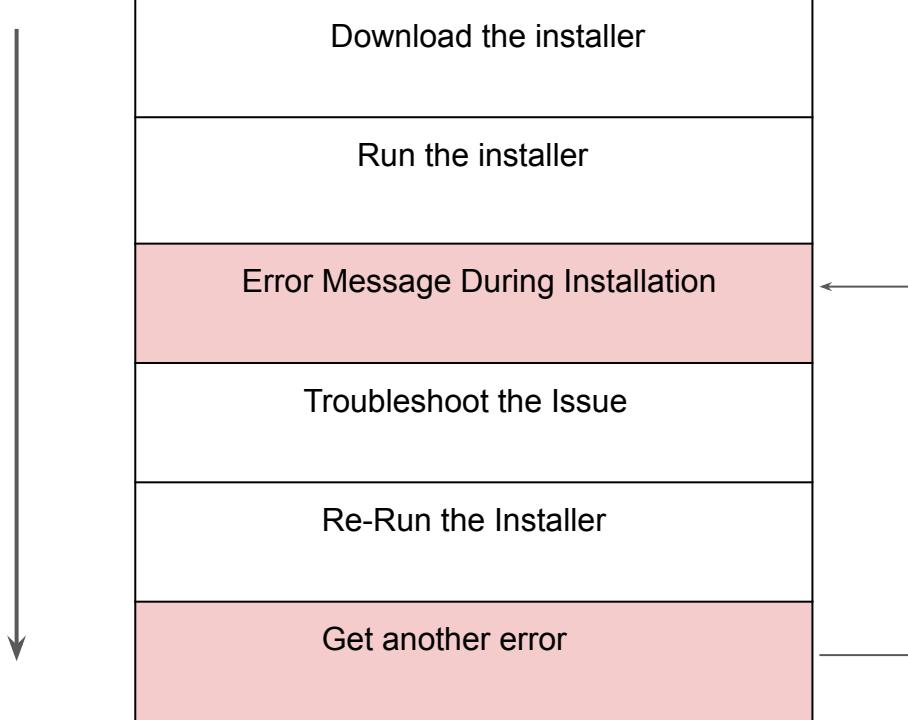

Docker Certified Associate

Instructor: Zeal Vora

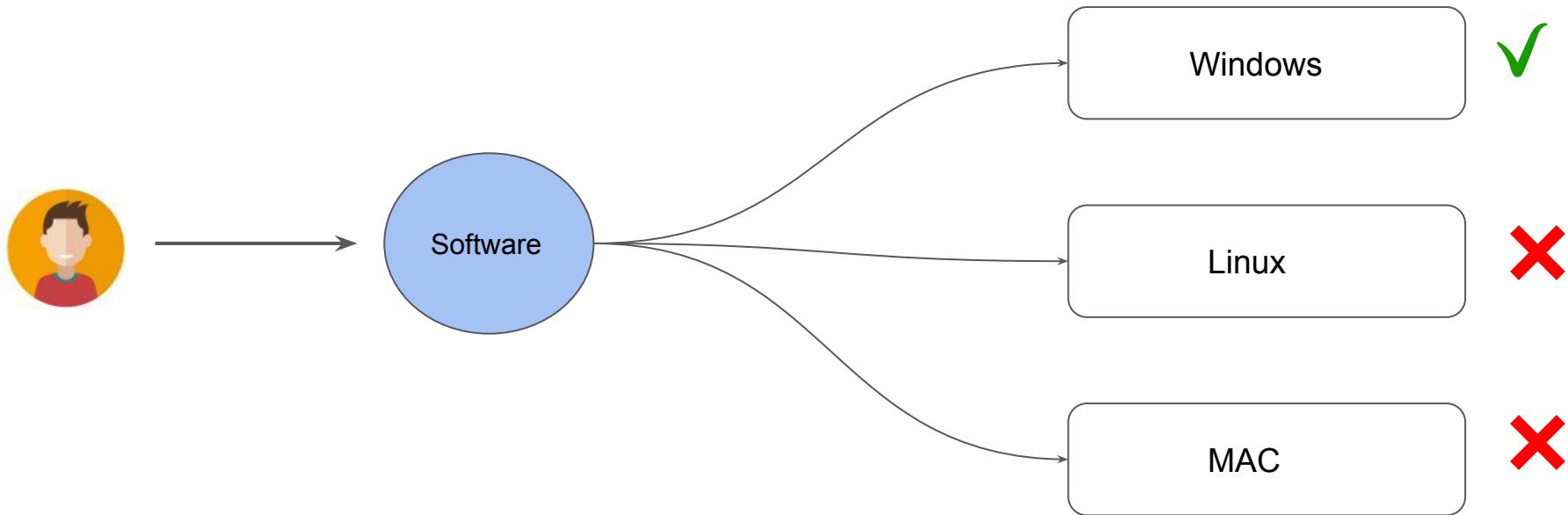
Introduction to Docker

Build once, use anywhere

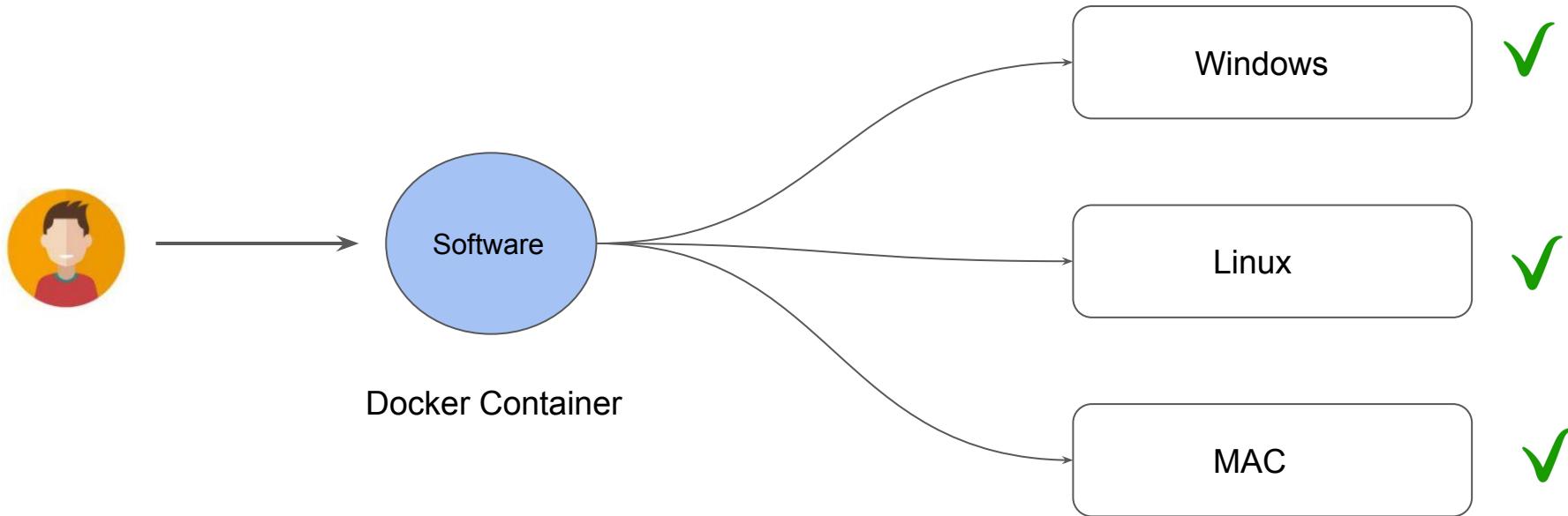
Installation of Software Workflow



What is Docker Trying to Achieve?



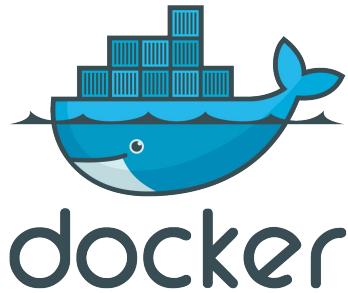
What is Docker Trying to Achieve?



Introduction

Docker is an open platform, once we build a docker container, we can run it anywhere, say it windows, linux, mac whether on laptop, data center or in cloud.

It follows the **build once, run anywhere** approach.

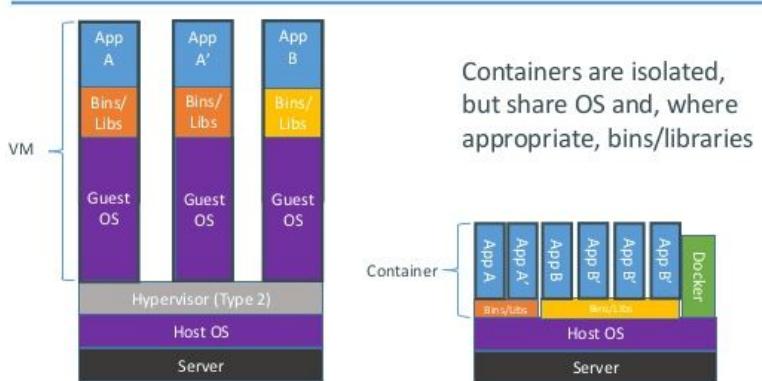


Containers vs Virtual Machines

Virtual Machine contains entire Operating System.

Container uses the resource of the host operating system

Containers vs. VMs

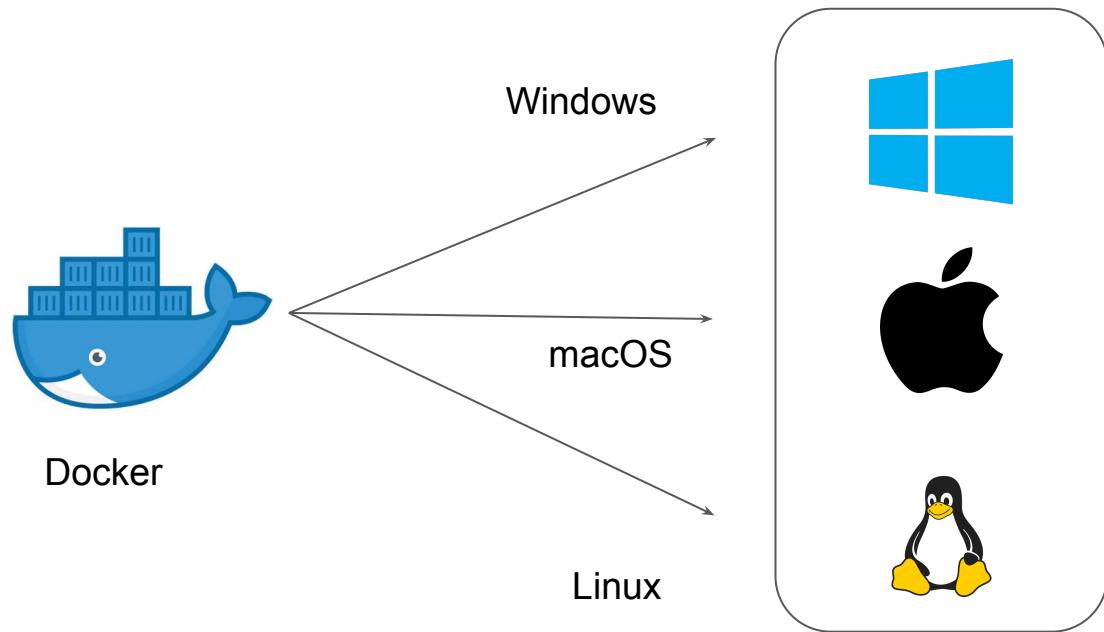


Installation Methods of Docker

Let's Install

Installing Docker

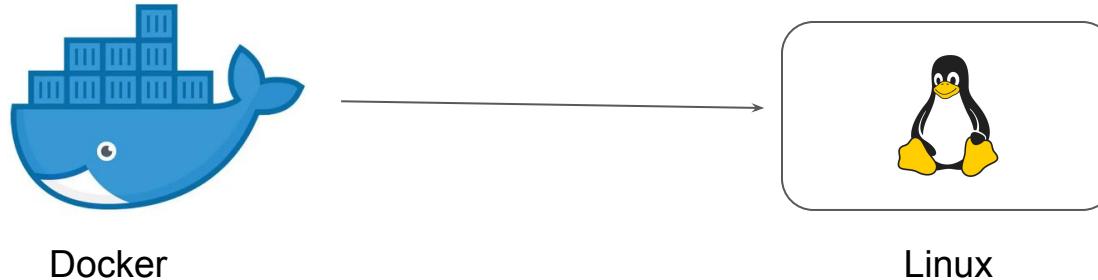
Docker can be installed in wide variety of operating systems.



Preferred Choice for Docker Installation Method

To begin with, you can install Docker Desktop directly within your laptop.

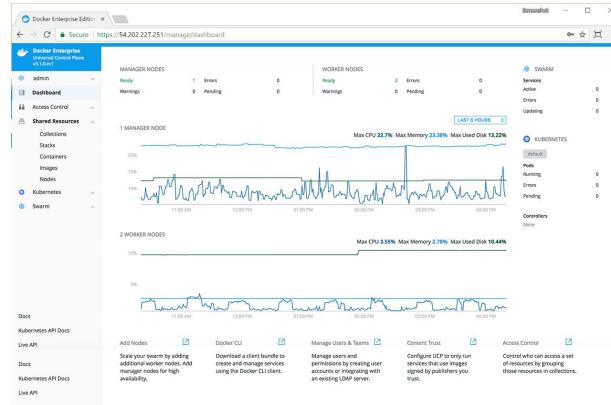
The preferred OS for Docker installation would be Linux.



Why Linux is Preferred Method?

In this course, we will be testing many Docker features and enterprise features.

- Basic Docker Features
- Docker Swarm
- Docker UCP/DTR
- Kubernetes



Revising the Choices

Following diagram illustrates the architecture that we will be following in this course:

Criteria	Choices
Operating System	Ubuntu
Cloud Provider	Digital Ocean

Why Digital Ocean?

1. They provide multiple coupon codes which gives great amount of credits ranging from \$50-100 USD for new users.
2. Provides Managed K8s cluster which will be required in the later section of the course.



Personal Account Security **Referrals**

Give \$100, Get \$25

Everyone you refer gets \$100 in credit over 60 days.
through referrals.

Infrastructure for Docker

Let's Install

Revising the Preferred Choice

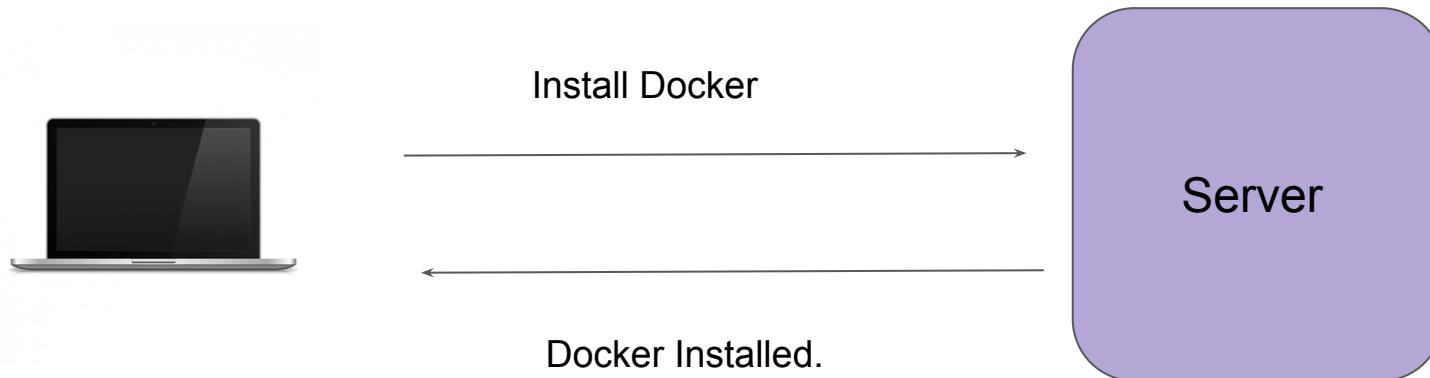
To begin with, you can install Docker Desktop directly within your laptop.

The preferred OS for Docker installation would be Linux.



Creating Infrastructure for Docker

To begin with the Docker installation process, we need one server hosting Ubuntu OS.



Password Based Authentication

There can be multiple methods for authentication against a system.

Password based authentication is the simplest form.

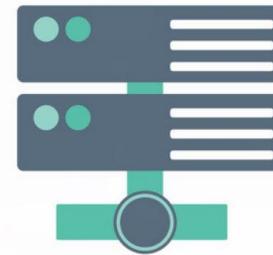


Laptop

My username is admin, I want to login



Hey there, what is your password?



Linux Server

Login with Credentials



Laptop

My username is admin, my password is 12345. I want to login.



Login successful.



Linux Server

Challenges with Password Based Authentication

Password based authentication is generally considered to be less-secure.

Many users write down the passwords in notepad files or as part of sticky notes.

Most users would not create a complex password that is difficult to hack.

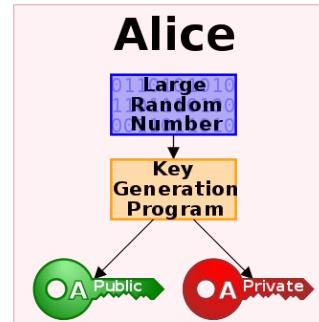


Key Based Authentication

In this type of authentication, there are two special keys that are generated.

One key is called as Public Key and second key is called as Private key.

If public key is stored in server and is used as authentication mechanism, only the corresponding private key can be used to successfully authenticate.



Key Based Authentication



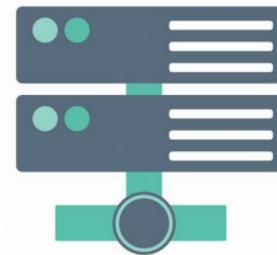
Laptop



My username is admin, I want to login



Hey there, password is not allowed.
You need to authenticate via the key.



Linux Server



Firewall Rules

We do not want the entire internet to connect to our server.

With the help of Firewall, you can restrict the connection to your Docker instance.

Ports	Description
22	Connection to SSH.

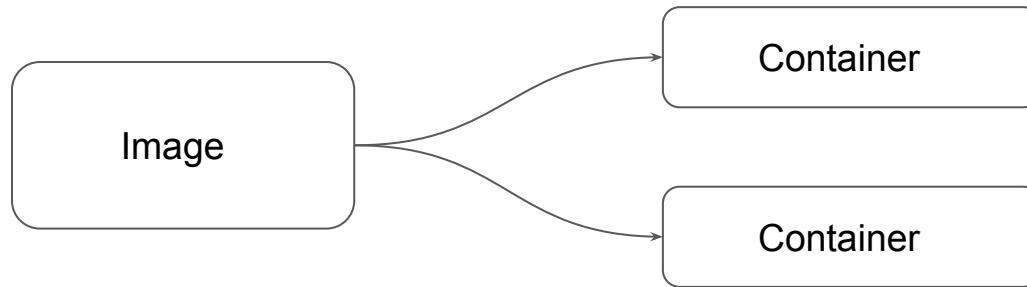
Images and Containers

Build once, use anywhere

Images and Containers

Docker Image is a file which contains all the necessary dependency and configurations which are required to run an application.

Docker Containers is basically a running instance an image.



Container Identification

Build once, use anywhere

Getting Started

When you create a Docker container, it is assigned a universally unique identifier (UUID).

These can help identify the docker container among others.

```
[root@docker-demo ~]# docker run -dt -p 80:80 nginx  
d5187cb1c7f4380b3e37e0c0c811a437d7b8a49d5beb705711a4e54e99d72d77
```

Random Container Names

To help humans, Docker also allow us to supply container names.

By default, if we do not specify the name, docker supplies randomly-generated name from two words, joined by an underscore

[root@docker-demo ~]# docker ps				
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
d5187cb1c7f4	nginx	"nginx -g 'daemon ..."	47 minutes ago	Up 31 minutes
0.0.0.0:80->80/tcp	inspiring_poitras			

Naming Docker Container

By adding `--name=meaningful_name` argument during docker run command, we can specify our own name to the containers.

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          NAMES
PORTS               NAMES
9fba1f62038d        nginx      "nginx -g 'daemon ..."   5 seconds ago   Up 5 seconds   mynginx
```

Port Binding

Build once, use anywhere

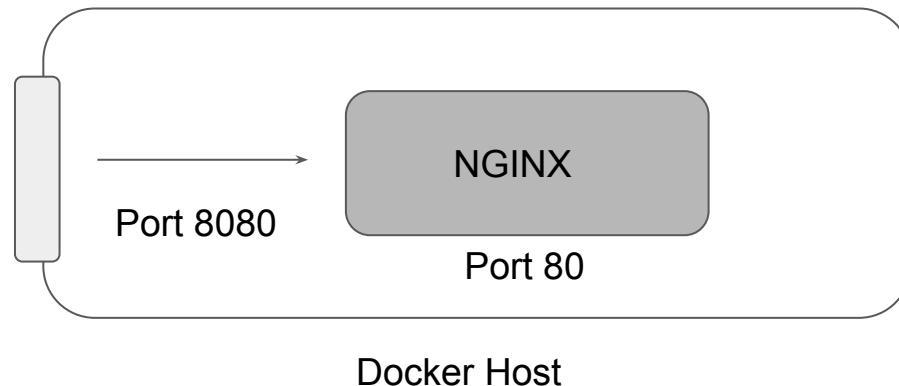
Sample Screenshot

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS
PORTS
NAMES
9fba1f62038d        nginx      "nginx -g 'daemon ..."   5 seconds ago     Up 5 seconds
0.0.0.0:800->80/tcp    mynginx
```

Getting Started

By default Docker containers can make connections to the outside world, but the outside world cannot connect to containers.

If we want to containers to accept incoming connection from the world, you will have to bind it to a host port.



