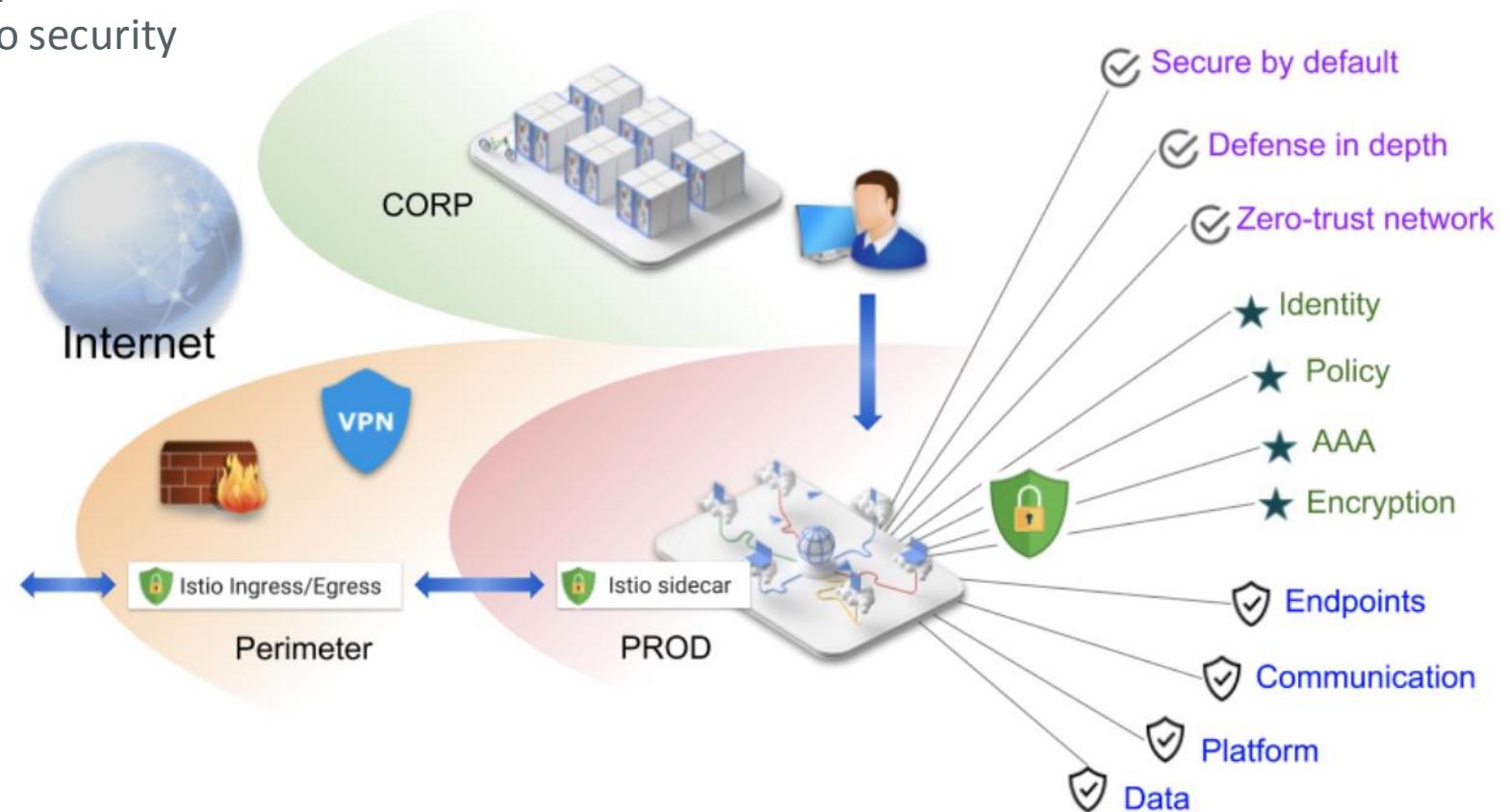
RBAC



Istio Security

It provide **strong identity, powerful policy, transparent TLS encryption, and authentication, authorization and audit (AAA) tools** to protect your services and data. The goals of Istio security are