Attach and Detached Mode

Build once, use anywhere

Sample Screenshot

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
PORTS
NAMES
9fba1f62038d        nginx              "nginx -g 'daemon ..."   5 seconds ago    Up 5 seconds
          0.0.0.0:800->80/tcp      mynginx
```

Understanding Detached Mode

When we start a docker container, we need to decide if we want to run in a default foreground mode or the detached mode.

You may want to use this if you want a container to run but do not want to view and follow all its output.

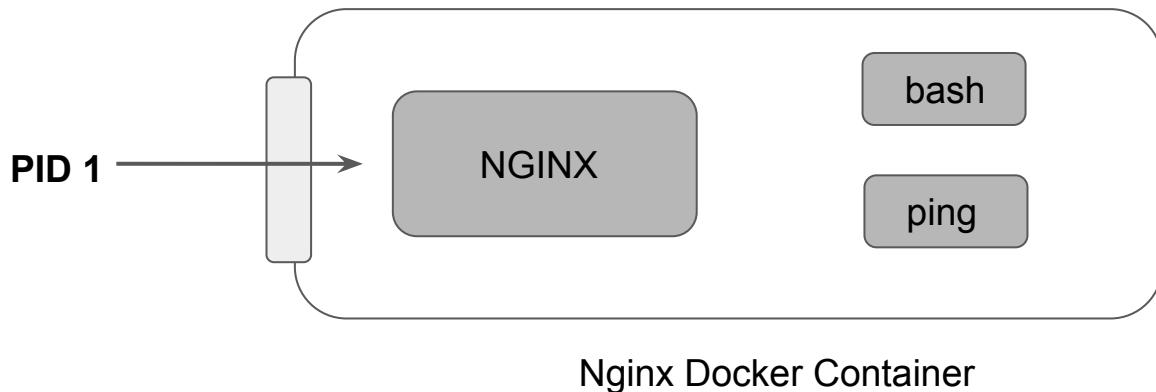
docker container exec

Build once, use anywhere

Overview of docker container exec

The `docker container exec` command runs a new command in a running container.

The command started using `docker exec` only runs while the container's primary process (PID 1) is running, and it is not restarted if the container is restarted.



Importance of IT flag

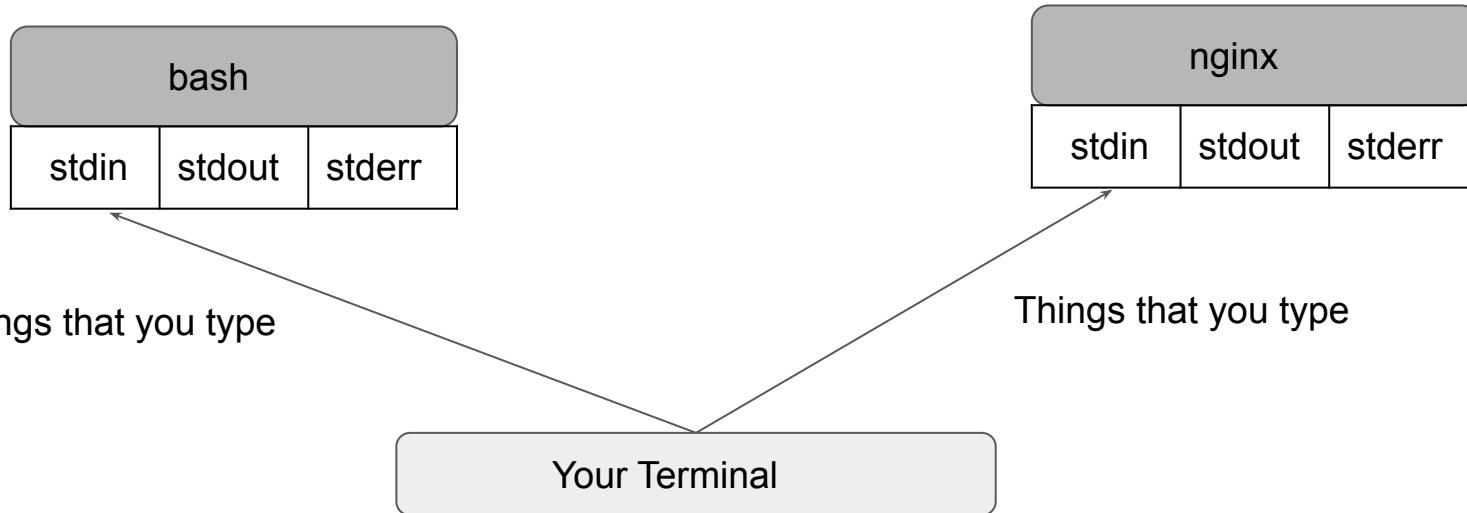
Build once, use anywhere

Sample Screenshot

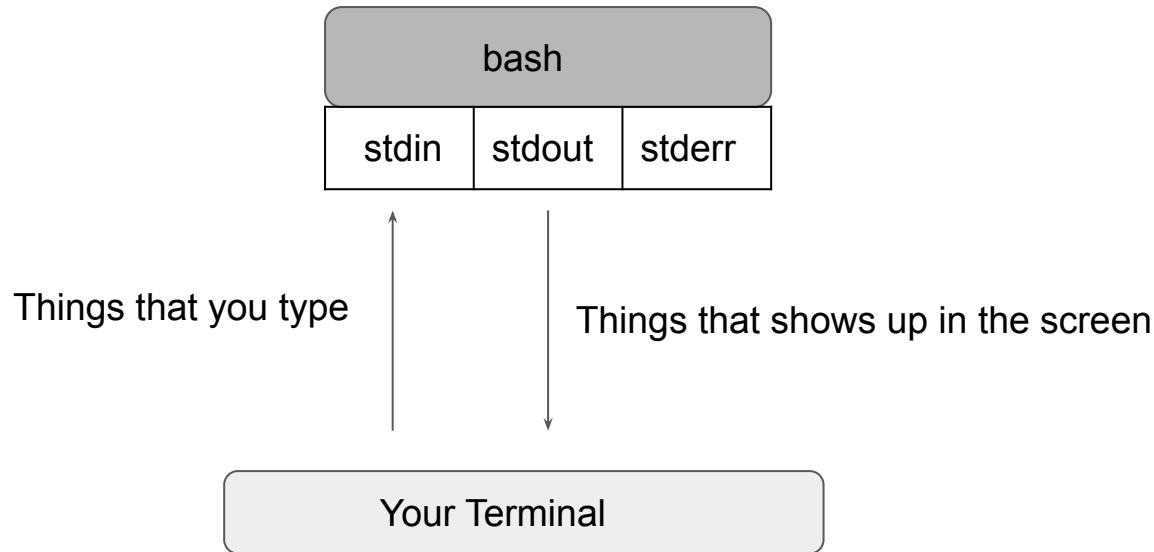
```
[root@docker-demo ~]# docker container exec -it docker-exec bash
root@1cbc45686152:/# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*            LISTEN     1/nginx: master pro
```

Getting Started

Every process that we create in Linux environment, has three open file descriptors; stdin, stdout, stderr.



stdin and stdout



Revising the Topic

Revising the importance of IT flag, remember that:

--interactive flag keeps stdin open even if not attached.

--**tty** flag allocates a pseudo-TTY

Default Container Command

Build once, use anywhere

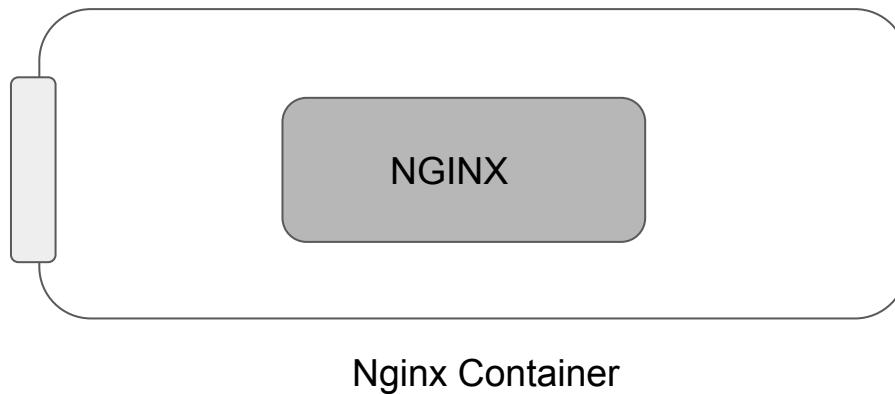
Sample Screenshot

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS
PORTS              NAMES
9fba1f62038d        nginx      "nginx -g 'daemon ..."   5 seconds ago   Up 5 seconds
  0.0.0.0:800->80/tcp    mynginx
```

Getting Started

Whenever we run a container, a default command executes which typically runs as PID 1.

This command can be defined while we are defining the container image.



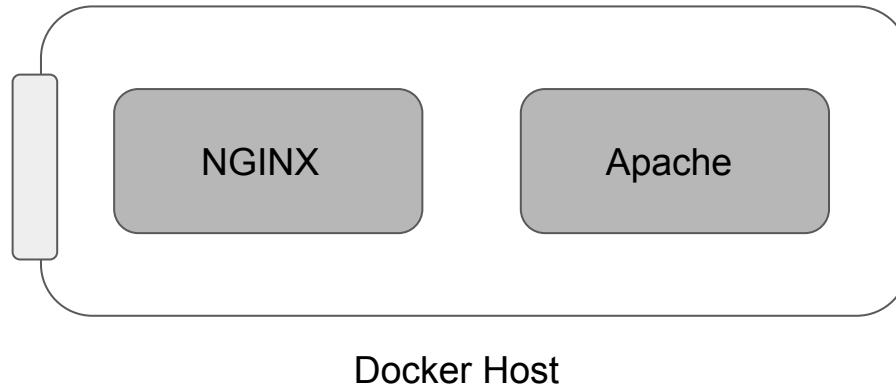
Restart Policies

Automate Starting of Containers

Getting Started

By default, Docker containers will not start when they exit or when docker daemon is restarted.

Docker provides restart policies to control whether your containers start automatically when they exit, or when Docker restarts.



Restart Policy Options

We can specify the restart policy by using the --restart flag with docker run command.

Flag	Description
no	Do not automatically restart the container. (the default)
on-failure	Restart the container if it exits due to an error, which manifests as a non-zero exit code.
unless-stopped	Restart the container unless it is explicitly stopped or Docker itself is stopped or restarted.
always	Always restart the container if it stops.

Working with Docker Images

Build once, use anywhere

Getting Started with Images

Every Docker container is based on an image.

Till now we have been using images which were created by others and available in Docker Hub.

Docker can build images automatically by reading the instructions from a **Dockerfile**

Understanding Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.



Dockerfile

Building Docker Images

Understanding Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.



Format of Dockerfile

The format of Dockerfile is similar to the below syntax:

```
# Comment  
INSTRUCTION arguments
```

A Dockerfile must start with a **FROM`** instruction.

The FROM instruction specifies the Base Image from which you are building.

Dockerfile Commands

The format of Dockerfile is similar to the below syntax:

- FROM
- RUN
- CMD
- LABEL
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- Many More ...

Use-Case - Create Custom Nginx Image

Simple Use-Case

We want to create a custom Nginx Image from which all containers within organization will be launched.

The base container image should have custom index.html file which states:

“Welcome to Base Nginx Container from KPLABS”

..

COPY vs ADD

Build once, use anywhere

Overview of COPY and ADD

COPY and ADD are both Dockerfile instructions that serve similar purposes.

They let you copy files from a specific location into a Docker image.

Difference between COPY and ADD

COPY takes in a **src** and **destination**. It only lets you copy in a local file or directory from your host

ADD lets you do that too, but it also supports 2 other sources.

- First, you can use a URL instead of a local file / directory.
- Secondly, you can extract a tar file from the source directly into the destination.

Use CURL and WGET whenever possible

Using ADD to fetch packages from remote URLs is strongly discouraged; you should use curl or wget instead.

```
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all

RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

EXPOSE

Build once, use anywhere

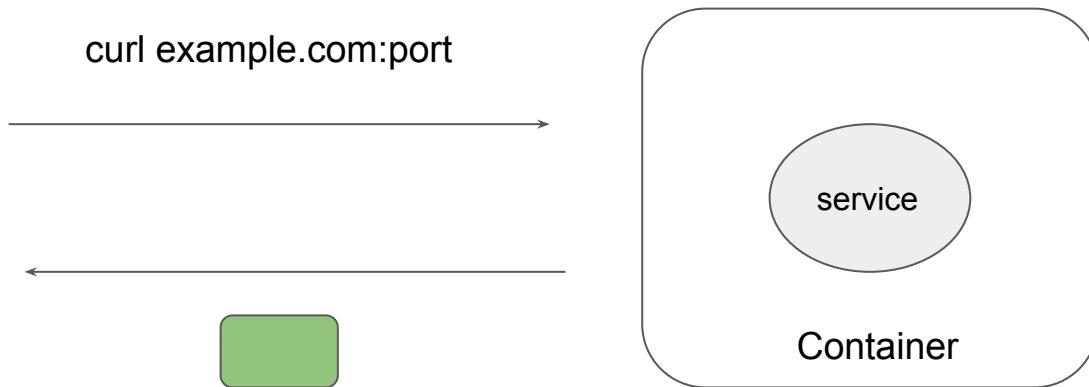
Overview of EXPOSE instruction

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

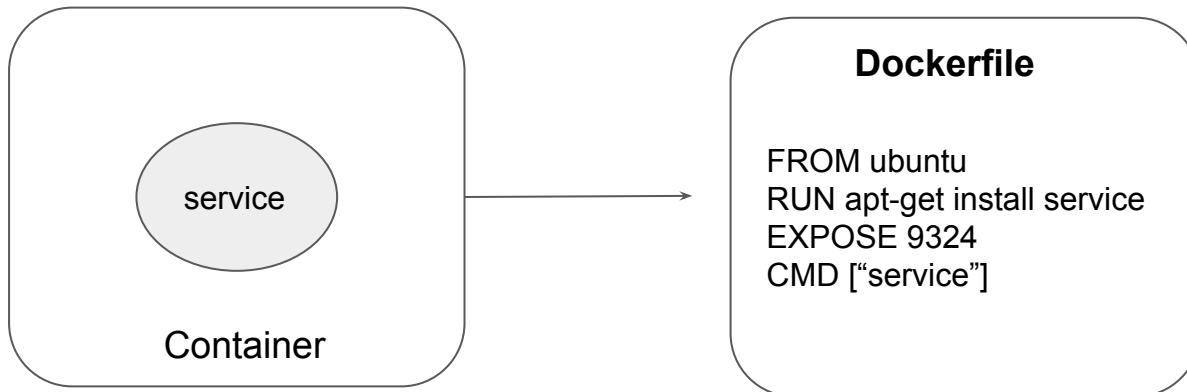
The EXPOSE instruction does not actually publish the port.

It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.

Understanding the Use-Case



Understanding the Use-Case



HEALTHCHECK Options

Docker HealthCheck

Overview of Instruction Options

HEALTHCHECK Instruction is combined with various options to get the desired results.

Options	Default Values
--interval=DURATION	30 seconds
--timeout=DURATION	30 seconds
--start-period=DURATION	0 seconds
--retries=N	3

Understanding the Concept

The health check will first run interval seconds after the container is started, and then again interval seconds after each previous check completes.

So if CMD is curl -I google.com

With default values, the health check will run after every 30 seconds.

Exit Status

The command's exit status indicates the health status of the container. The possible values are:

Possible Values	Description
0: Success	the container is healthy and ready for use
1: Failure	the container is not working correctly
2: Reserved	do not use this exit code

Sample Use-Case

Check every five minutes or so that a web-server is able to serve the site's main page within three seconds:

```
HEALTHCHECK --interval=5m --timeout=3s \
CMD curl -f http://localhost/ || exit 1
```

Overview of CURL

CURL is used with -f option to fail silently.

This is mostly done to better enable scripts etc to better deal with failed attempts.

```
bash-4.2# curl dexter.kplabs.in/test
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.10.1</center>
</body>
</html>
```

Without -f option

```
bash-4.2# curl -f dexter.kplabs.in/test
curl: (22) The requested URL returned error: 404 Not Found
```

With -f option

Dockerfile - ENTRYPOINT

Build once, use anywhere

Overview of ENTRYPOINT

The best use for ENTRYPOINT is to set the image's main command

ENTRYPOINT doesn't allow you to override the command.

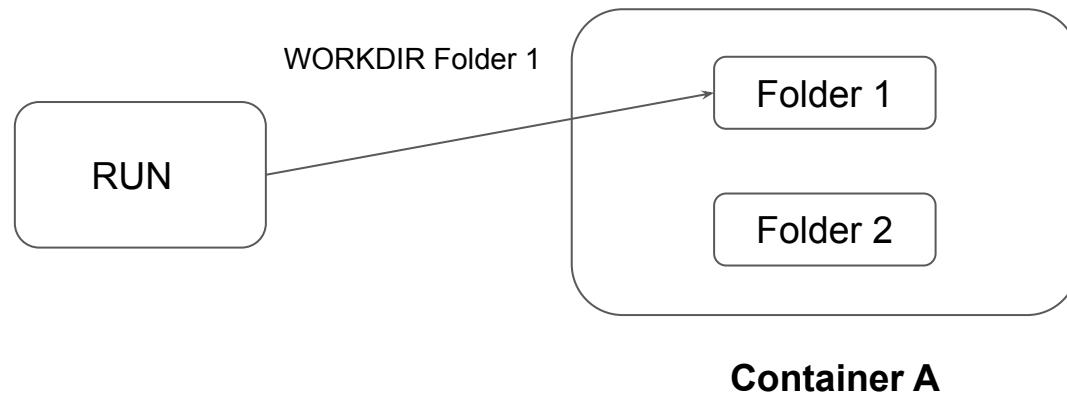
It is important to understand distinction between CMD and ENTRYPOINT.

Dockerfile - WORKDIR

Build once, use anywhere

Overview of the Instruction

The **WORKDIR** instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile



Important Pointer

The WORKDIR instruction can be used multiple times in a Dockerfile

Sample Snippet:

```
WORKDIR /a  
WORKDIR b  
WORKDIR c  
RUN pwd
```

Output = /a/b/c

Dockerfile - ENV

Build once, use anywhere

Overview of the Instruction

The ENV instruction sets the environment variable <key> to the value <value>.

Example Snippet:

```
ENV NGINX 1.2
RUN curl -SL http://example.com/web-\$NGINX.tar.xz
RUN tar -xzvf web-\$NGINX.tar.xz
```

Setting Environment Variables from CLI

You can use the `-e`, `--env`, and `--env-file` flags to set simple environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.

Example Snippet:

```
docker run --env VAR1=value1 --env VAR2=value2 ubuntu env | grep VAR
```

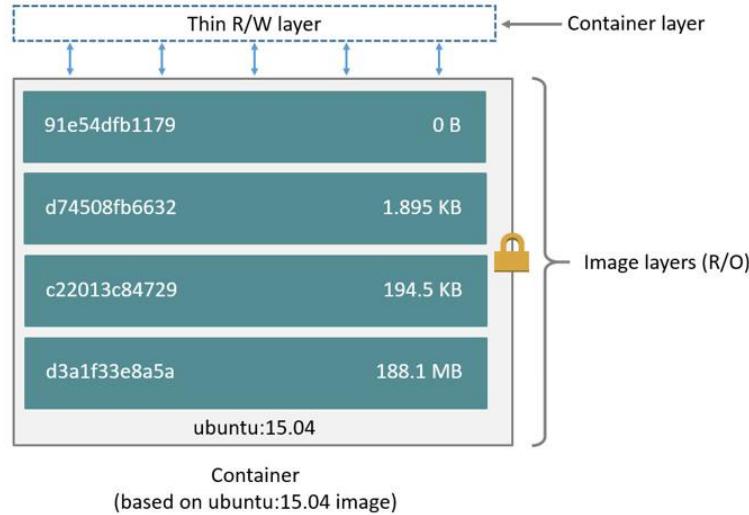
Layers of Docker Image

Building Docker Images

Understanding Layers

- A Docker image is built up from a series of layers.
- Each layer represents an instruction in the image's Dockerfile.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Containers and Layers

The major difference between a container and an image is the top writable layer.

All writes to the container that add new or modify existing data are stored in this writable layer.

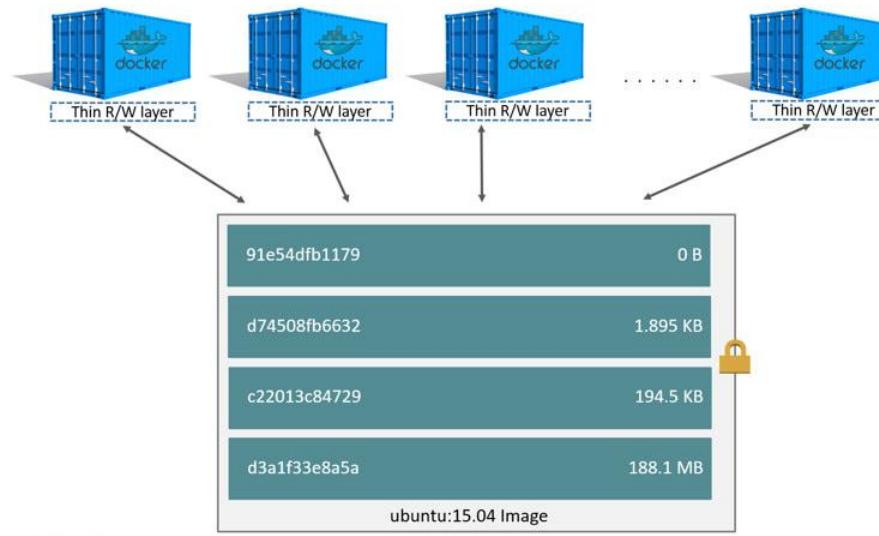
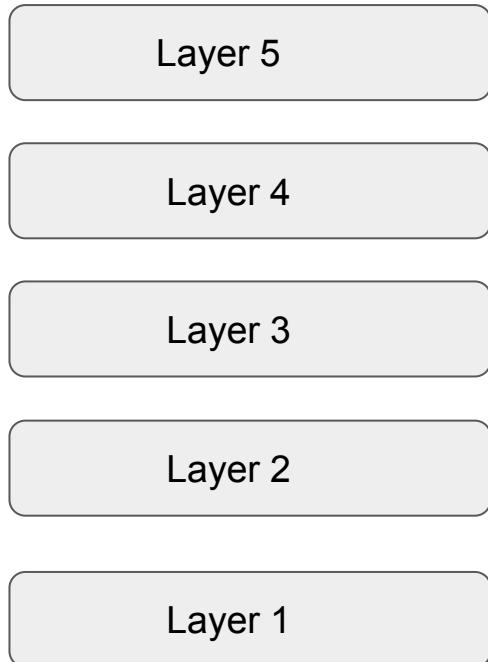


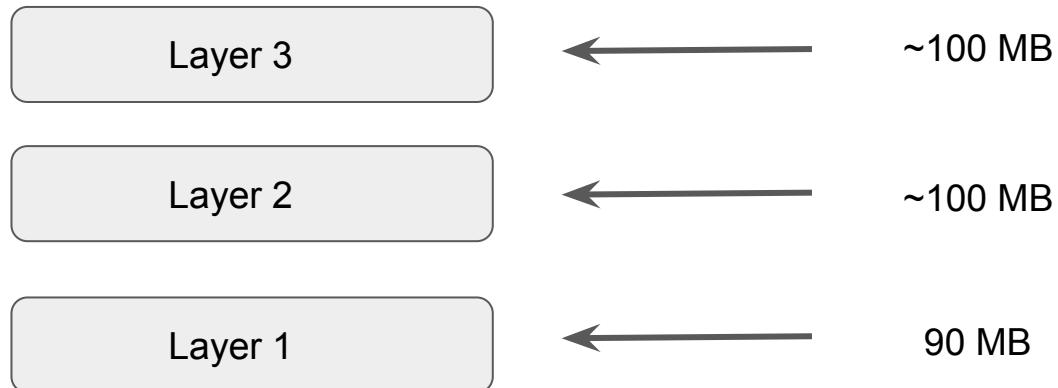
Image and Layer



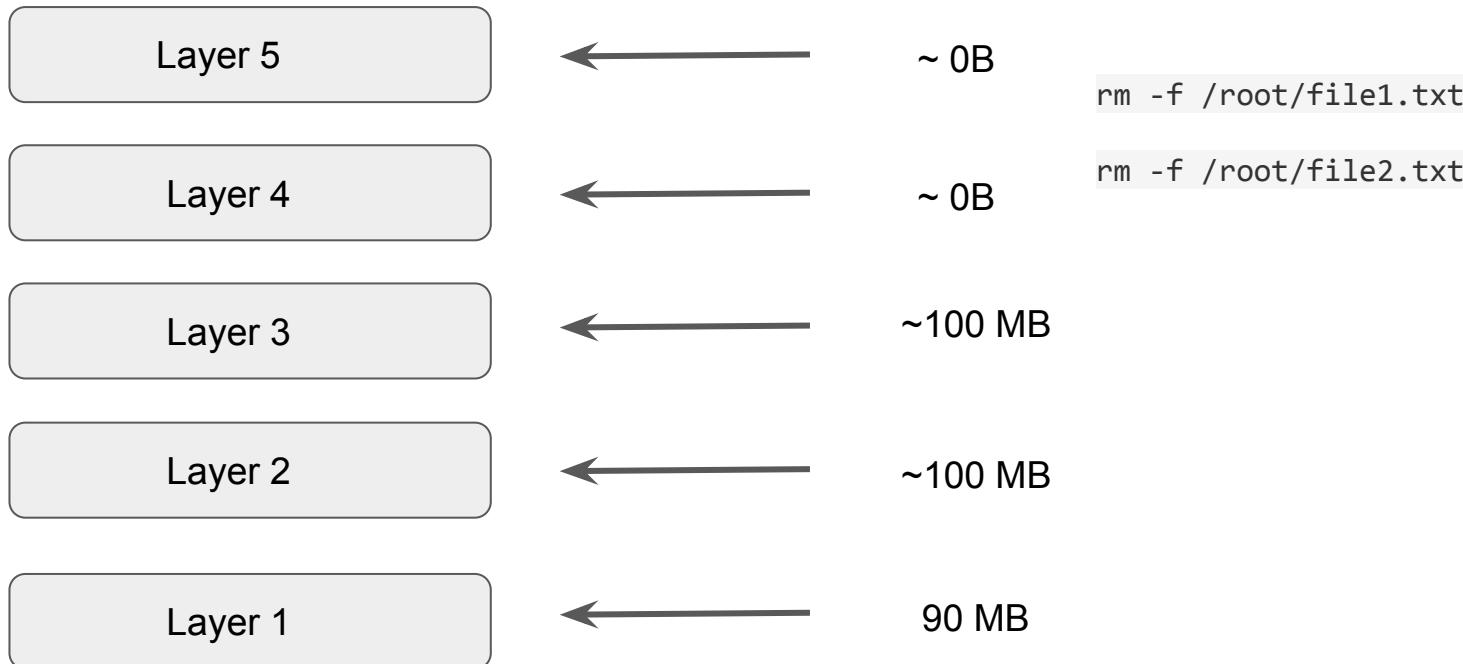
```
FROM ubuntu
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
RUN rm -f /root/file1.txt
RUN rm -f /root/file2.txt
```

First Three Instructions