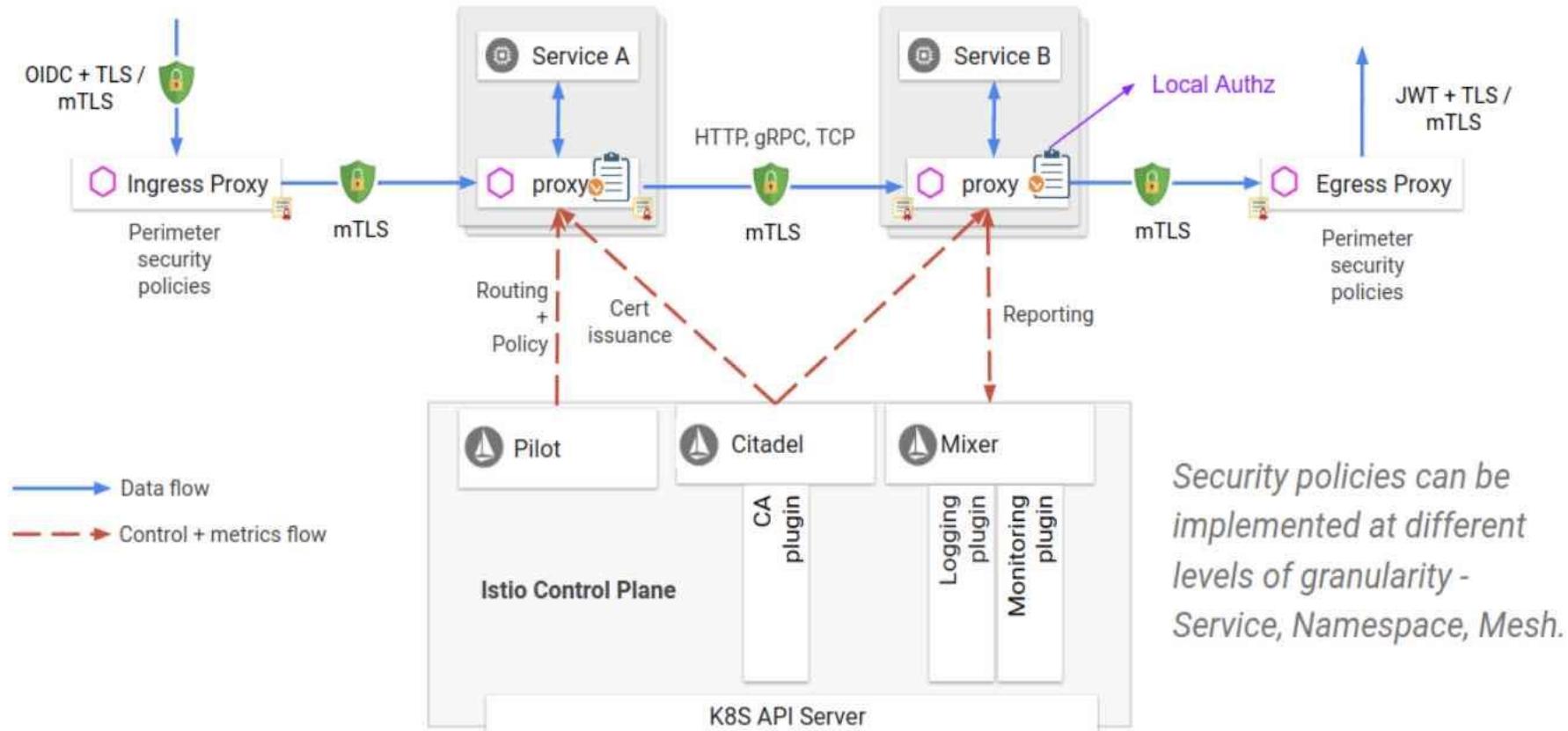
- **Security by default:** no changes needed for application code and infrastructure
- **Defense in depth:** integrate with existing security systems to provide multiple layers of defense
- **Zero-trust network:** build security solutions on untrusted networks





Istio Security Architecture

- **Citadel** for key and certificate management
- **Sidecar and perimeter proxies** to implement secure communication between clients and servers
- **Pilot** to distribute authentication policies and secure naming information to the proxies
- **Mixer** to manage authorization and auditing





Istio Mesh Policy / Policy

- **Mesh-wide policy:** A policy defined in the mesh-scope storage with no target selector section. There can be at most **one** mesh-wide policy **in the mesh**.
- **Namespace-wide policy:** A policy defined in the namespace-scope storage with name default and no target selector section. There can be at most **one** namespace-wide policy **per namespace**.
- **Service-specific policy:** a policy defined in the namespace-scope storage, with non-empty target selector section. A namespace can have **zero, one, or many** service-specific policies

```

    shopping-tls-enable.yml • mtls-ns-wide.yml ×
1 apiVersion: "authentication.istio.io/v1alpha1"
2 kind: "Policy"
3 metadata:
4   name: "default"
5   namespace: "shoppingportal"
6 spec:
7   peers:
8     - mtls: {}