```
FROM ubuntu  
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100  
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
```



ALL Instruction



Managing Images with CLI

Build once, use anywhere

Getting Started

Docker CLI can be used to manage various aspects related to Docker Images which includes building, removing, saving, tagging and others.

We should be familiar with the `docker image` child-commands

Child Commands for Docker Images

As of this date of recording, following child commands are available:

- docker image build
- docker image history
- docker image import
- docker image inspect
- docker image load
- docker image ls
- docker image prune
- docker image pull
- docker image push
- docker image rm
- docker image save
- docker image tag

Inspecting Docker Images

Build once, use anywhere

Getting Started

A Docker Image contains lots of information, some of these include:

- Creation Date
- Command
- Environment Variables
- Architecture
- OS
- Size

[`docker image inspect`](#) command allows us to see all the information associated with a docker image.

Docker Image Prune

Build once, use anywhere

Image Pruning Steps

`docker image prune` command allows us to clean up unused images.

By default, the above command will only clean up dangling images.

Dangling Images = Image without Tags and Image not referenced by any container

Modify Image to Single Layer

Build once, use anywhere

Getting Started

In a generic scenerio, the more the layers an image has, the more the size of the image.

Some of the image size goes from 5GB to 10GB.

Flattening an image to single layer can help reduce the overall size of the image.

Docker Registry

Building Docker Registry

Understanding Docker Registry

A [Registry](#) a stateless, highly scalable server side application that stores and lets you distribute Docker images.

Docker Hub is the simplest example that all of us must have used.

There are various types of registry available, which includes:

- Docker Registry
- Docker Trusted Registry
- Private Repository (AWS ECR)
- Docker Hub

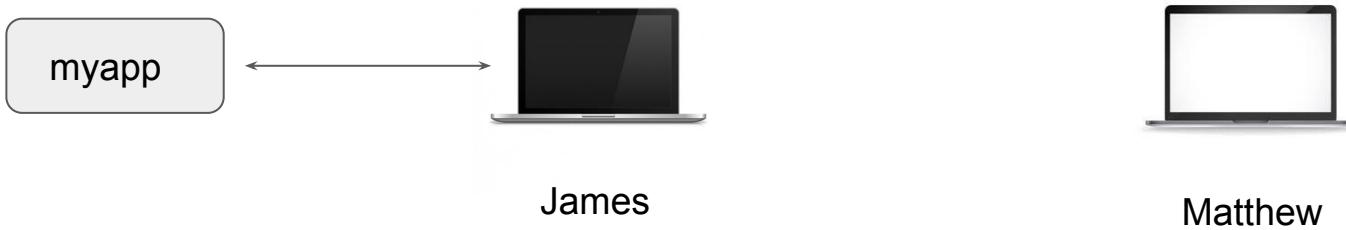
Moving Images Across Hosts

Build once, use anywhere

Example Use-Case

James has created an application based on Docker. He has the image file in his laptop.

He wants to send the image to Matthew over email.



Overview of Docker Save

The `docker save` command will save one or more images to a tar archive

Example Snippet:

```
docker save busybox > busybox.tar
```

Overview of Docker Load

The `docker load` command will load an image from a tar archive

Example Snippet:

```
docker load < busybox.tar
```

Build Cache

Build once, use anywhere

Overview Slide

Docker creates container images using layers.

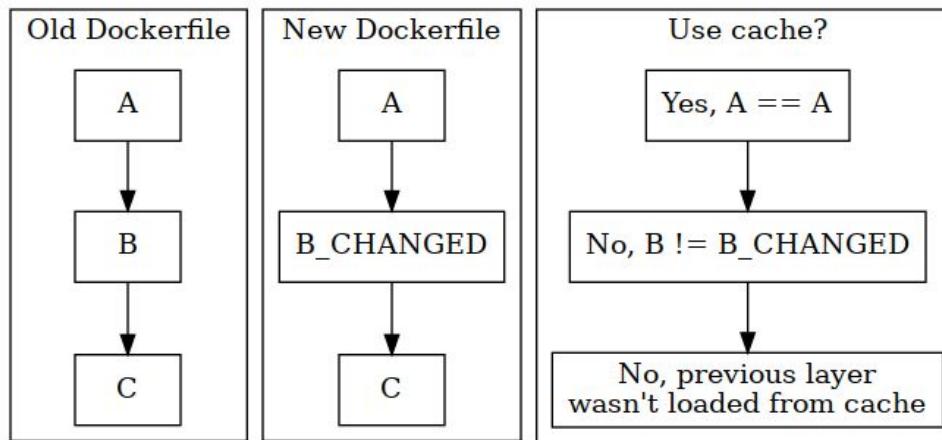
Each command that is found in a Dockerfile creates a new layer.

Docker uses a layer cache to optimize the process of building Docker images and make it faster.

```
[root@swarm02 build-cache]# docker build -t demo2
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM python:3.7-slim-buster
--> 87b1022604d5
Step 2/4 : COPY . .
--> Using cache
--> fe355c27a8ff
```

Important Pointer

If the cache can't be used for a particular layer, all subsequent layers won't be loaded from the cache.



Overview of Docker Networking

Build once, use anywhere

Getting Started

Docker takes care of the networking aspects so that container can communicate with other containers and also with the Docker Host.

```
[root@docker-demo ~]# ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
        inet6 fe80::42:b6ff:fea6:f62b prefixlen 64 scopeid 0x20<link>
              ether 02:42:b6:a6:f6:2b txqueuelen 0 (Ethernet)
              RX packets 147774 bytes 42989090 (40.9 MiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 153893 bytes 273579092 (260.9 MiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 139.59.82.72 netmask 255.255.240.0 broadcast 139.59.95.255
        inet6 fe80::8838:79ff:feed:1335 prefixlen 64 scopeid 0x20<link>
              ether 8a:38:79:ed:13:35 txqueuelen 1000 (Ethernet)
              RX packets 822197 bytes 2191970977 (2.0 GiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 634774 bytes 44815091 (42.7 MiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

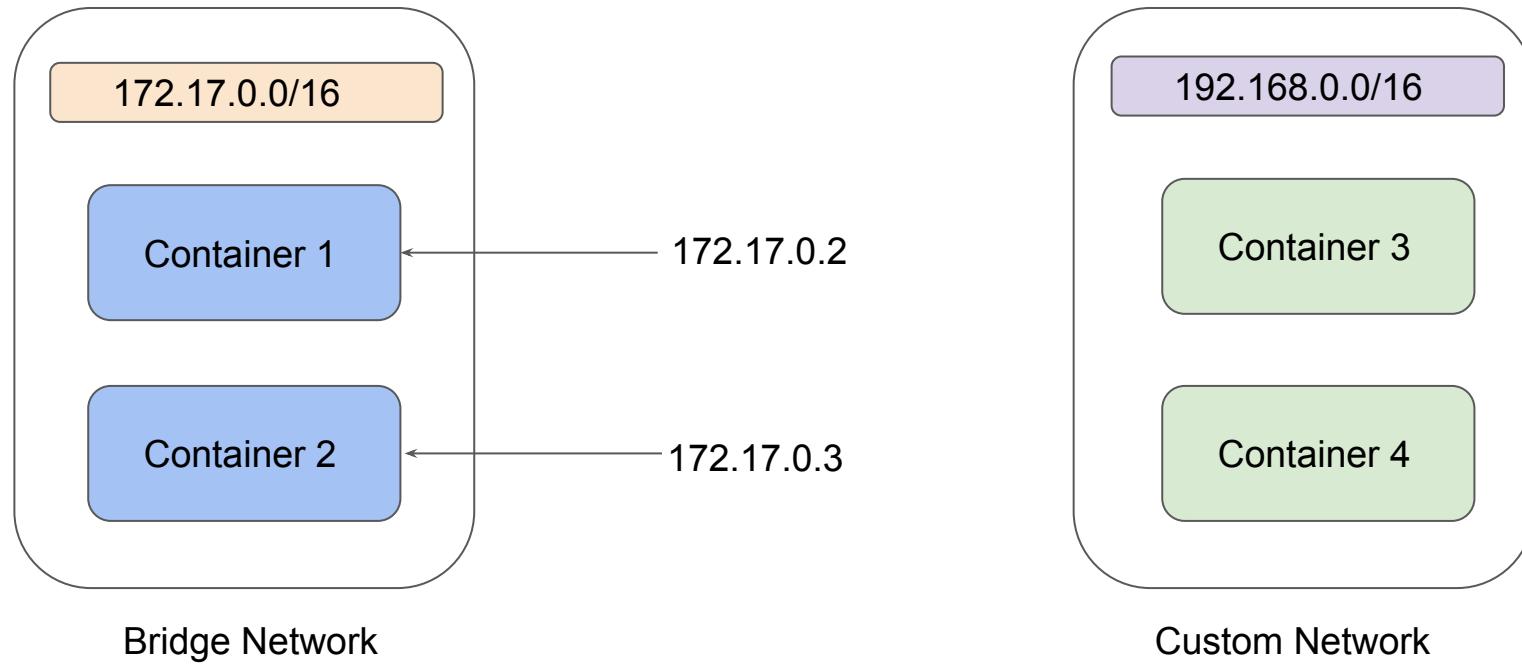
Overview of Network Drivers

Docker networking subsystem is pluggable, using drivers.

There are several drivers available by default, and provides core networking functionality.

- bridge
- host
- overlay
- macvlan
- none

Diagrammatic Representation

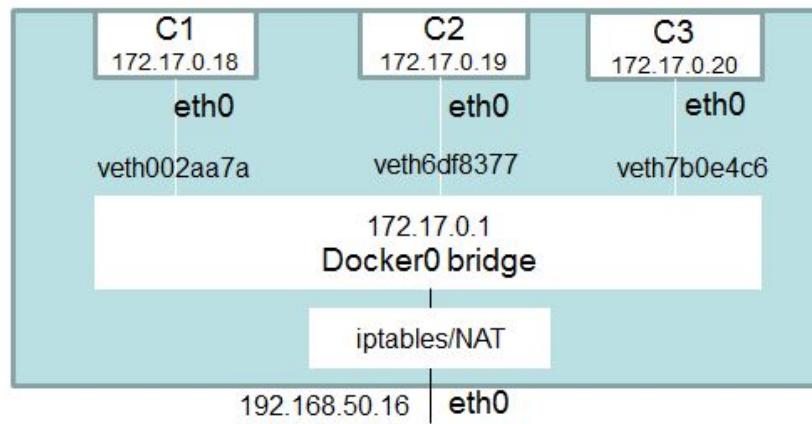


Bridge Network Driver

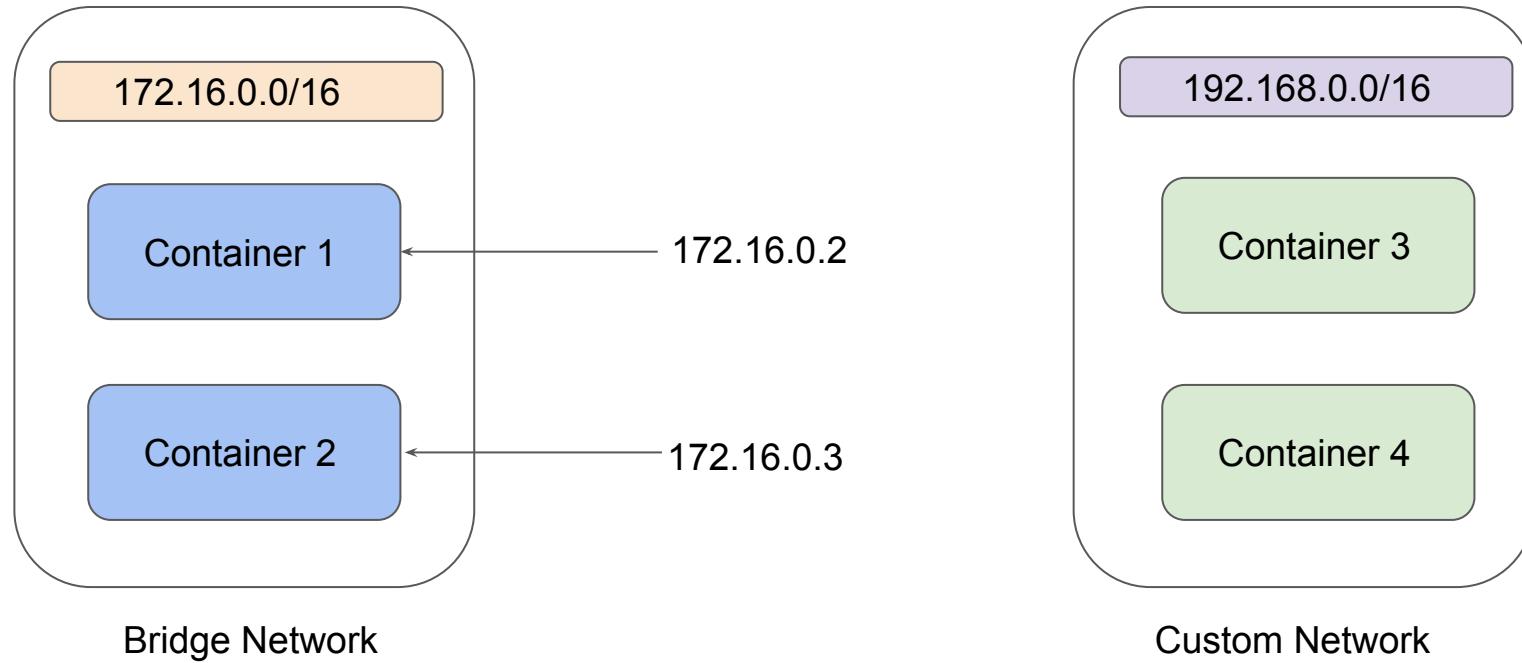
Build once, use anywhere

Overview of Bridge Network

A **bridge network** uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network.



Diagrammatic Representation



Overview of Network Drivers

Bridge is the default network driver for Docker.

If we do not specify a driver, this is the type of network you are creating.

When you start Docker, a default bridge network (also called bridge) is created automatically, and newly-started containers connect to it unless otherwise specified.

We also can create User-Defined Bridge Network which are superior to the default bridge.

Host Networks

Build once, use anywhere

Overview of Host Network

This driver removes the network isolation between the docker host and the docker containers to use the host's networking directly.

For instance, if you run a container which binds to port 80 and you use host networking, the container's application will be available on port 80 on the host's IP address.

None Network

Build once, use anywhere

Overview of None Network

If you want to completely disable the networking stack on a container, you can use the none network.

This mode will not configure any IP for the container and doesn't have any access to the external network as well as for other containers.

Publishing Exposed Ports of Container

Build once, use anywhere

Overview of Port Publishing

We were discussing about an approach to publishing container port to host.

```
docker container run -dt --name webserver -p 80:80 nginx
```

This is also referred as publish list as it publishes only list of port specified.

Overview of Port Publishing

There is also a second approach to publish all the exposed ports of the container.

```
docker container run -dt --name webserver -P nginx
```

This is also referred as a publish all .

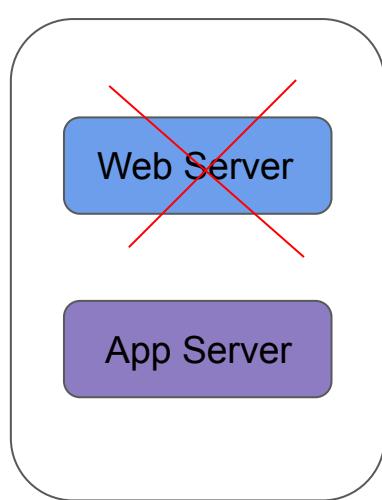
In this approach, all exposed ports are published to random ports of the host.

Container Orchestration

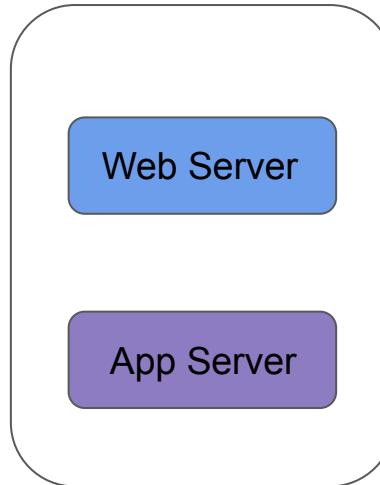
Build once, use anywhere

Getting Started

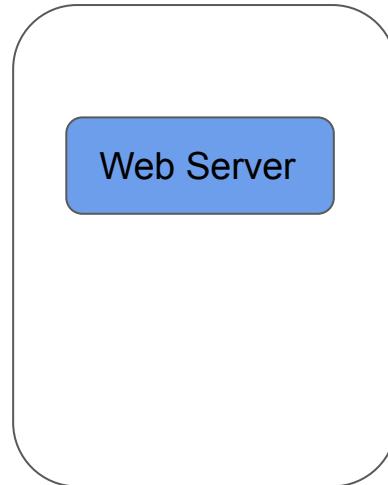
Container orchestration is all about managing the life cycles of containers, especially in large, dynamic environments.



VM 1

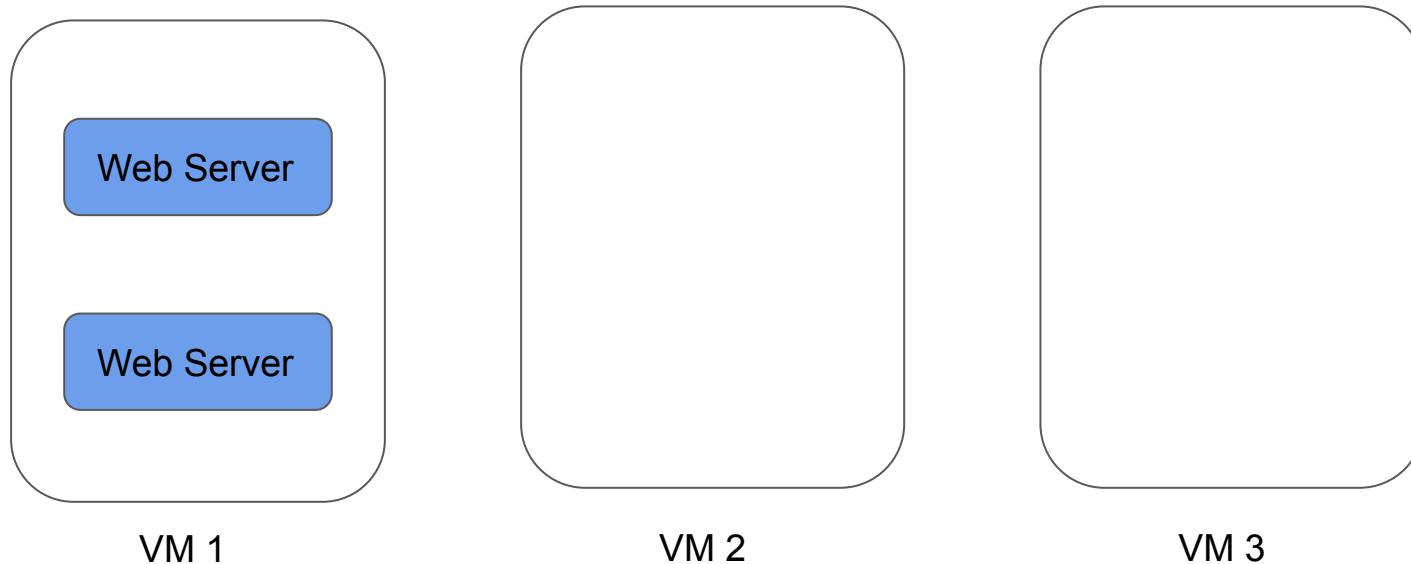


VM 2



VM 3

Requirement: Minimum of 2 web-server should be running all the time.



Importance of Container Orchestration

Container Orchestration can be used to perform lot of tasks, some of them includes:

- Provisioning and deployment of containers
- Scaling up or removing containers to spread application load evenly
- Movement of containers from one host to another if there is a shortage of resources
- Load balancing of service discovery between containers
- Health monitoring of containers and hosts

Container Orchestration Solutions

There are many container orchestration solutions which are available, some of the popular ones include:

- Docker Swarm
- Kubernetes
- Apache Mesos
- Elastic Container Service (AWS ECS)

There are also various container orchestration platforms available like EKS.

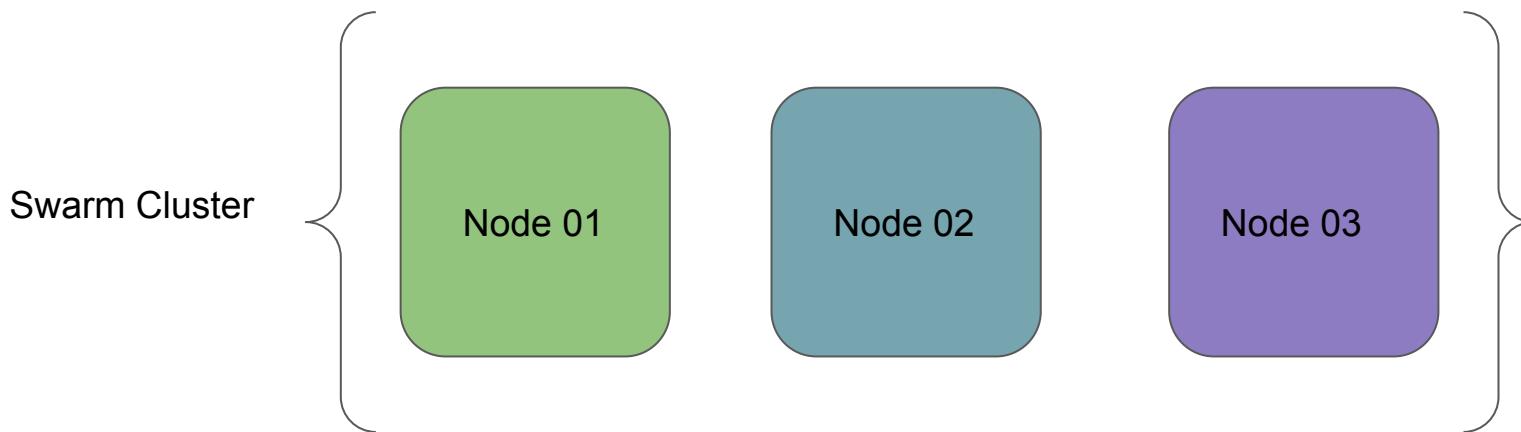
Overview of Docker Swarm

Build once, use anywhere

Getting Started

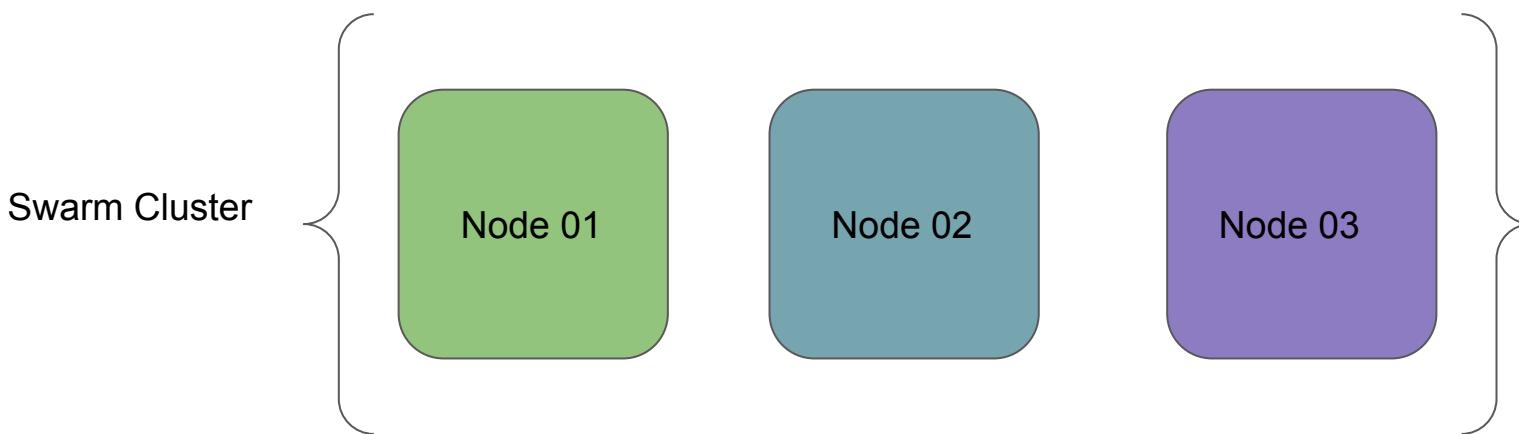
Docker Swarm is a container orchestration tool which is natively supported by Docker.

Our Lab Setup:



Initializing Docker Swarm

- A **node** is an instance of the Docker engine participating in the swarm.
- To deploy your application to a swarm, you submit a service definition to a **manager node**.
- The manager node dispatches units of work called tasks to **worker nodes**.

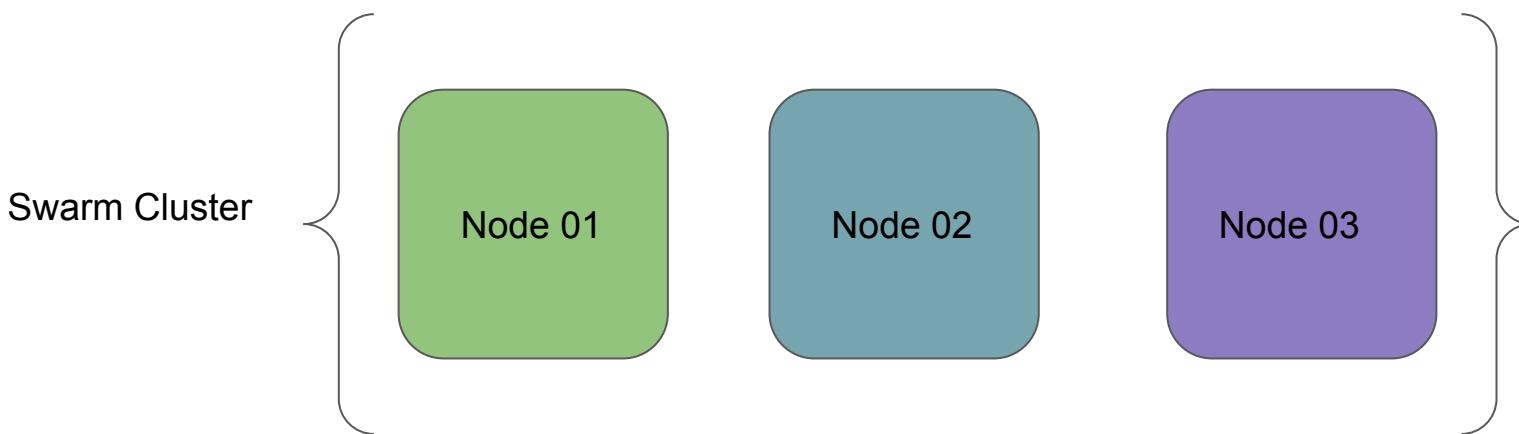


Initializing Docker Swarm

Build once, use anywhere

Initializing Docker Swarm

- A **node** is an instance of the Docker engine participating in the swarm.
- To deploy your application to a swarm, you submit a service definition to a **manager node**.
- The manager node dispatches units of work called tasks to **worker nodes**.



Initializing Docker Swarm

1. Manager Node Command:

```
docker swarm init --advertise-addr <MANAGER-IP>
```

2. Worker Nodes Command:

```
docker swarm join-token worker
```

Services, Tasks, Containers

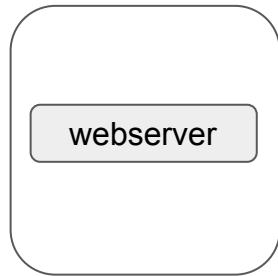
Build once, use anywhere

Getting Started

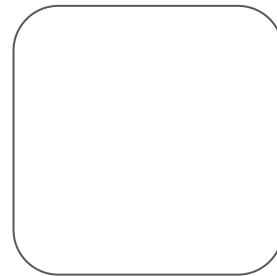
A service is the definition of the tasks to execute on the manager or worker nodes.

```
docker service create --name webserver --replicas 1 nginx
```

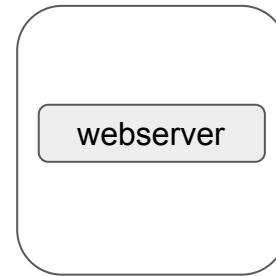
Swarm Cluster



Node 01

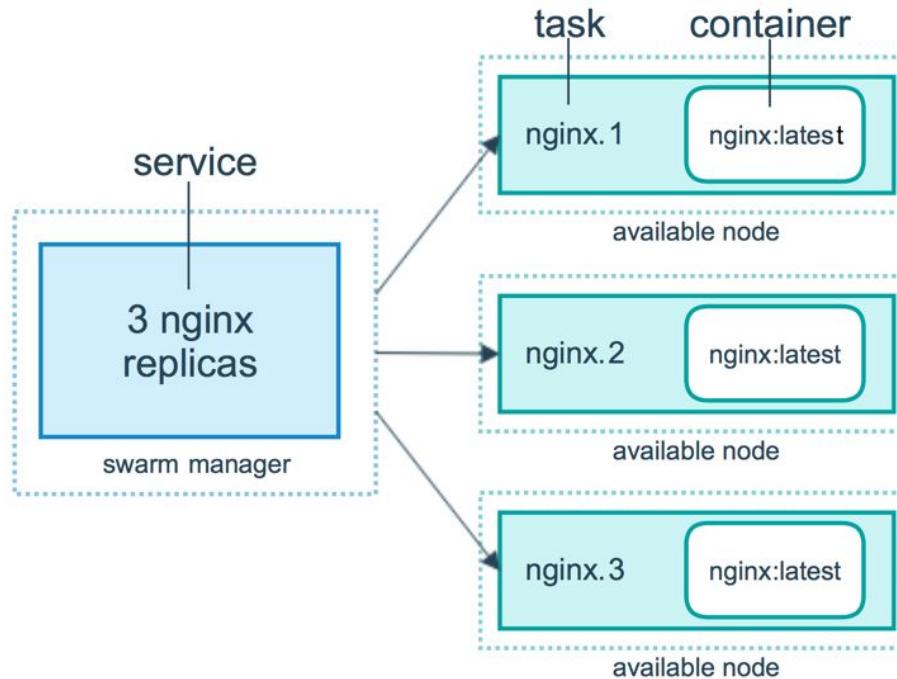


Node 02



Node 03

Services, Tasks and Containers



Scaling Service in Swarm

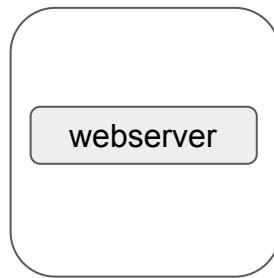
Build once, use anywhere

Getting Started

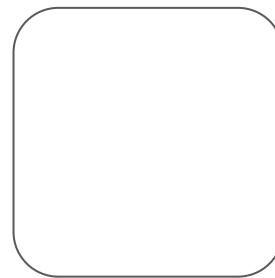
Once you have deployed a service to a swarm, you are ready to use the Docker CLI to scale the number of containers in the service.

Containers running in a service are called “tasks.”

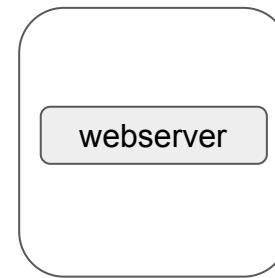
Swarm Cluster



Node 01

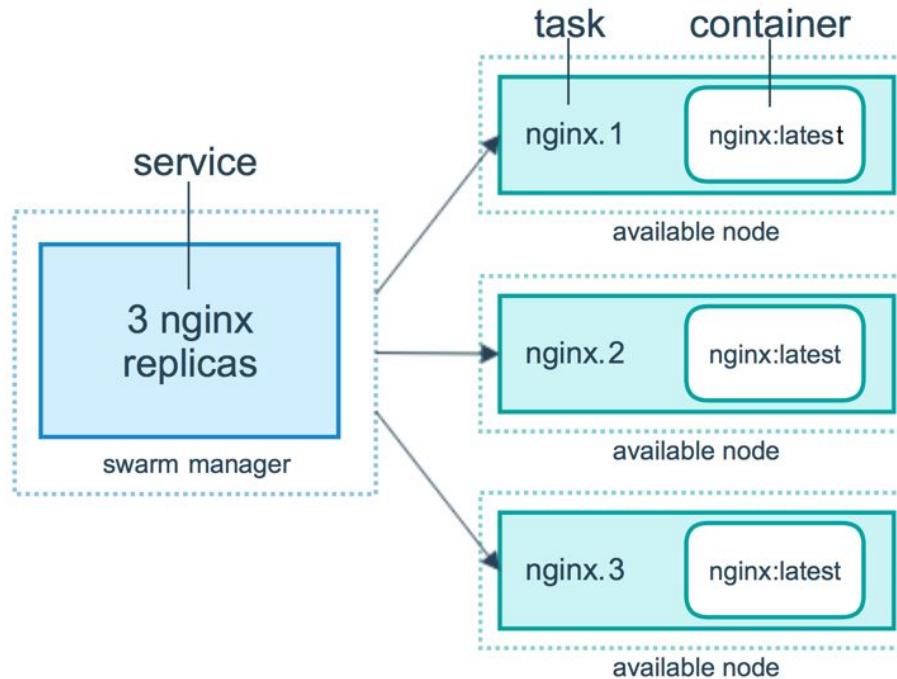


Node 02



Node 03

Services, Tasks and Containers

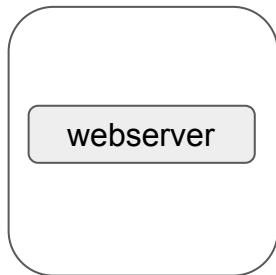


Two Commands to Scale Service

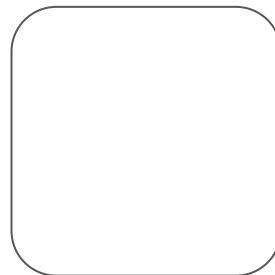
There are two ways in which you can scale service in swarm:

- docker service scale webserver=5
- docker service update --replicas 5 mywebsever

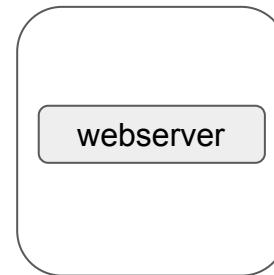
Swarm Cluster



Node 01



Node 02



Node 03

Replicated vs Global Service

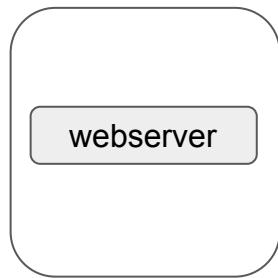
Build once, use anywhere

Getting Started

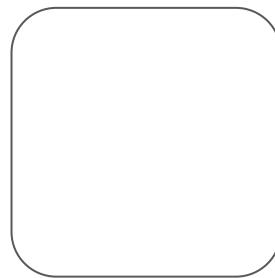
There are two types of services deployments, **replicated** and **global**

For a replicated service, you specify the number of identical tasks you want to run. For example, you decide to deploy an NGINX service with two replicas, each serving the same content.

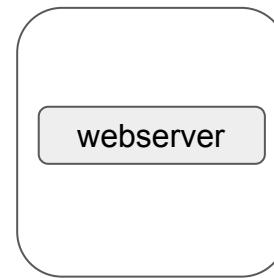
Swarm Cluster



Node 01



Node 02



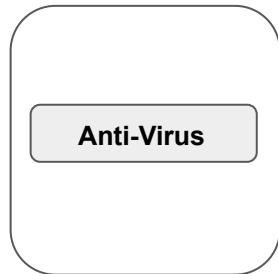
Node 03

Global Service

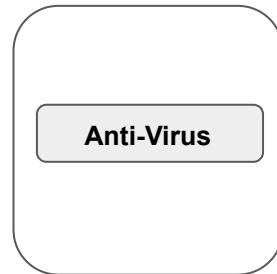
A global service is a service that runs one task on every node.

Each time you add a node to the swarm, the orchestrator creates a task and the scheduler assigns the task to the new node.

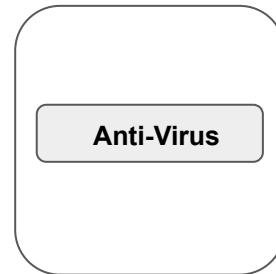
Swarm Cluster



Node 01



Node 02



Node 03

Inspecting - Swarm Services & Nodes

Build once, use anywhere

Overview of Docker Inspect

Docker Inspect provides detailed information of the Docker Objects.

Some of these Docker Object includes:

- Docker Containers
- Docker Network
- Docker Volumes
- Docker Swarm Services
- Docker Swarm Nodes

Overview of Docker Compose

Build once, use anywhere

Getting Started

Compose is a tool for defining and running multi-container Docker applications.

With Compose, you use a [YAML](#) file to configure your application's services.

We can start all the services with single command - docker compose up

We can stop all the services with single command - docker compose down

Deploying Multi-Service Apps in Swarm

Build once, use anywhere

Getting Started

A specific web-application might have multiple containers that are required as part of the build process.

Whenever we make use of docker service, it is typically for a single container image.

The [docker stack](#) can be used to manage a multi-service application.

Overview of Docker Stack

A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together.

A stack can compose **YAML** file like the one that we define during Docker Compose.

We can define everything within the YAML file that we might define while creating a Docker Service.

Locking Swarm Cluster

Build once, use anywhere

Getting Started

Swarm Cluster **contains lot of sensitive information**, which includes:

- TLS key used to encrypt communication among swarm node
- Keys used to encrypt and decrypt the Raft logs on disk

If your Swarm is compromised and if data is stored in plain-text, an attack can get all the sensitive information.

Docker Lock allows us to have control over the keys.

Troubleshoot Service Deployments

Build once, use anywhere

Getting Started

A service may be configured in such a way that no node currently in the swarm can run its tasks.

In this case, the service remains in state **pending**

There are multiple reasons why service might go into pending state

docker service ps demotroubleshoot						
ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
ulggnljj0ber	demotroubleshoot.1	nginx:latest		Running	Pending 17 seconds ago	"no suitable node (scheduling ..."
g819kx5dqp9b	demotroubleshoot.2	nginx:latest		Running	Pending 17 seconds ago	"no suitable node (scheduling ..."
k83f9wo0u6xc	demotroubleshoot.3	nginx:latest		Running	Pending 17 seconds ago	"no suitable node (scheduling ..."

Examples when services go in Pending State

If all nodes are drained, and you create a service, it is pending until a node becomes available.

You can reserve a specific amount of memory for a service. If no node in the swarm has the required amount of memory, the service remains in a pending state until a node is available which can run its tasks.

You have imposed some kind of placement constraints

Control Service Placement

Build once, use anywhere

Overview of Placement Constraints

Swarm services provide a few different ways for you to control scale and placement of services on different nodes.

- Replicated and Global Services
- Resource Constraints [requirement of CPU and Memory]
- Placement Constraints [only run on nodes with label `pci_compliance = true`]
- Placement Preferences

Overlay Network

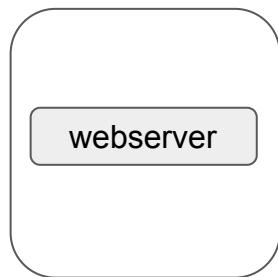
Build once, use anywhere

Overview of Overlay Networks

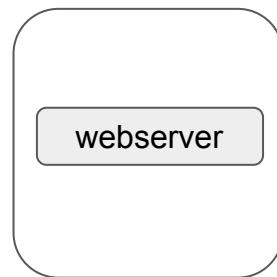
The overlay network driver creates a distributed network among multiple Docker daemon hosts

Overlay network allows containers connected to it to communicate securely.

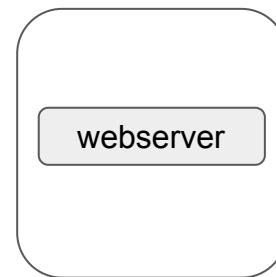
Swarm Cluster



Node 01



Node 02



Node 03

Secure Overlay Networks

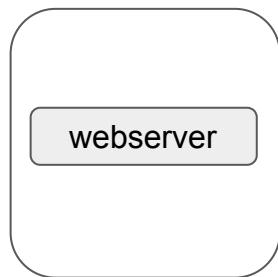
Build once, use anywhere

Overview of Overlay Networks

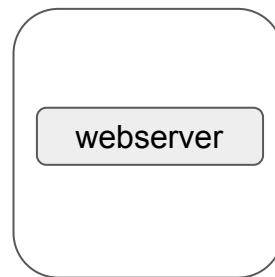
The overlay network driver creates a distributed network among multiple Docker daemon hosts

Overlay network allows containers connected to it to communicate securely.

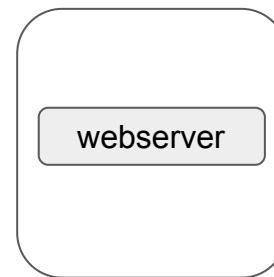
Swarm Cluster



Node 01



Node 02



Node 03

Overview of Secure Overlay Network

For the overlay networks, the containers can be spreaded across multiple servers.