```

shopping-tls-enable.yml • mtls-ns-wide.yml ×

```

1 apiVersion: authentication.istio.io/v1alpha1
2 kind: "MeshPolicy"
3 metadata:
4   name: "default"
5 spec:
6   peers:
7     - mtls: {}
8
9
10
11
12
13
14
15
16
17
18
19
20

```

```

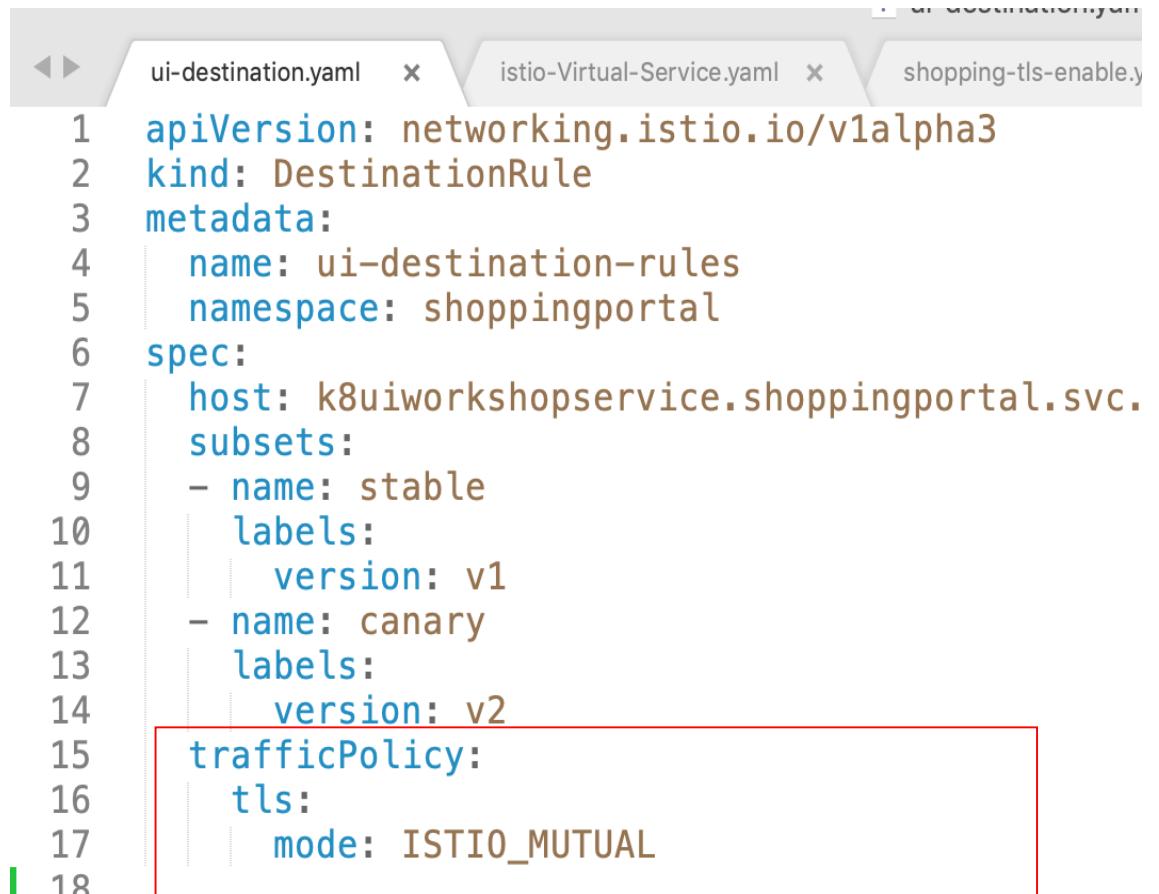
9 apiVersion: networking.istio.io/v1alpha3
10 kind: DestinationRule
11 metadata:
12   name: "enable-mtls"
13   namespace: "default" # even though we specify a namespace,
14   # this rule applies to all namespaces
15 spec:
16   host: "*.local"
17   trafficPolicy:
18     tls:
19       mode: ISTIO_MUTUAL
20

```

To enforce **uniqueness** for mesh-wide and namespace-wide policies, **Istio accepts only one authentication policy per mesh and one authentication policy per namespace**. Istio also requires mesh-wide and namespace-wide **policies to have the specific name default**.