If the containers are communicating with each other, it is recommended to secure the communication.

To enable encryption, when you create an overlay network pass the --opt encrypted flag:

```
docker network create --opt encrypted --driver overlay my-overlay-secure-network
```

Important Pointers

When you enable overlay encryption, Docker creates IPSEC tunnels between all the nodes where tasks are scheduled for services attached to the overlay network.

These tunnels also use the AES algorithm in GCM mode and manager nodes automatically rotate the keys every 12 hours.

Overlay network encryption is not supported on Windows. If a Windows node attempts to connect to an encrypted overlay network, no error is detected but the node will not be able to communicate.

Creating Services using Templates

Build once, use anywhere

Getting Started

We can make use of **templates** while running the **service create** command in swarm.

Let us understand this with an example:

```
docker service create --name demoservice  
--hostname="{{.Node.Hostname}}-{{.Service.Name}}" nginx
```

Valid Placeholders

Placeholder	Description
Service.ID	Service ID
.Service.Name	Service name
.Service.Labels	Service labels
.Node.ID	Node ID
.Node.Hostname	Node Hostname
.Task.ID	Task ID
.Task.Name	Task name
.Task.Slot	Task slot

Supported Flags

There are three supported flags for the placeholder templates:

--hostname

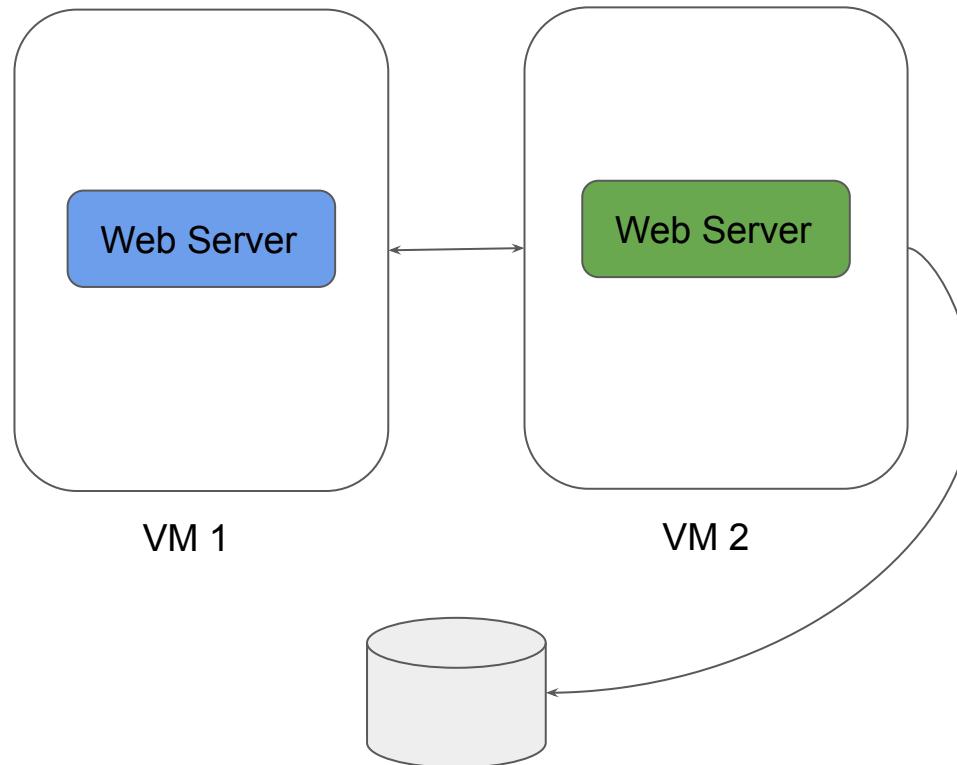
--mount

--env

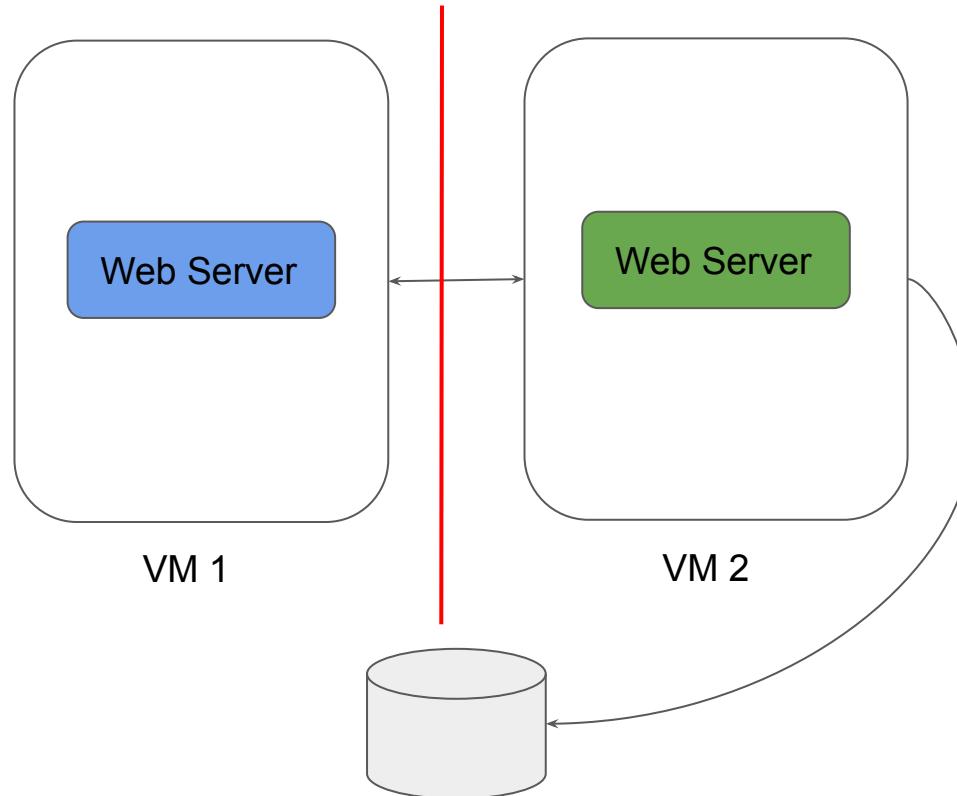
Split-Brain & Quorum

Build once, use anywhere

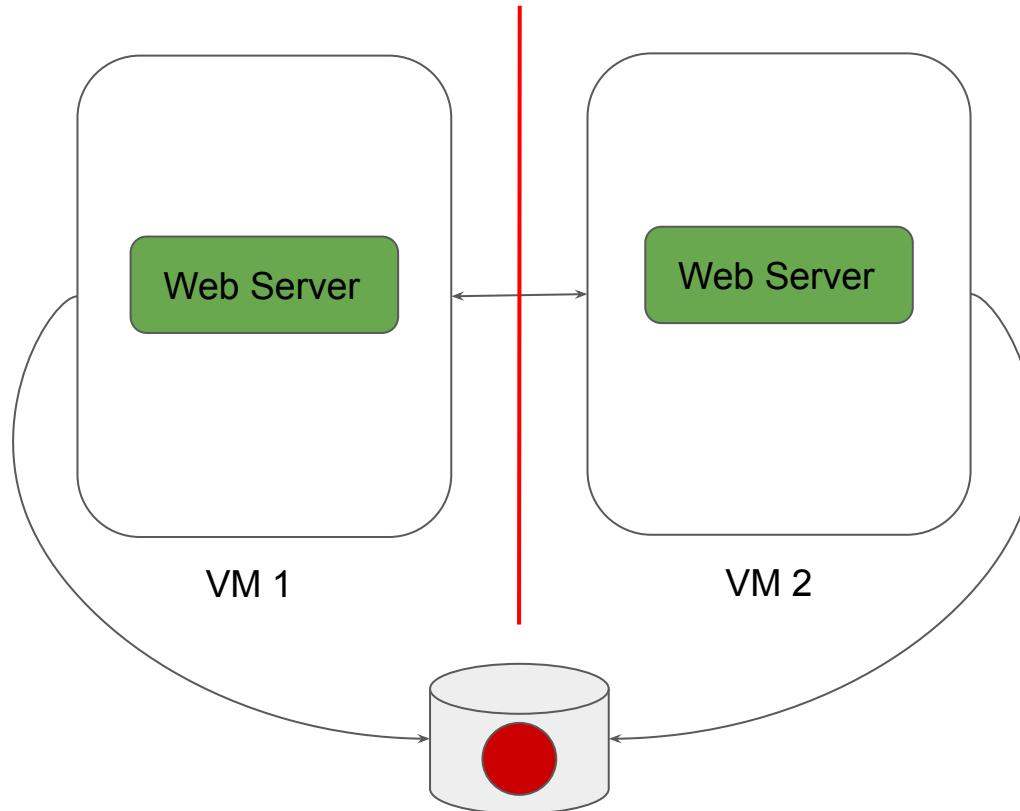
Split Brain Problem



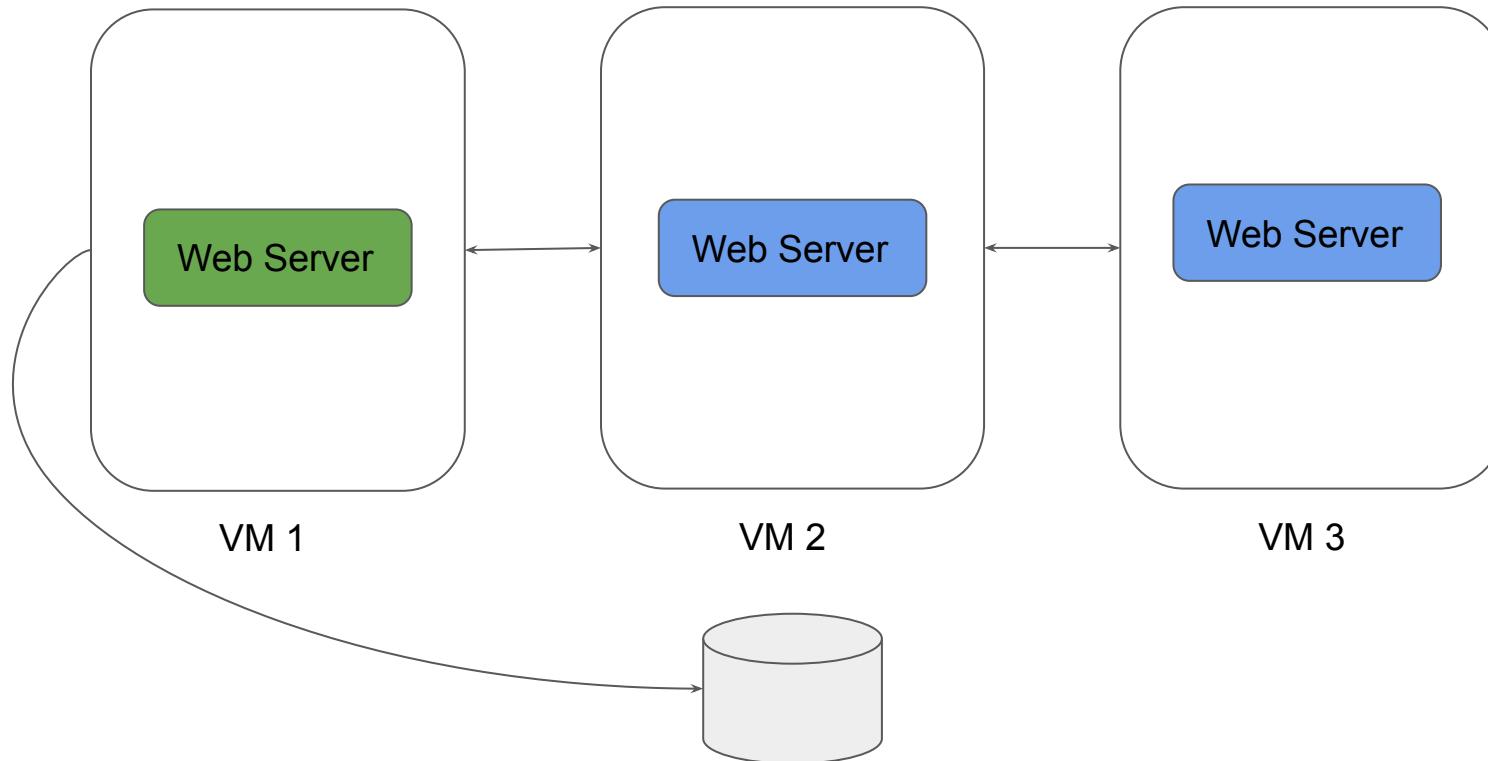
Split Brain Problem



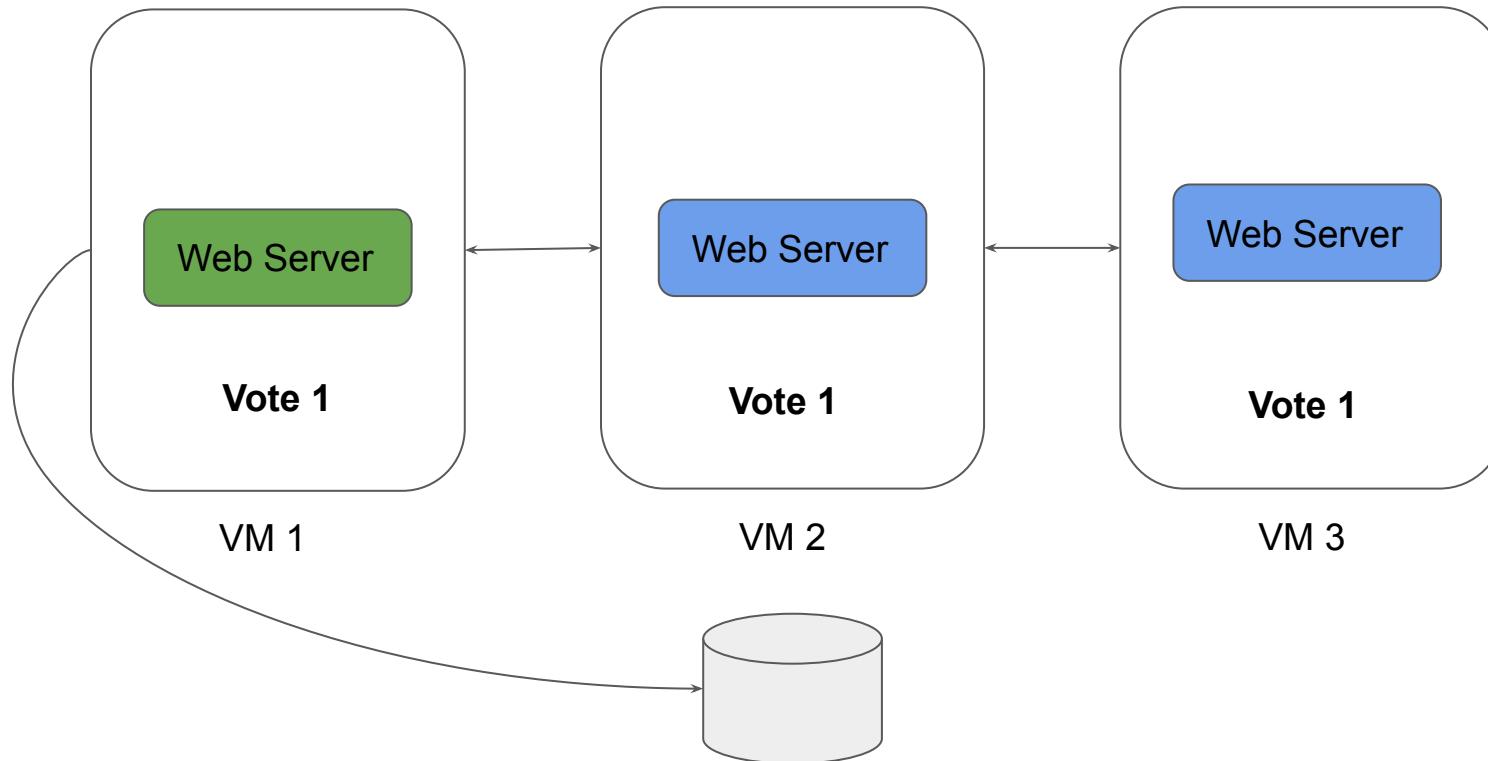
Split Brain Problem



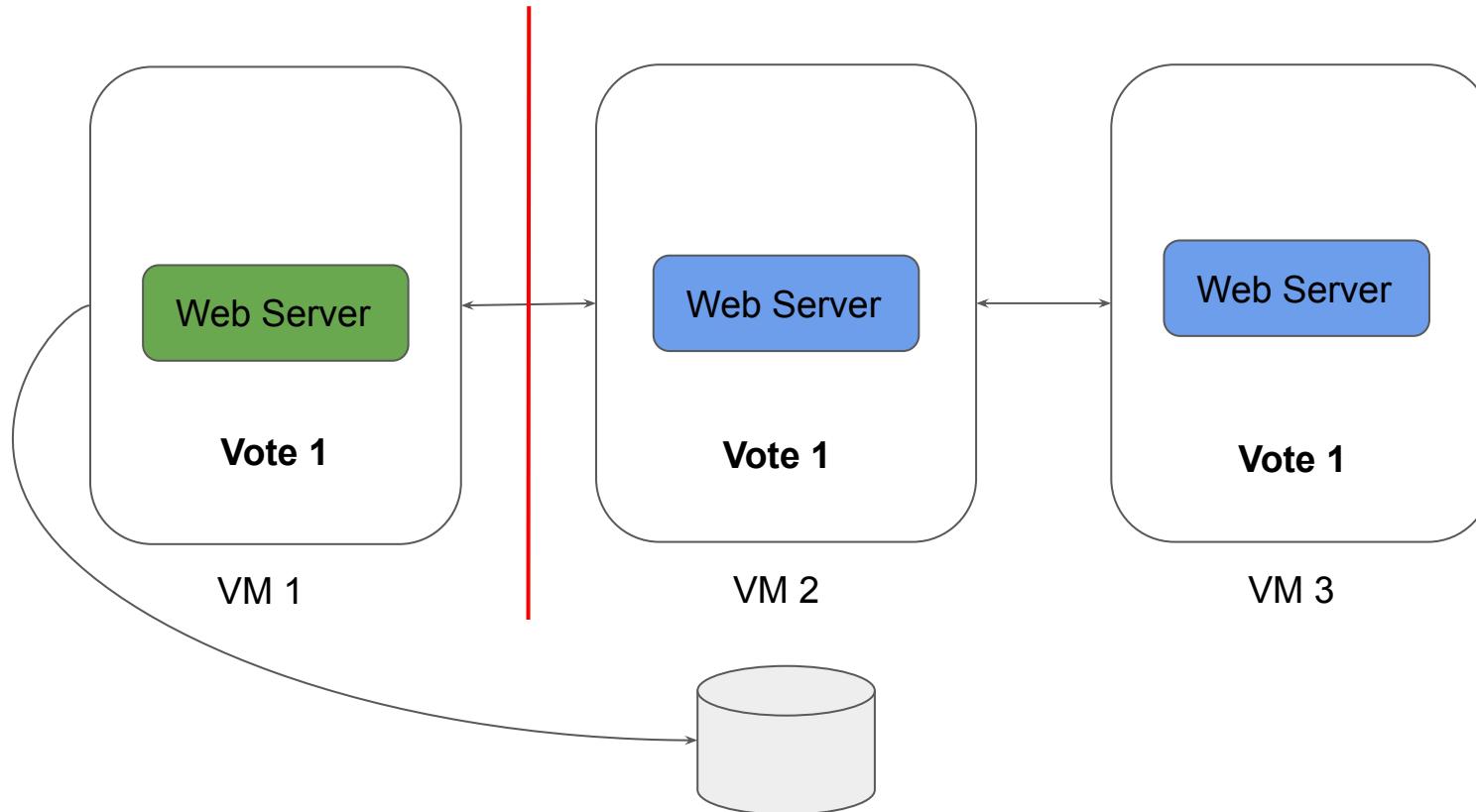
Right Quorum to the Rescue



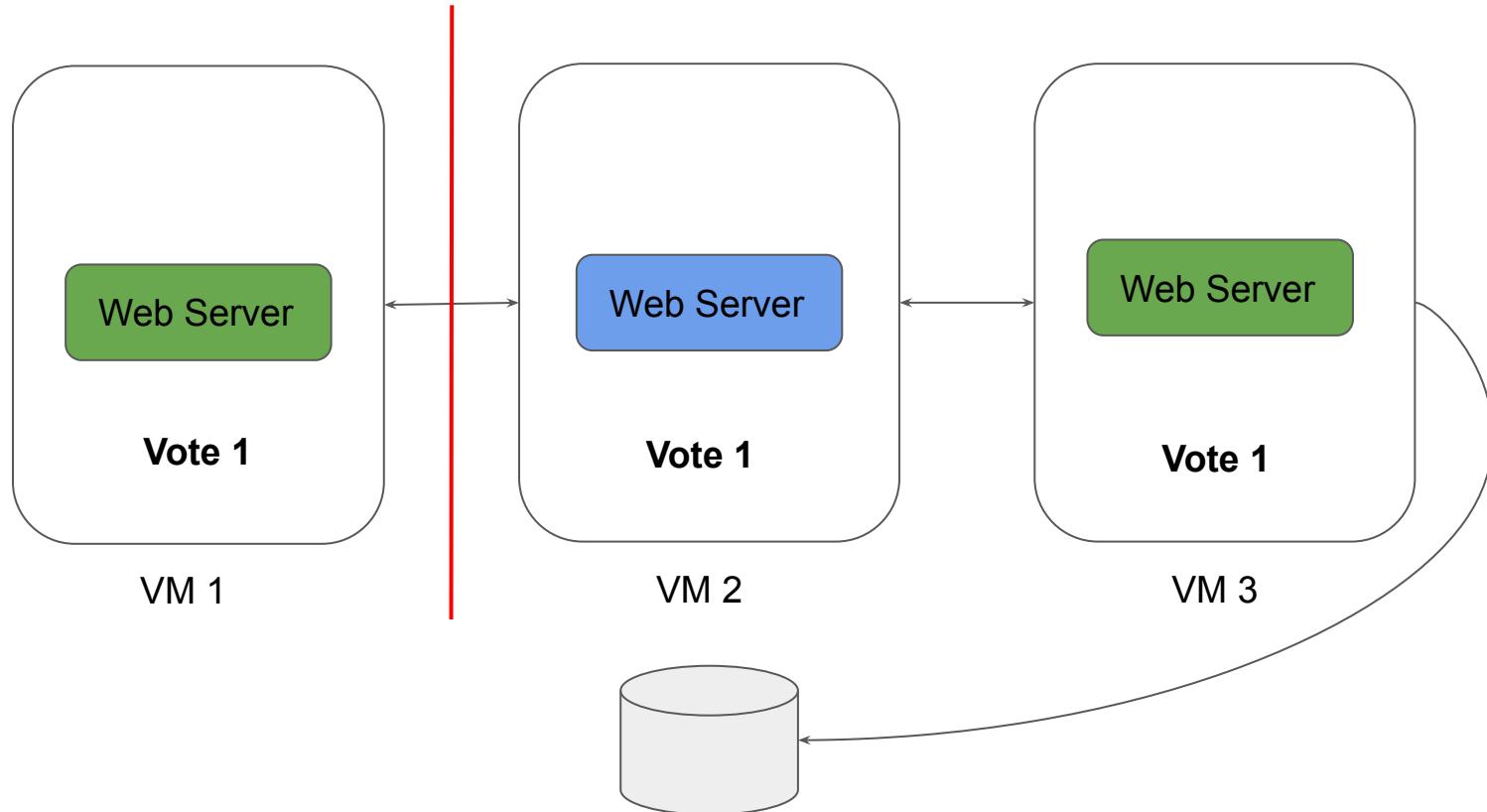
Right Quorum to the Rescue



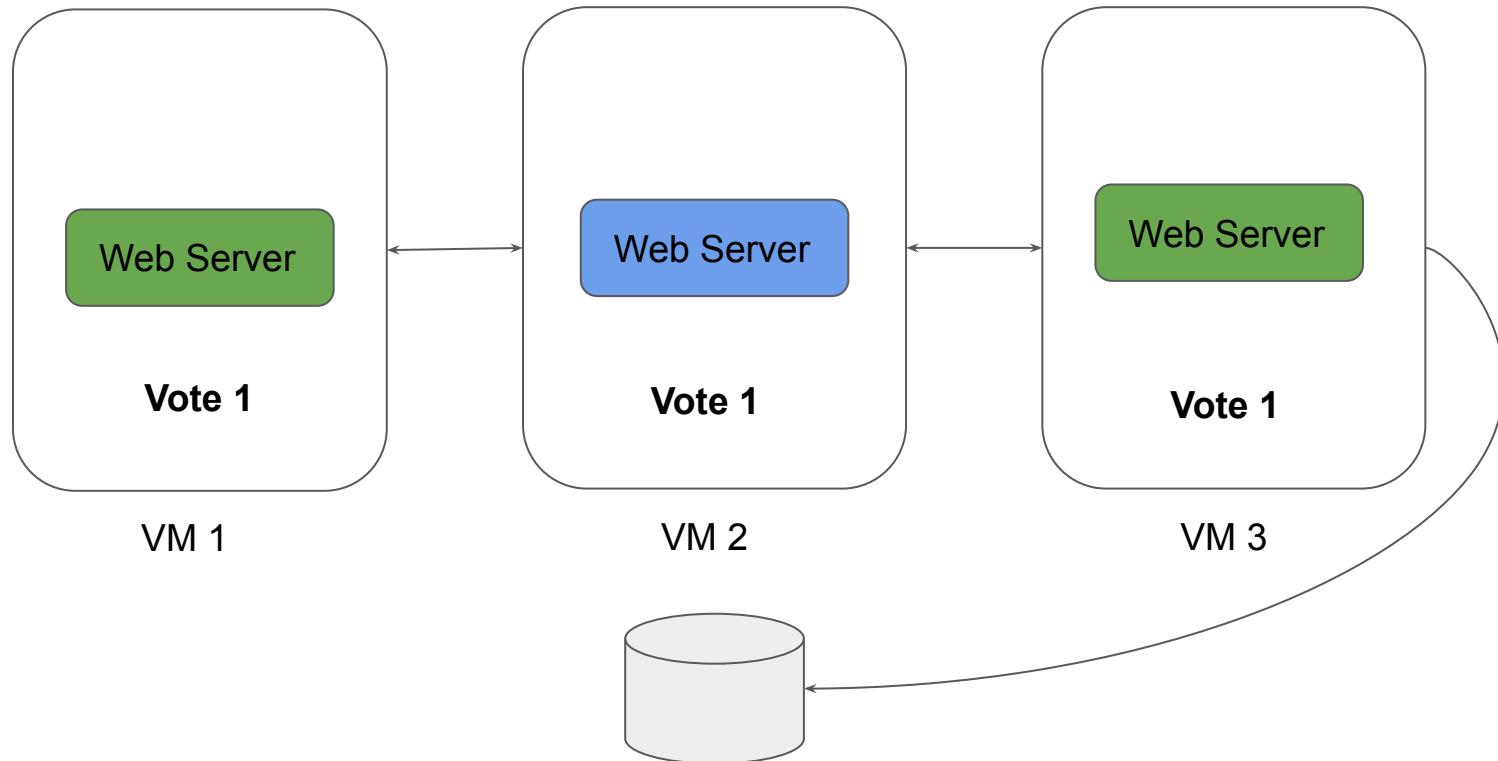
Right Quorum to the Rescue



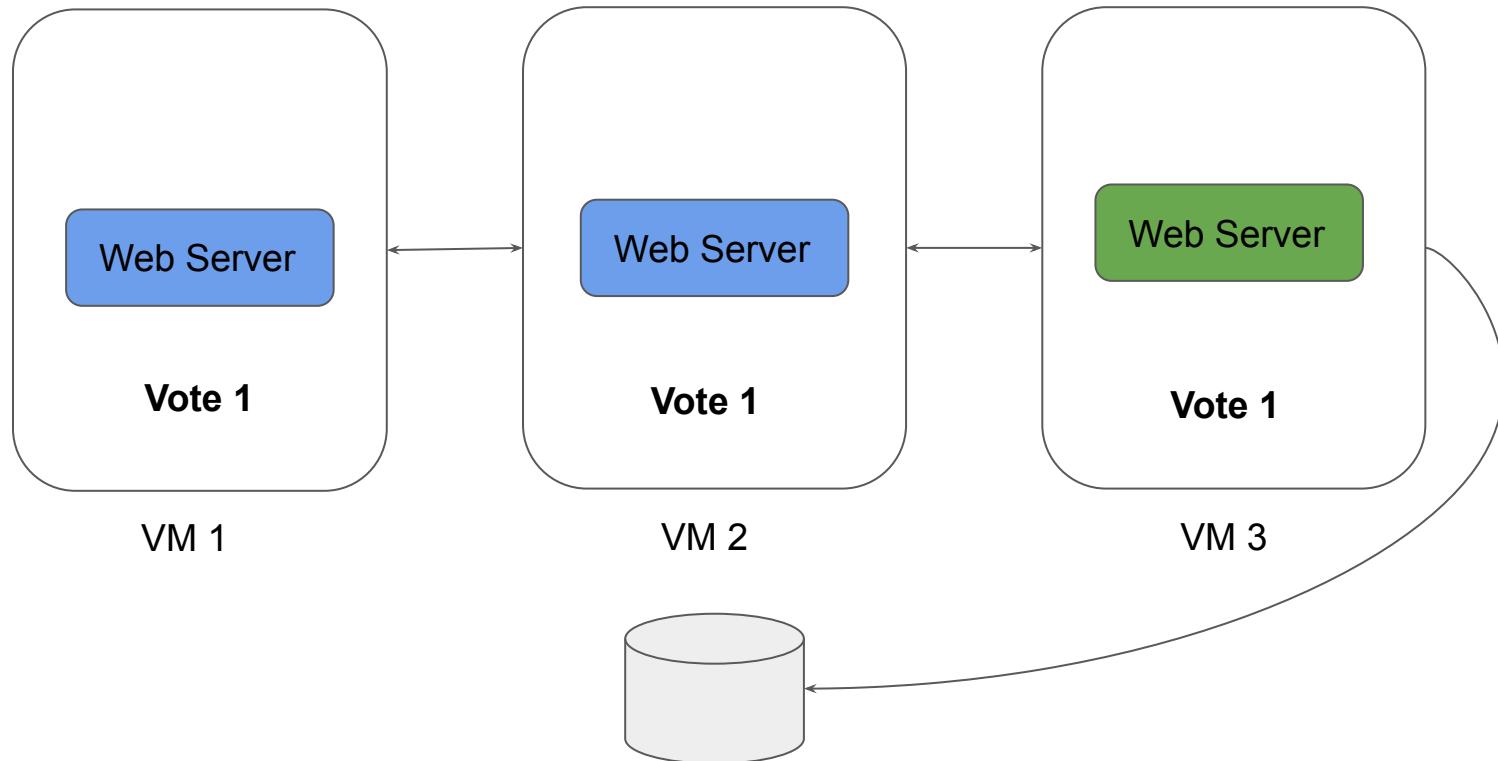
Right Quorum to the Rescue



Right Quorum to the Rescue



Right Quorum to the Rescue



Important Pointers for Quorum

We should maintain an odd number of nodes within the cluster.

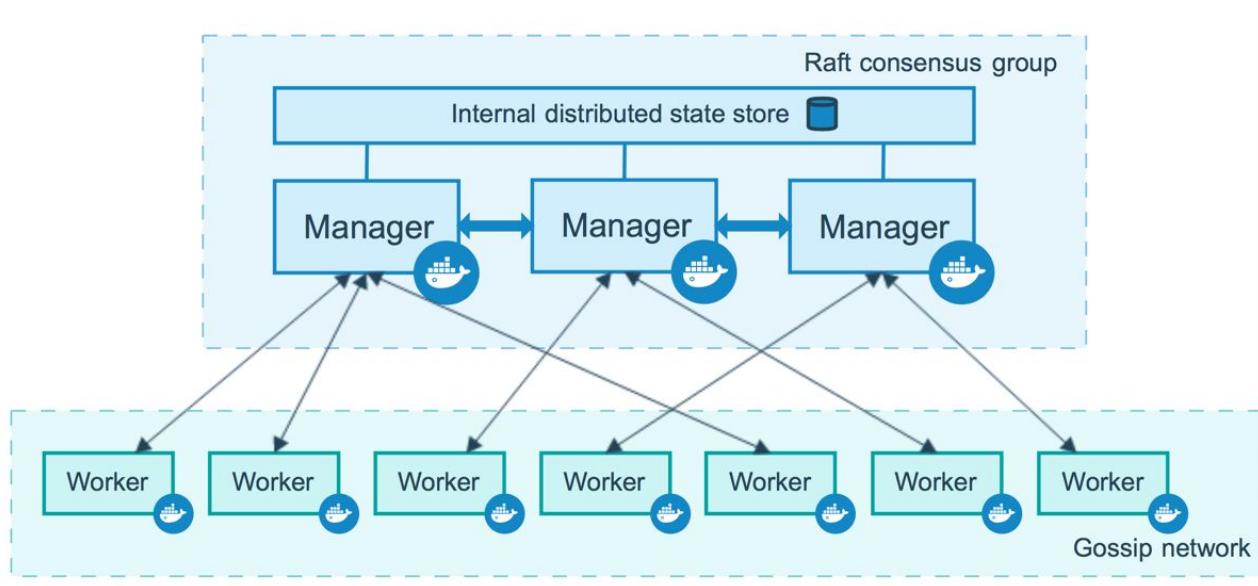
Cluster Size	Majority	Fault Tolerance
1	0	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

Swarm Manager Node HA

Build once, use anywhere

Getting Started

Manager Nodes are responsible for handling the cluster management tasks.



Importance of Container Orchestration

Manager Node has many responsibility within swarm, these include:

- maintaining cluster state
- scheduling services
- serving swarm mode HTTP API endpoints

Using a Raft implementation, the managers maintain a consistent internal state of the entire swarm and all the services running on it

High Availability in Swarm

Swarm comes with its own fault tolerance features.

Docker recommends you implement an odd number of nodes according to your organization's high-availability requirements.

Using a Raft implementation, the managers maintain a consistent internal state of the entire swarm and all the services running on it

An N manager cluster tolerates the loss of at most $(N-1)/2$ managers.

Introduction to Kubernetes

Orchestrator Engine

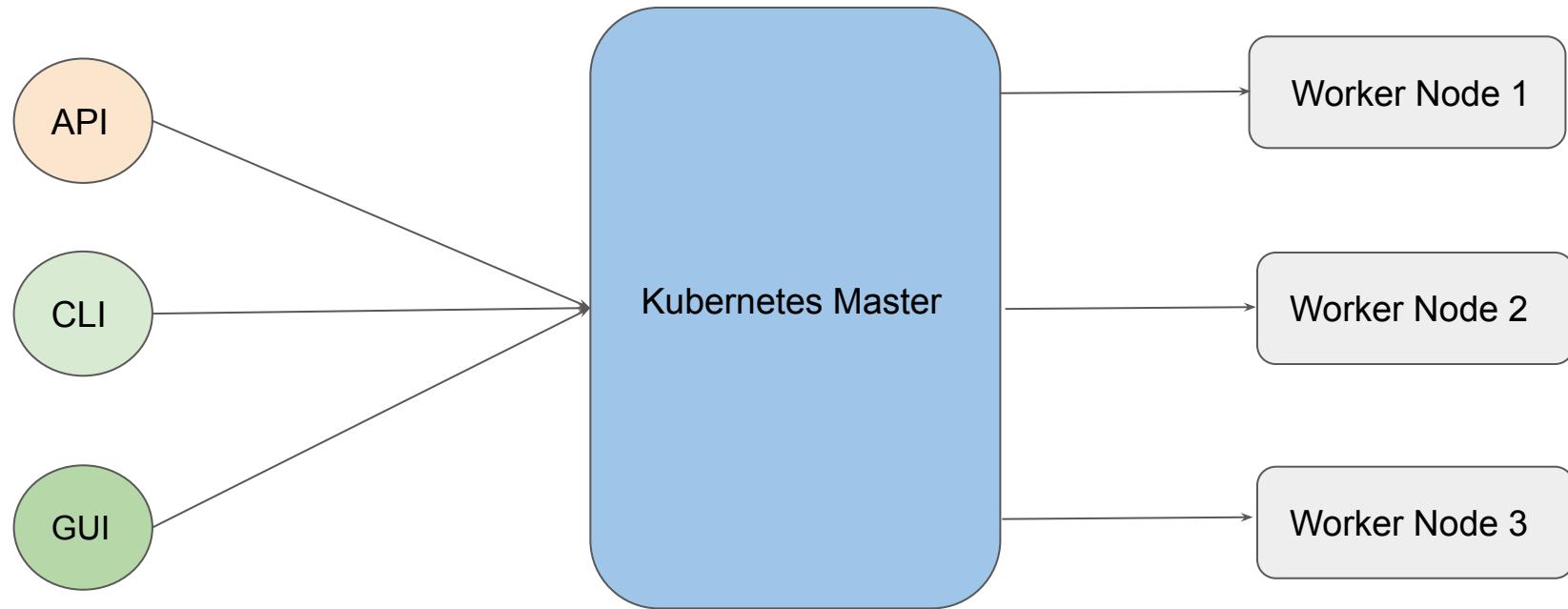
Introduction to Kubernetes

Kubernetes (K8s) is an open-source container orchestration engine developed by Google.

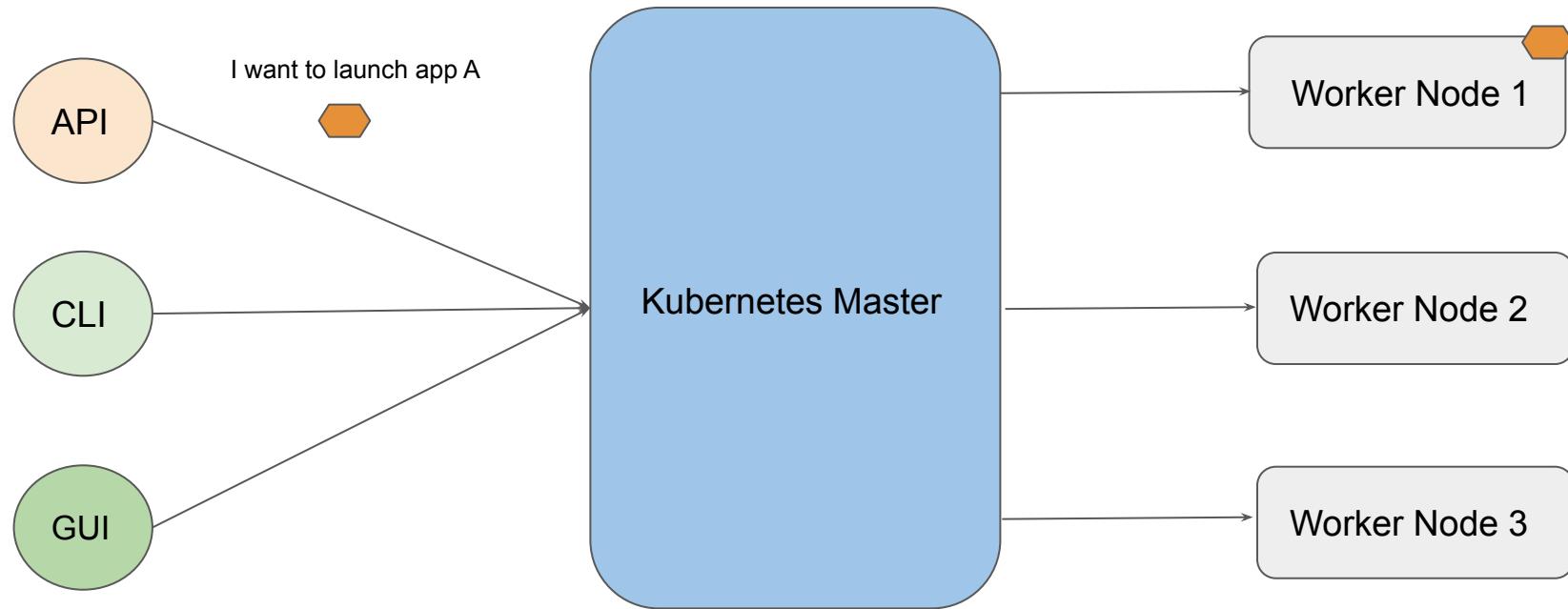
It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation.



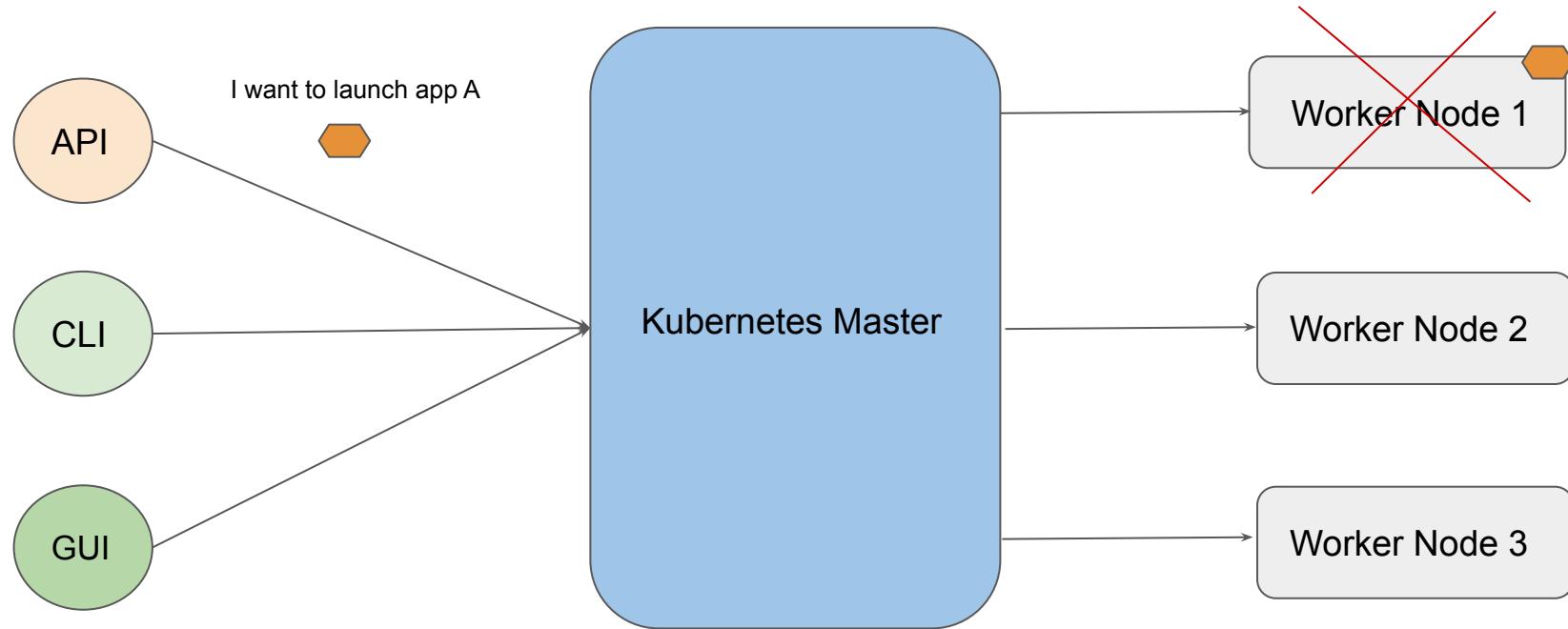
Architecture of Kubernetes



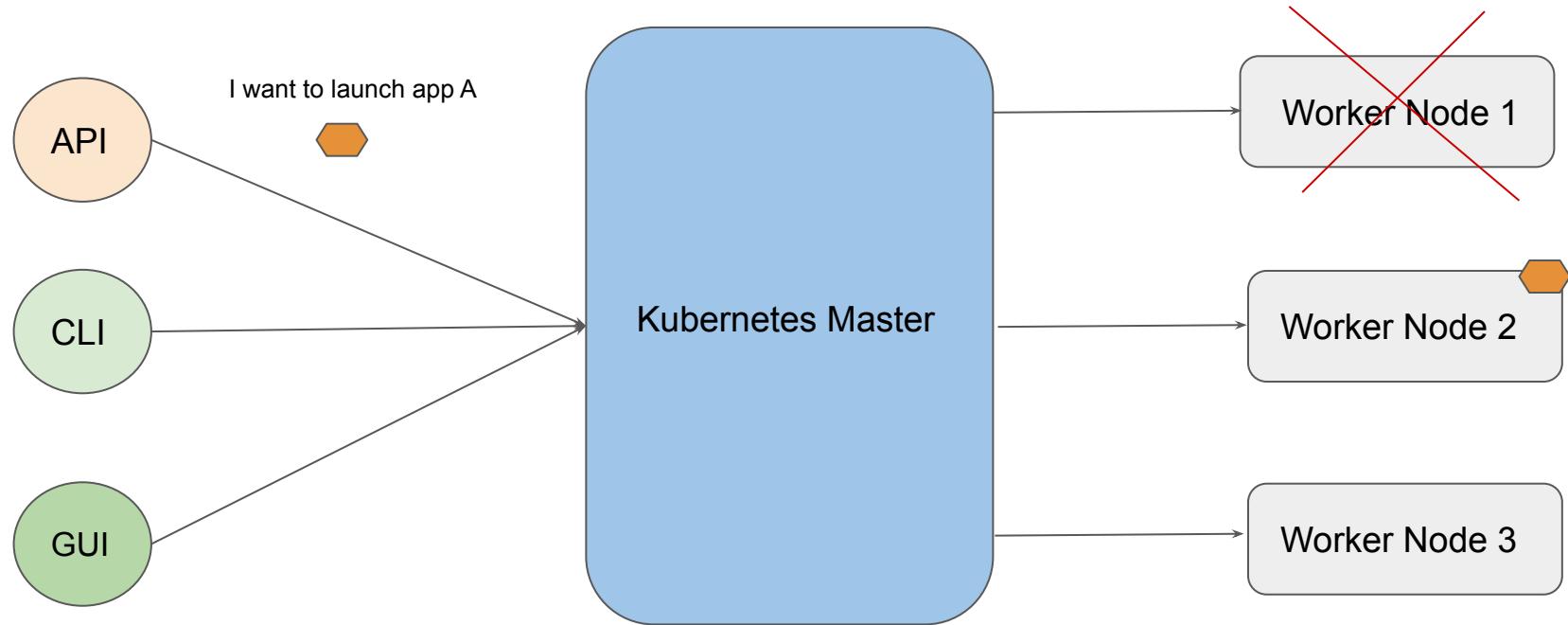
Architecture of Kubernetes



Architecture of Kubernetes



Architecture of Kubernetes



Installation Options

Let's Get Started

Understanding Installation Methods

There are multiple ways to get started with fully functional kubernetes environment

1. Use the Managed Kubernetes Service
2. Use Minikube
3. Install & Configure Kubernetes Manually (Hard Way)

1. Managed Kubernetes Service

Various providers like AWS, IBM, GCP and others provides managed Kubernetes clusters.