Istio Destination Rule

Configure Istio services to send mutual TLS traffic by setting Destination Rule.



```
ui-destination.yaml  x  istio-Virtual-Service.yaml  x  shopping-tls-enable.yaml
1 apiVersion: networking.istio.io/v1alpha3
2 kind: DestinationRule
3 metadata:
4   name: ui-destination-rules
5   namespace: shoppingportal
6 spec:
7   host: k8uiworkshopservice.shoppingportal.svc.
8   subsets:
9     - name: stable
10       labels:
11         version: v1
12     - name: canary
13       labels:
14         version: v2
15       trafficPolicy:
16         tls:
17           mode: ISTIO_MUTUAL
```

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



Istio RBAC

Enable / Disable
RBAC for specific
namespace(s) or
all.

The image shows a code editor interface with two tabs: "sa-define.yml" and "rbac-config-ON.yaml". The "rbac-config-ON.yaml" tab is active, displaying the following YAML configuration:

```
1 apiVersion: "rbac.istio.io/v1alpha1"
2 kind: ClusterRbacConfig
3 metadata:
4   name: default
5 spec:
6   mode: 'ON_WITH_INCLUSION'
7   inclusion:
8     namespaces: ["shoppingportal"]
```



Service Account

```
sa-define.yml x
1 apiVersion: v1
2 kind: ServiceAccount
3 metadata:
4   name: productms-sa
5   namespace: shoppingportal
6 ---
7 apiVersion: v1
8 kind: ServiceAccount
9 metadata:
10  name: productreviewms-sa
11  namespace: shoppingportal
12 ---
13 apiVersion: v1
14 kind: ServiceAccount
15 metadata:
16   name: ui-sa
17   namespace: shoppingportal
18
```

```
sa-define.yml x rbac-config-ON.yaml x productreviewms-servicerole.yml x
1 apiVersion: "rbac.istio.io/v1alpha1"
2 kind: ServiceRole
3 metadata:
4   name: productreviewms-reviews-viewer
5   namespace: shoppingportal
6 spec:
7   rules:
8     - services: ["productreviewservice.shoppingportal.svc.cluster.local"]
9       methods: ["GET", "POST"]
10  ---
11 apiVersion: "rbac.istio.io/v1alpha1"
12 kind: ServiceRoleBinding
13 metadata:
14   name: bind-productreviewms-reviews
15   namespace: shoppingportal
16 spec:
17   subjects:
18     - user: "cluster.local/ns/shoppingportal/sa/productms-sa"
19   roleRef:
20     kind: ServiceRole
21     name: "productreviewms-reviews-viewer"
```



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 vpca-7532a92g --internet-gateway- id igw-b837249v1
```

Create Route table for the VPC

```
$ aws ec2 create-route-table --vpc-id vpca-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
```



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

Memory limit =
Memory Request

CPU Limit =
CPU Request

QoS:
Burstable

!= Guaranteed
and
Has either
Memory OR
CPU Request

QoS:
Best Effort

No
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.

```
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://kubernetes.io/docs/concepts/policy/resource-quotas/>

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
1 apiVersion: v1
2 kind: LimitRange
3 metadata:
4   name: sigma-lr
5 spec:
6   limits:
7     - default:
8       memory: 256Mi
9         cpu: 600m
10      defaultRequest:
11        memory: 128Mi
12        cpu: 100m
13      max:
14        cpu: 1000m
15        memory: 200Mib
16      min:
17        cpu: 10m
18        memory: 10Mib
19      type: Container
```

Kubernetes Pod Environment Variables

```

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: dapi-envars-fieldref
5   namespace: sigma
6 spec:
7   containers:
8     - name: test-container
9       image: k8s.gcr.io/busybox
10      command: [ "sh", "-c"]
11      args:
12        - while true; do
13          echo -en '\n';
14          printenv MY_NODE_NAME MY_POD_NAME MY_POD_NAMESPACE;
15          printenv MY_POD_IP MY_POD_SERVICE_ACCOUNT;
16          sleep 10;
17        done;
18      env:
19        - name: MY_NODE_NAME
20          valueFrom:
21            fieldRef:
22              fieldPath: spec.nodeName
23        - name: MY_POD_NAME
24          valueFrom:
25            fieldRef:
26              fieldPath: metadata.name
27        - name: MY_POD_NAMESPACE
28          valueFrom:
29            fieldRef:
30              fieldPath: metadata.namespace
31        - name: MY_POD_IP
32          valueFrom:
33            fieldRef:
34              fieldPath: status.podIP
35        - name: MY_POD_SERVICE_ACCOUNT
36          valueFrom:
37            fieldRef:
38              fieldPath: spec.serviceAccountName
39
40 restartPolicy: Never

```