Most organizations prefer to make use of this approach.



2. Minikube

Minikube is a tool that makes it easy to run Kubernetes locally.

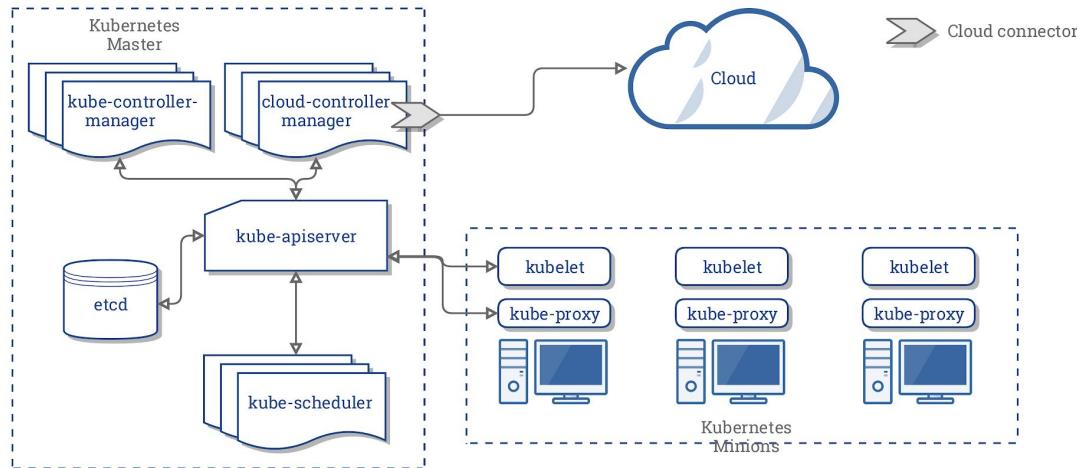
Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.



minikube

Install & Configure Kubernetes Manually (Hard Way)

In this approach, you install and configure components of Kubernetes individually.

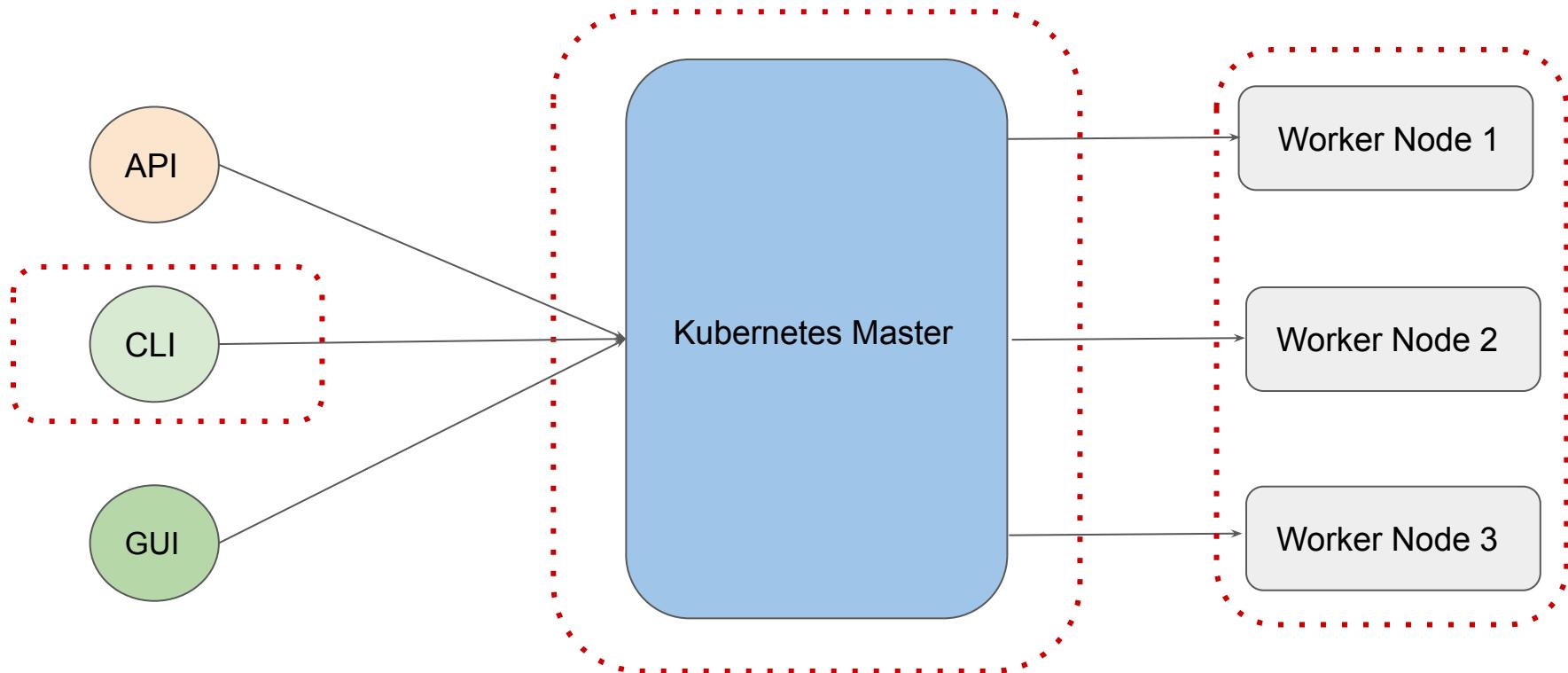


Installation Aspects

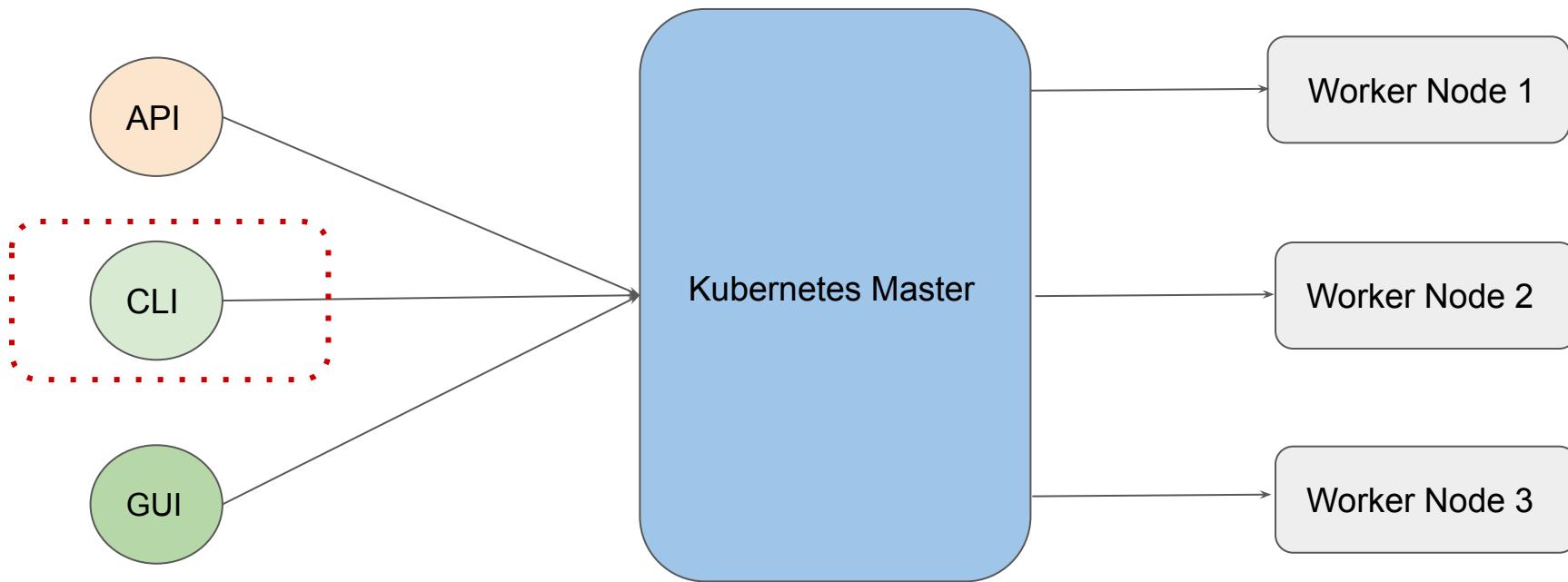
Things to configure while working with Kubernetes.

Sr No	Things to Install	Description
1	kubectl	CLI for running user commands against cluster.
2	Kubernetes Master	Kubernetes Cluster by itself.
3	Worker Node Agents	Kubernetes Node Agent

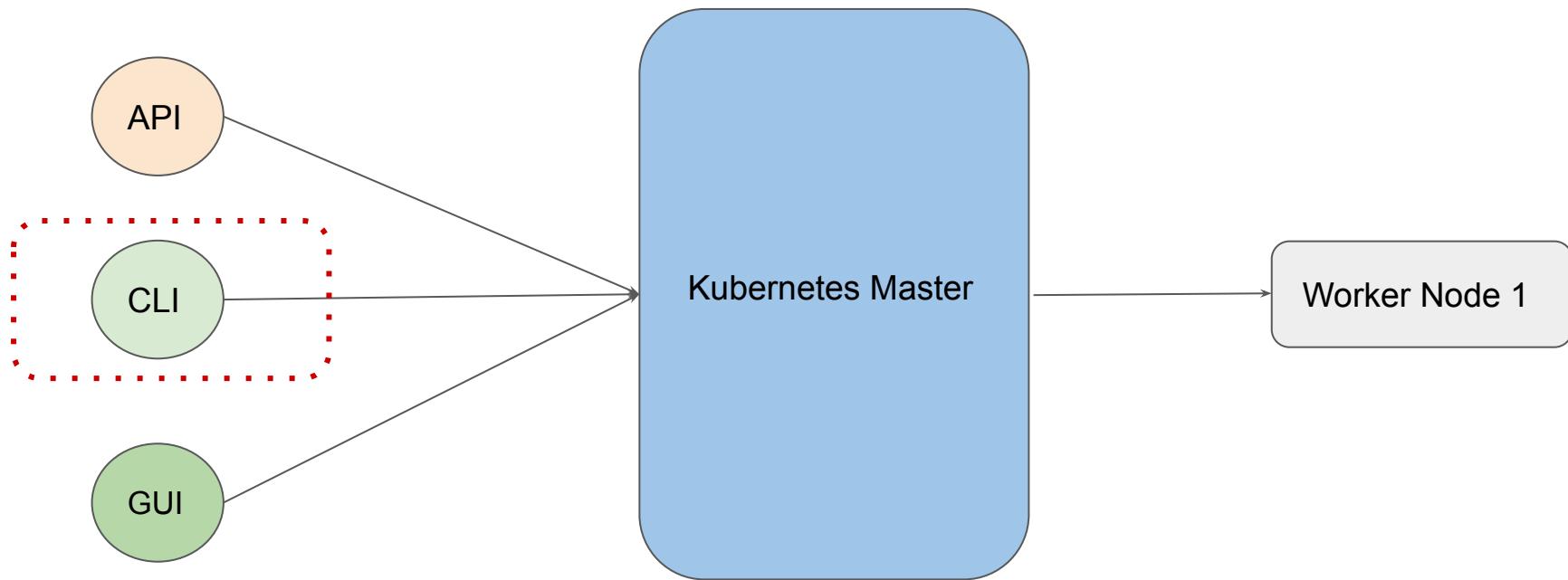
Components to be Configured - Hard Way



Components to be Configured - Managed Service



Components to be Configured - Minikube



PODS

Let's get started

The first thing to do!

The initial thing we love to do in any orchestrator is to run our first container.

Docker Command:

```
docker run --name mywebserver nginx
```

Kubectl command:

```
kubectl run mywebserver --image=nginx
```

Exec into Containers

The initial thing we love to do in any orchestrator is to run our first container.

Docker Command:

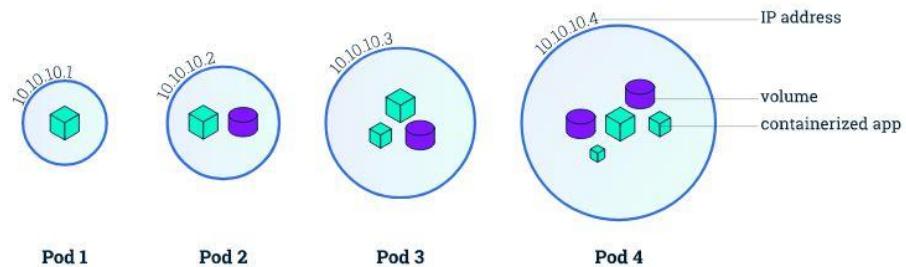
```
docker exec -it mywebserver bash
```

Kubectl command:

```
kubectl exec -it mywebserver -- bash
```

Kubernetes POD

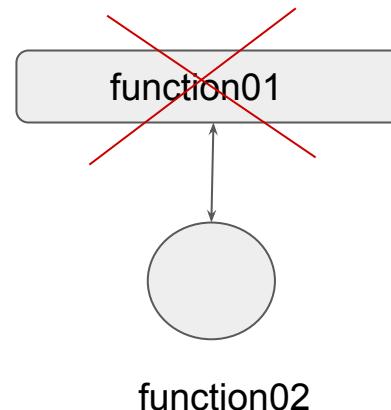
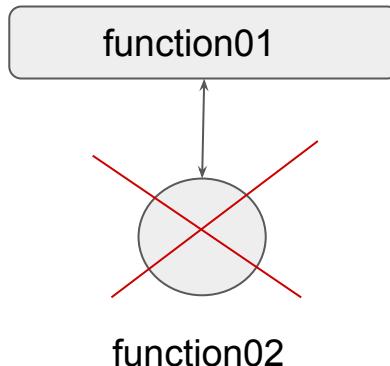
A Pod in Kubernetes represents a group of one or more application containers , and some shared resources for those containers.



Benefits of Pods

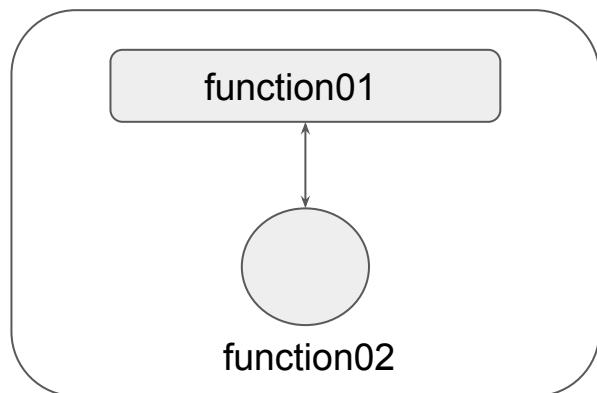
Many application might have more then one container which are tightly coupled and in one-to-one relationship.

- docker run -dt --name myweb01 function01
- docker run -dt --name myapp01 function02

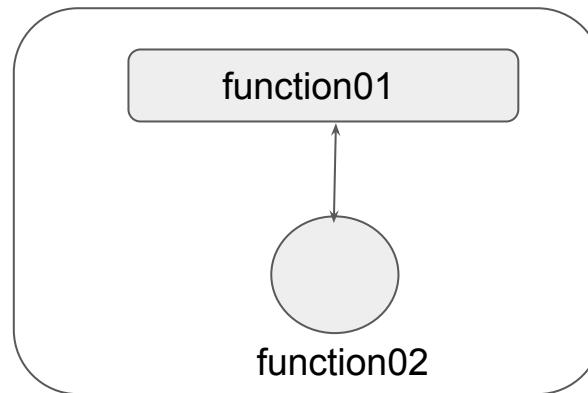


Dealing with Tightly Coupled Application

Containers within a Pod share an IP address and port space, and can find each other via localhost.



POD 1



POD 2

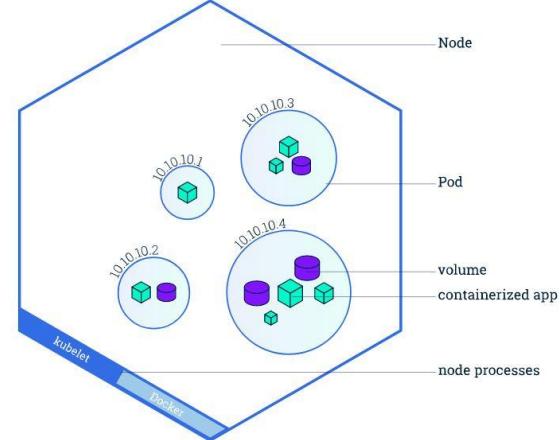
Kubernetes POD

A Pod always runs on a Node.

A Node is a worker machine in Kubernetes.

Each Node is managed by the Master.

A Node can have multiple pods.

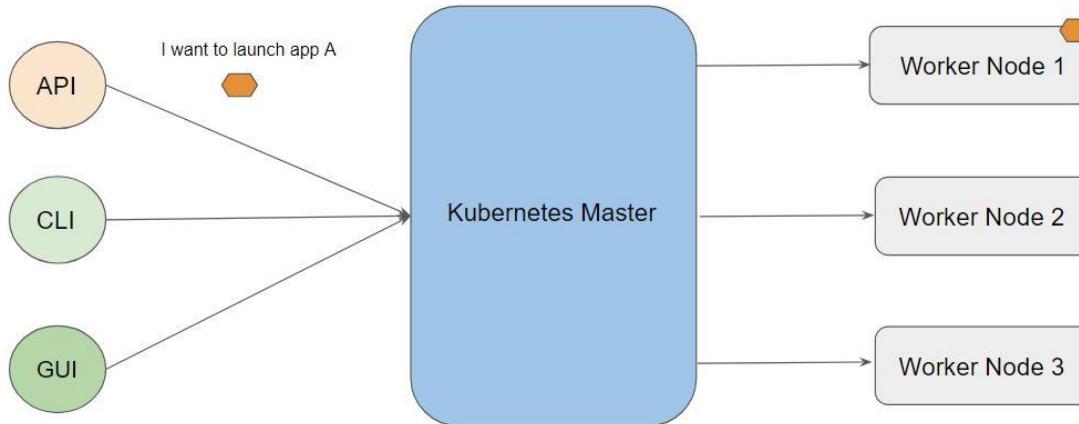


Kubernetes Objects

Let's get started

Overview of Kubernetes Objects

Kubernetes Objects is basically a record of intent that you pass on to the Kubernetes cluster. Once you create the object, the Kubernetes system will constantly work to ensure that object exists.



Approach to Configure an Object

There are various ways in which we can configure an Kubernetes Object.

- First approach is through the kubectl commands.
- Second approach is through configuration file written in YAML.

```
pod.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: mywebserver
5  spec:
6    containers:
7      - name: mywebserver
8        image: nginx
```

Importance of YAML

YAML is a human-readable data-serialization language.

It designed to be human friendly and works perfectly with other programming languages.

XML	JSON	YAML
<pre><Servers> <Server> <name>Server1</name> <owner>John</owner> <created>123456</created> <status>active</status> </Server> </Servers></pre>	<pre>{ Servers: [{ name: Server1, owner: John, created: 123456, status: active }] }</pre>	<pre>Servers: - name: Server1 owner: John created: 123456 status: active</pre>

Benefits of Configuration File

Integrates well with change review processes.

Provides the source of record on what is live within the Kubernetes cluster.

Easier to troubleshoot changes with version control.

POD Configuration in YAML

Let's get started

Approach to Configure an Object

There are various ways in which we can configure an Kubernetes Object.

- First approach is through the kubectl commands.
- Second approach is through configuration file written in YAML.

```
pod.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: mywebserver
5  spec:
6    containers:
7      - name: mywebserver
8        image: nginx
```

Understanding Important fields

Key	Description
apiVersion	Version of API
kind	Kind of object you want to create.
metadata name	Name of object that uniquely identifies it.
spec	Desired state of the object.



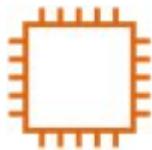
```
pod.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: mywebserver
5  spec:
6    containers:
7      - name: mywebserver
8        image: nginx
```

Labels and Selectors

Let's get started

Overview of Labels

Labels are key/value pairs that are attached to objects, such as pods.