```

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: dapi-envars-resourcefieldref
5   namespace: sigma
6 spec:
7   containers:
8     - name: test-container
9       image: k8s.gcr.io/busybox:1.24
10      command: [ "sh", "-c"]
11      args:
12        - while true; do
13          echo -en '\n';
14          printenv MY_CPU_REQUEST MY_CPU_LIMIT;
15          printenv MY_MEM_REQUEST MY_MEM_LIMIT;
16          sleep 10;
17        done;
18      resources:
19        requests:
20          memory: "32Mi"
21          cpu: "125m"
22        limits:
23          memory: "64Mi"
24          cpu: "250m"
25
26      env:
27        - name: MY_CPU_REQUEST
28          valueFrom:
29            resourceFieldRef:
30              containerName: test-container
31              resource: requests.cpu
32        - name: MY_CPU_LIMIT
33          valueFrom:
34            resourceFieldRef:
35              containerName: test-container
36              resource: limits.cpu
37        - name: MY_MEM_REQUEST
38          valueFrom:
39            resourceFieldRef:
40              containerName: test-container
41              resource: requests.memory
42        - name: MY_MEM_LIMIT
43          valueFrom:
44            resourceFieldRef:
45              containerName: test-container
46              resource: limits.memory
47
48 restartPolicy: Never

```

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 x
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: productreviewmssecret
5   namespace: shoppingportal
6   type: Opaque
7 data:
8   secretkey: cm9vdA==
9
```

```
productreview-deployment.yaml x
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: productreview-deploy
5   namespace: shoppingportal
6 spec:
7   replicas: 1
8   selector:
9     matchLabels:
10    name: productreview
11    zone: prod
12   minReadySeconds: 10
13   strategy:
14     type: RollingUpdate
15     rollingUpdate:
16       maxUnavailable: 1
17       maxSurge: 1
18   template:
19     metadata:
20       labels:
21         name: productreview
22         version: "1.0.0"
23         release: stable
24         tier: fe
25         zone: prod
26         managed-by: m2
27 spec:
28   containers:
29     - name: productreview-ctr
30       image: metamagicglobal/productreviewms
31       imagePullPolicy: Always
32       ports:
33         - containerPort: 8082
34     env:
35       - name: PRODUCTREVIEW_VERSION
36         valueFrom:
37           secretKeyRef:
38             name: productreviewmssecret
39             key: secretkey
40
```

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



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 x
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: special-config
5   namespace: sigma
6 data:
7   SPECIAL_LEVEL: very
8   SPECIAL_TYPE: charm
```

```
app1-configmap-2.yaml x
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: pod-configmap
5   namespace: sigma
6 spec:
7   containers:
8     - name: test-container
9       image: k8s.gcr.io/busybox
10      command: [ "/bin/sh", "-c",
11                  "echo ${SPECIAL_LEVEL_KEY} ${SPECIAL_TYPE_KEY}" ]
12 env:
13   - name: SPECIAL_LEVEL_KEY
14     valueFrom:
15       configMapKeyRef:
16         name: special-config
17         key: SPECIAL_LEVEL
18   - name: SPECIAL_TYPE_KEY
19     valueFrom:
20       configMapKeyRef:
21         name: special-config
22         key: SPECIAL_TYPE
23 restartPolicy: Never
```

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



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 ×  
1 apiVersion: settings.k8s.io/v1alpha1  
2 kind: PodPreset  
3 metadata:  
4   name: allow-database  
5   namespace: sigma  
6 spec:  
7   selector:  
8     matchLabels:  
9       role: backend  
10 env:  
11   - name: DB_PORT  
12     value: "6379"
```

```
app1-pod-presets.yaml ×  
1 apiVersion: v1  
2 kind: Pod  
3 metadata:  
4   name: customersvc  
5   namespace: sigma  
6 labels:  
7   app: customersvc  
8   role: backend  
9 spec:  
10  containers:  
11    - name: customersvc-ctr  
12      image: nginx  
13      ports:  
14        - containerPort: 8080
```



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.

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.

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: $400 > \text{CODE} \geq 200$.



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
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: productreview-deploy
5   namespace: shoppingportal
6 spec:
7   replicas: 1
8   selector:
9     matchLabels:
10    name: productreview
11    zone: prod
12   minReadySeconds: 10
13   strategy:
14     type: RollingUpdate
15     rollingUpdate:
16       maxUnavailable: 1
17       maxSurge: 1
18   template:
19     metadata:
20       labels:
21         name: productreview
22         version: "1.0.0"
23         release: stable
24         tier: fe
25         zone: prod
26         managed-by: m2
27 spec:
28   containers:
29     - name: productreview-ctr
30       image: metamagicglobal/productreviewms
31       imagePullPolicy: Always
32       ports:
33         - containerPort: 8082
34       livenessProbe:
35         httpGet:
36           path: /productreviewms/check/live
37           port: 8082
38         initialDelaySeconds: 15
39         periodSeconds: 15
```



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
1 apiVersion: policy/v1beta1
2 kind: PodDisruptionBudget
3 metadata:
4   name: app1pod-pdb
5 spec:
6   minAvailable: 2
7   selector:
8     matchLabels:
9       desire3d.io/name: app1pod
10      desire3d.io/zone: prod
11
```

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



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 use label selectors to make the selection.
- Assign the label to Node
- Assign Node Selector to a Pod

```
app1-pod-affinity1.yaml ×
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx
5   namespace: sigma
6   labels:
7     env: test
8 spec:
9   containers:
10    - name: nginx
11      image: nginx
12      imagePullPolicy: IfNotPresent
13      nodeSelector:
14        disktype: ssd
15
```

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

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



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/>

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.

Persistent Volume

Storage Class

Persistent Volume Claim

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/*)



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



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 •
1 kind: StorageClass
2 apiVersion: storage.k8s.io/v1
3 metadata:
4   name: omega-sc-local
5 provisioner: k8s.io/minikube-hostpath
6 volumeBindingMode: Immediate
7
```

1 v-local-volume.yaml x
1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4 name: omega-volume-local
5 labels:
6 name: storage-local
7 spec:
8 storageClassName: omega-sc-local
9 capacity:
10 storage: 10Gi
11 volumeMode: Filesystem
12 accessModes:
13 - ReadWriteMany
14 persistentVolumeReclaimPolicy: Retain
15 hostPath:
16 path: "/Users/arafkarsh/data"
17

Change the above path in your system



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 x
1 kind: PersistentVolumeClaim
2 apiVersion: v1
3 metadata:
4   name: omega-pvc-local
5   namespace: omega-local
6 spec:
7   storageClassName: omega-sc-local
8   accessModes:
9     - ReadWriteMany
10  resources:
11    requests:
12      storage: 2Gi
```

2

```
v-local-dep.yaml x
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: omega-local-deploy
5   namespace: omega-local
6 spec:
7   replicas: 3
8   selector:
9     matchLabels:
10       desire3d.io/name: omega-pod
11   minReadySeconds: 10
12   strategy:
13     type: RollingUpdate
14     rollingUpdate:
15       maxUnavailable: 1
16       maxSurge: 1
17   template:
18     metadata:
19       labels:
20         desire3d.io/name: omega-pod
21         desire3d.io/version: "1.0.0"
22         desire3d.io/release: stable
23         desire3d.io/tier: fe
24         desire3d.io/zone: prod
25         desire3d.io/managed-by: m2
26   spec:
27     volumes:
28       - name: omega-volume-local
29         persistentVolumeClaim:
30           # Local Storage Claim
31           claimName: omega-pvc-local
32     containers:
33       - name: omega-ctr
34         image: metamagic/oshell
35         env:
36           - name: POD_IP
37             valueFrom:
38               fieldRef:
39                 fieldPath: status.podIP
40             volumeMounts:
41               # Mount omega-volume-local from
42               # persistentVolumeClaim
43               - mountPath: "/home/data"
44                 name: omega-volume-local
```

3



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](#)

```
pvc — Shell — bash — bash — 187x23
~/k8s/sigma/omega/pvc — Shell — bash      ~/k8s/sigma/omega/pvc — Shell — bash      ~/k8s/workshop — Shell — bash      ~/data — Shell — bash
[1] 1
ODC-MacBook-Pro-2017:pvc arafkarsh$ kubectl get sc,pv,pvc,pods -n omega-local
NAME                                     PROVISIONER          AGE
storageclass.storage.k8s.io/omega-sc-local   k8s.io/minikube-hostpath  4m
storageclass.storage.k8s.io/standard (default)  k8s.io/minikube-hostpath  5d
[2] 2
[3] 3
NAME                                     CAPACITY    ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
persistentvolume/omega-volume-local        10Gi       RWX           Retain        Bound     omega-local/omega-pvc-local
persistentvolume/pvc-7eba2ae4-d191-11e8-8f90-08002762493a  4Gi        RWX           Delete        Bound     omega-local/omega-pvc-local-dynamic
[4] 4
NAME                                     STATUS      VOLUME
persistentvolumeclaim/omega-pvc-local      Bound      omega-volume-local
persistentvolumeclaim/omega-pvc-local-dynamic  Bound      pvc-7eba2ae4-d191-11e8-8f90-08002762493a
[5] 5
NAME                                     READY      STATUS    RESTARTS  AGE
pod/omega-local-deploy-578fcf8c97-dlxvt  1/1      Running   0         4m
pod/omega-local-deploy-578fcf8c97-dwj9r   1/1      Running   0         4m
pod/omega-local-deploy-578fcf8c97-x2qxw   1/1      Running   0         4m
pod/omega-local-deploy-dynamic-74d8b769c-fpxls 1/1      Running   0         4m
pod/omega-local-deploy-dynamic-74d8b769c-kjcdf 1/1      Running   0         4m
pod/omega-local-deploy-dynamic-74d8b769c-qctxj 1/1      Running   0         4m
[6] 6
ODC-MacBook-Pro-2017:pvc arafkarsh$ [7] 7
```



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 x
1 apiVersion: storage.k8s.io/v1
2 kind: StorageClass
3 metadata:
4   name: omega-sc-aws-slow
5 provisioner: kubernetes.io/aws-ebs
6 parameters:
7   type: io1
8   zone: us-east-1b
9   iopsPerGB: "10"
10
```

```
v-aws-volume.yaml x
1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   name: omega-volume-aws
5   labels:
6     name: storage-aws
7 spec:
8   storageClassName: omega-sc-aws-slow
9   capacity:
10    storage: 30Gi
11   accessModes:
12     - ReadWriteMany
13   awsElasticBlockStore:
14     # Volume ID is auto generated by AWS
15     # $ aws ec2 create-volume --size 100
16     # Returns the volume ID
17     # with volume size of 100 Gi
18     volumeID: vol-06e25d93b3b4e3302
19   fsType: ext4
```

\$ aws ec2 create-volume --size 100

Volume ID is auto generated



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
1 kind: PersistentVolumeClaim
2 apiVersion: v1
3 metadata:
4   name: omega-pvc-aws
5   namespace: omega-aws
6 spec:
7   storageClassName: omega-sc-aws-slow
8   accessModes:
9     - ReadWriteMany
10  resources:
11    requests:
12      storage: 2Gi
```

2

```
v-omega-dep.yaml
1 apiVersion: apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: omega-aws-deploy
5   namespace: omega-aws
6 spec:
7   replicas: 3
8   selector:
9     matchLabels:
10    desire3d.io/name: omega-pod-aws
11   minReadySeconds: 10
12   strategy:
13     type: RollingUpdate
14     rollingUpdate:
15       maxUnavailable: 1
16       maxSurge: 1
17   template:
18     metadata:
19       labels:
20         desire3d.io/name: omega-pod-aws
21         desire3d.io/version: "1.0.0"
22         desire3d.io/release: stable
23         desire3d.io/tier: fe
24         desire3d.io/zone: prod
25         desire3d.io/managed-by: m2
26   spec:
27     volumes:
28       - name: omega-volume-aws
29         persistentVolumeClaim:
30           # Local Storage Claim
31           claimName: omega-pvc-aws
32     containers:
33       - name: omega-ctr
34         image: metamagic/oshell
35         env:
36           - name: POD_IP
37             valueFrom:
38               fieldRef:
39                 fieldPath: status.podIP
40         volumeMounts:
41           # Mount omega-fs from persistentVolumeClaim
42             - mountPath: "/home/data"
43               name: omega-volume-aws
```



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.



```
1 apiVersion: batch/v1
2 kind: Job
3 metadata:
4   name: pi
5   namespace: sigma
6 spec:
7   template:
8     spec:
9       containers:
10      - name: pi
11        image: perl
12        command: ["perl",
13          "-Mbignum=bpi",
14          "-wle",
15          "print bpi(2000)"]
16        restartPolicy: Never
17        backoffLimit: 4
```

Command is wrapped for display purpose.



Kubernetes Cron Jobs

You can use [CronJobs](#) to run jobs on a time-based schedule. These automated jobs run like [Cron](#) tasks on a Linux or UNIX system.

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period

```
1 apiVersion: batch/v1beta1
2 kind: CronJob
3 metadata:
4   name: hello
5 spec:
6   schedule: "*/1 * * * *"
7   jobTemplate:
8     spec:
9       template:
10      spec:
11        containers:
12        - name: hello
13          image: busybox
14          args:
15            - /bin/sh
16            - -c
17            - date; echo Hello from the Kubernetes cluster
18        restartPolicy: OnFailure
19
```

Command is wrapped for display purpose.



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.

```

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

```

 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)



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



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
```

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
```

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

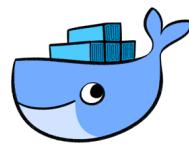
You use field selectors across multiple resource types. This kubectl command selects all `Statefulsets` and `Services` that are not in the default namespace:

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

Best Practices

Docker Best Practices

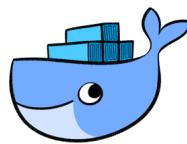
Kubernetes Best Practices



Build Small Container Images



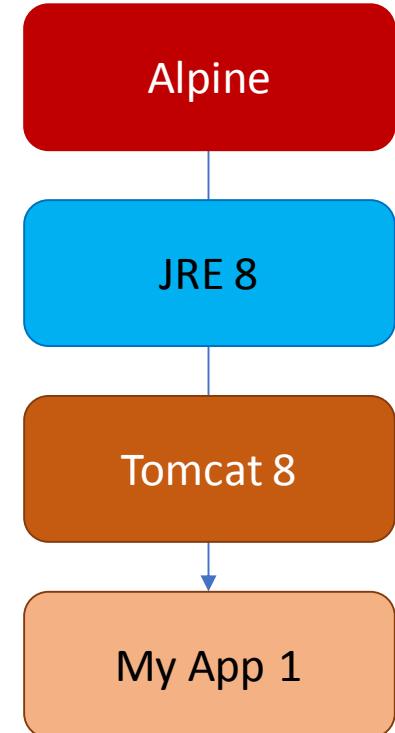
1. Simple Java Web Apps with Ubuntu & Tomcat can have a size of 700 MB
2. Use Alpine Image as your base Linux OS
3. Alpine images are 10x smaller than base Ubuntu images
4. Smaller Image size reduce the Container vulnerabilities.
5. Ensure that only Runtime Environments are there in your container. For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

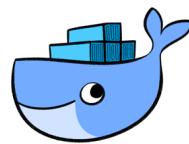


Docker: To Root or Not to Root!



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

1. 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.
2. Never access a Pod directly from another Pod. Always use a Service to access a Pod.
3. User labels to select the pods { app: myapp, tier: frontend, phase: test, deployment: v3 }.
4. Never use :latest tag in the image in the production scenario.

Kubernetes: Namespace



Service-Name.Namespace.svc.cluster.local

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

Kubernetes Cluster

default

Kube system

Kube public

```
$ 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).



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

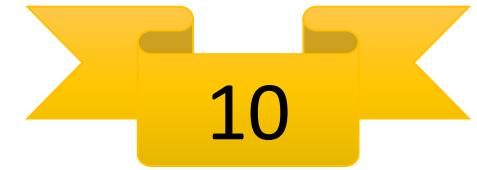
1. For the Best Quality define the requests and limits for your Pods.
2. You can set specific resource requests for a Dev Namespace to ensure that developers don't create pods with a very large resource or a very small resource.
3. Limit Range can be set to ensure that containers were create with too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



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



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



1. Make sure that the Master behind a Load Balancer.
2. Upgrade Master
 1. Scale up the Node with an extra Node
 2. Drain the Node and
 3. Upgrade Node
3. 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](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/>

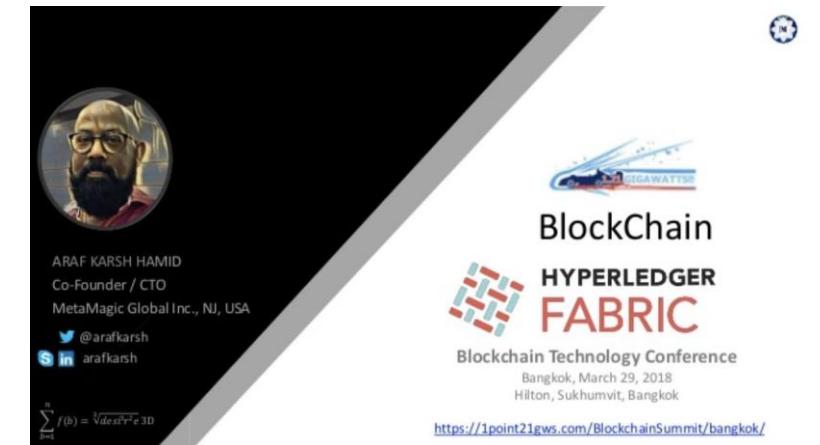
Micro Services Architecture

Part 1 : Infrastructure Comparison &

Design Styles (DDD, Event Sourcing / CQRS, Functional Reactive Programming)

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

A Micro Service will have its own Code Pipeline for build and deployment functionalities and its scope will be defined by the Bounded Context focusing on the Business Capabilities and the interoperability between Micro Services will be achieved using message based communication.



Event Storming & SAGA DESIGN PATTERN

Part 2/4 : Event Sourcing and Distributed Transactions
for Micro Services

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

Part 1 : [Microservices Architecture](#)