Server



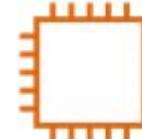
Database



Load Balancer



Load Balancer

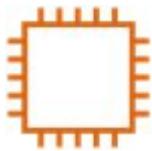


Server



Database

Adding Labels to Resource



name: kplabs-gateway
env: production



name: kplabs-db
env: production



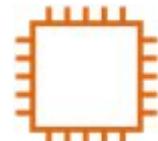
name: kplabs-elb
env: production



name: kp-elb
env: dev



name: kp-db
env: dev



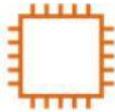
name: kp-gateway
env: dev

Overview of Selectors

Selectors allows us to filter objects based on labels.

Example:

1. Show me all the objects which has label where **env: prod**



name: kplabs-gateway
env: production



name: kplabs-db
env: production



name: kplabs-elb
env: production

Overview of Selectors

Selectors allows us to filter objects based on labels.

Example:

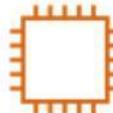
2. Show me all the objects which has label where **env: dev**



name: kp-elb
env: dev



name: kp-db
env: dev



name: kp-gateway
env: dev

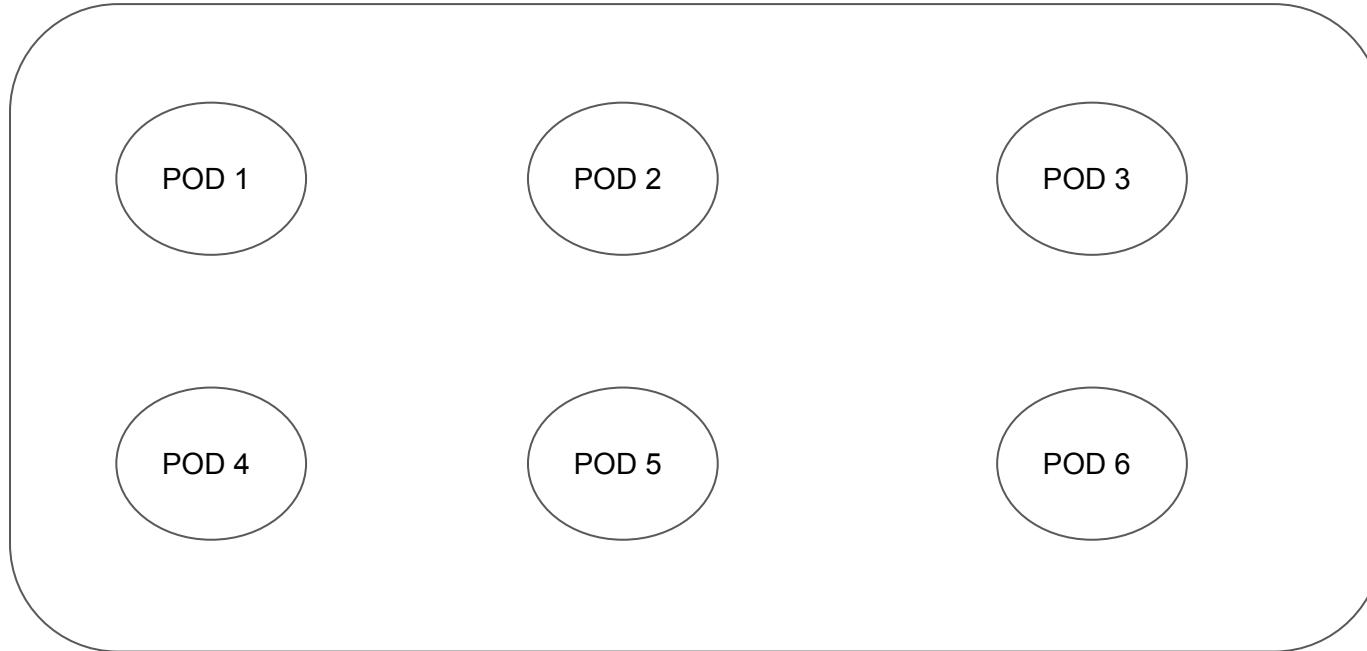
Kubernetes Perspective

There can be multiple objects within a Kubernetes cluster.

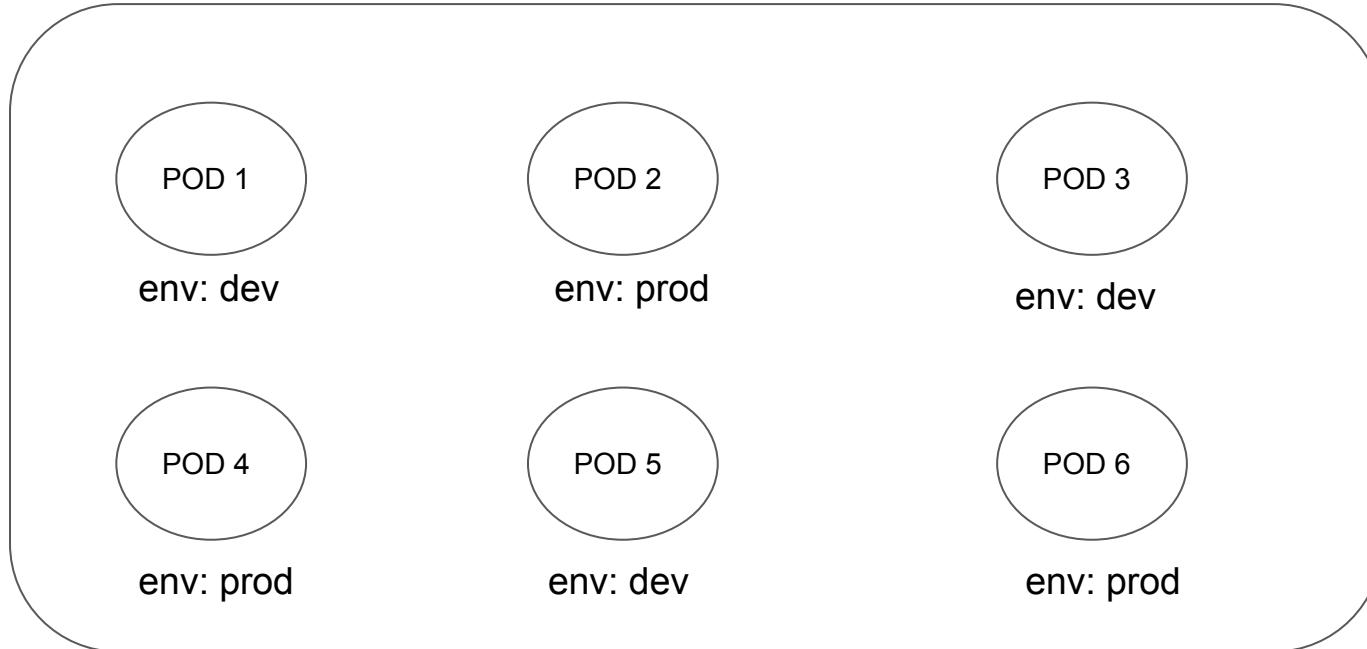
Some of the objects are as follows:

1. Pods
2. Services
3. Secrets
4. Namespaces
5. Deployments
6. Daemonsets

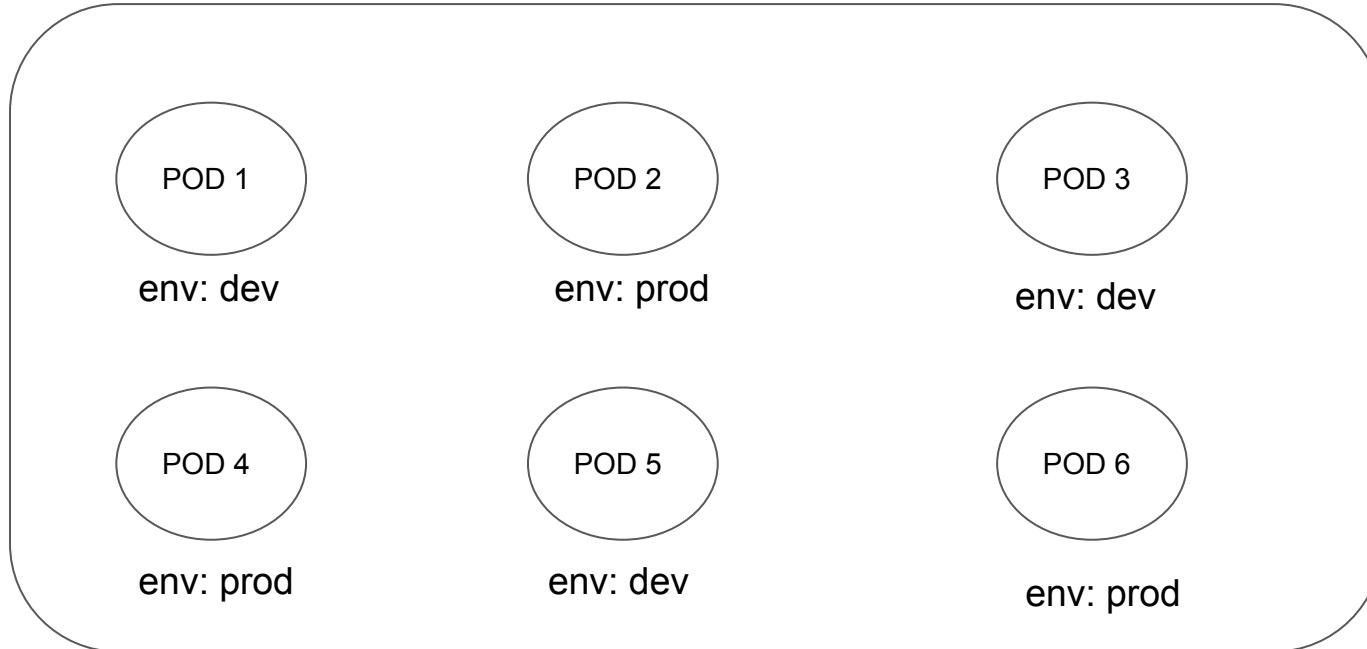
Kubernetes Perspective



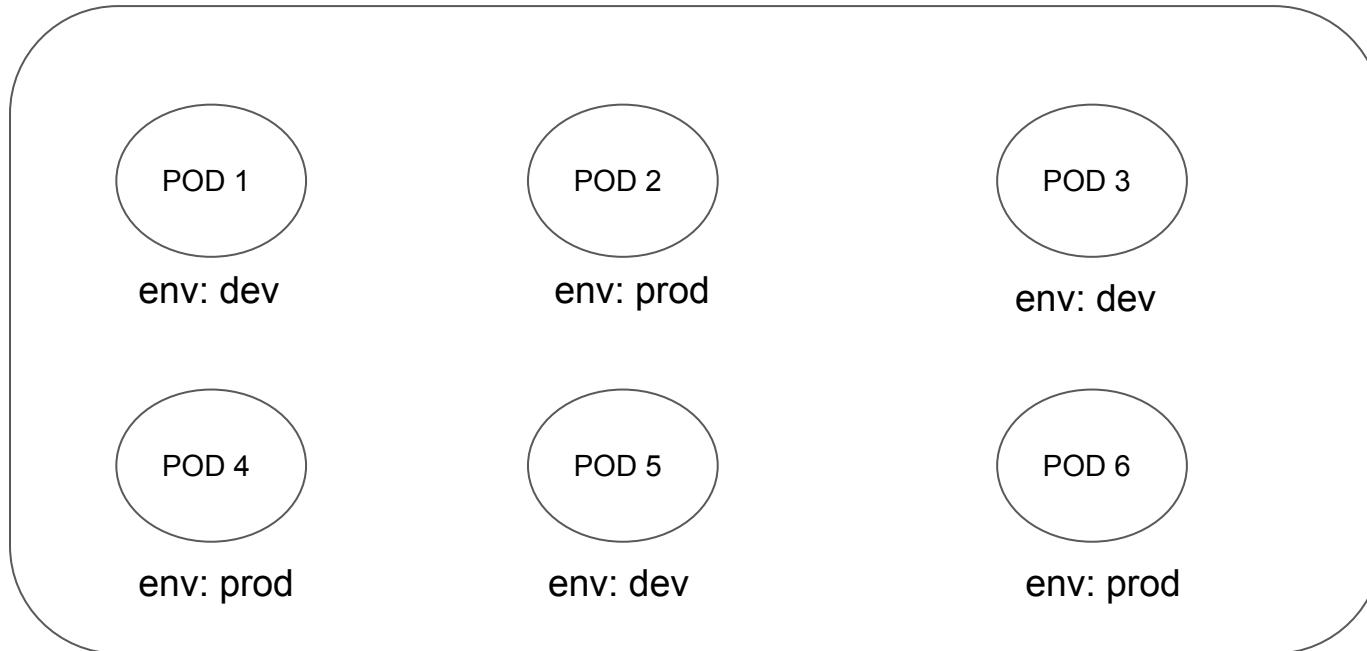
Assigning Labels to Kubernetes Objects



Selector: List All Pods from Dev Environment



Selector: List All Pods from Production Environment

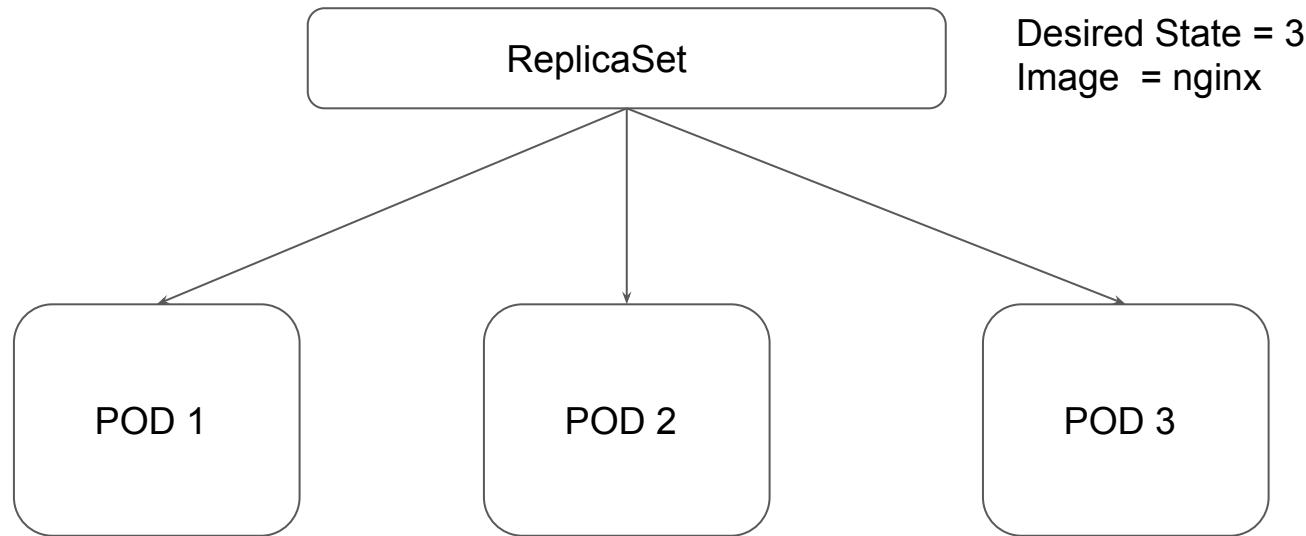


ReplicaSets

Let's get started

Overview of Kubernetes ReplicaSets

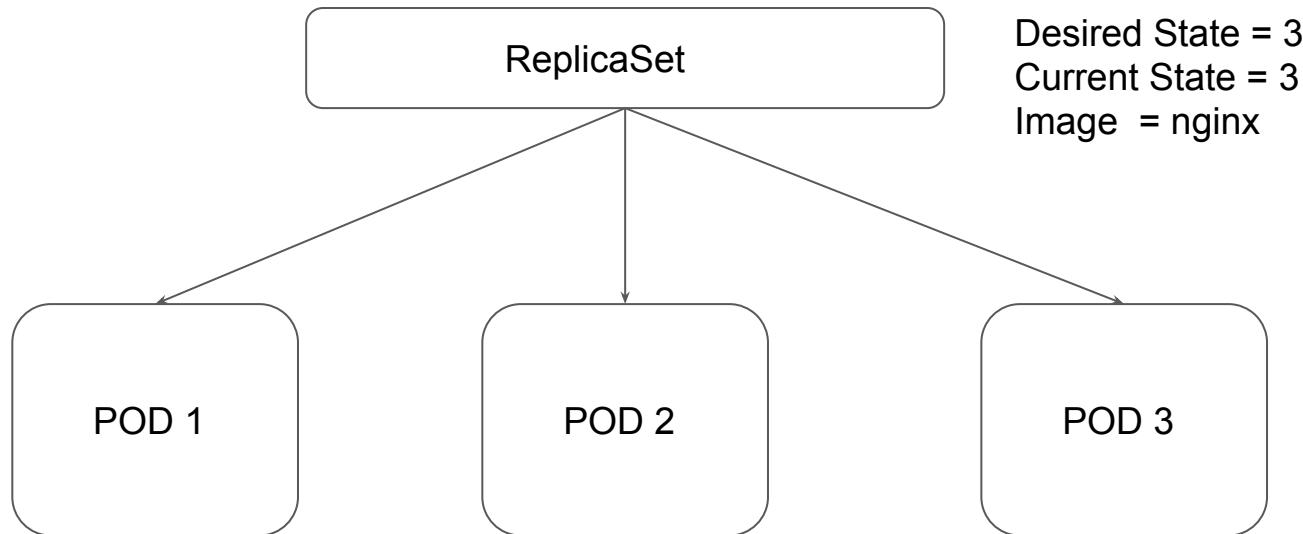
A ReplicaSet purpose is to maintain a stable set of replica Pods running at any given time.



Desired State vs Current State

Desired State - The state of pods which is desired.

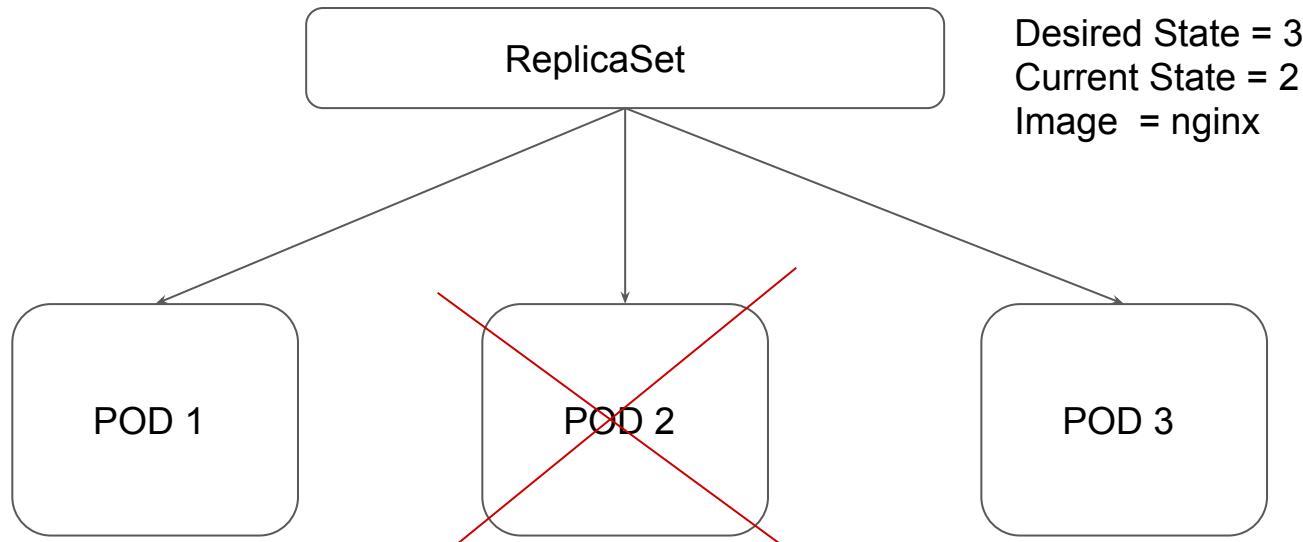
Current State - The actual state of pods which are running.



Desired State vs Current State

Desired State - The state of pods which is desired.

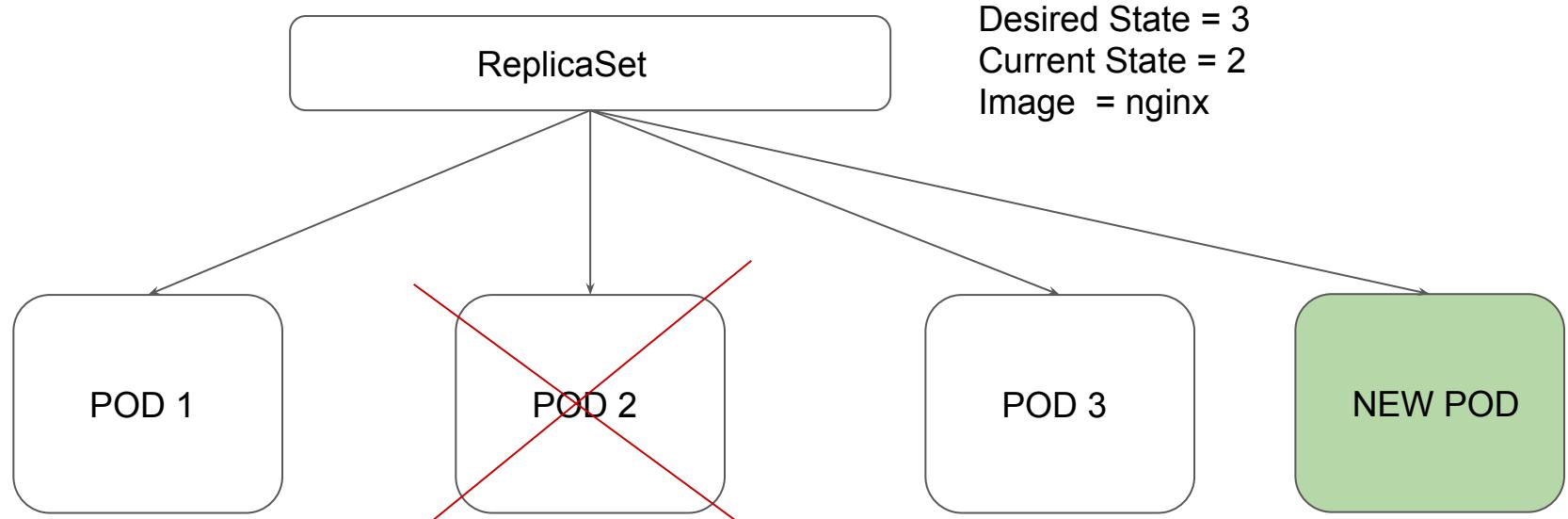
Current State - The actual state of pods which are running.



Desired State vs Current State

Desired State - The state of pods which is desired.

Current State - The actual state of pods which are running.

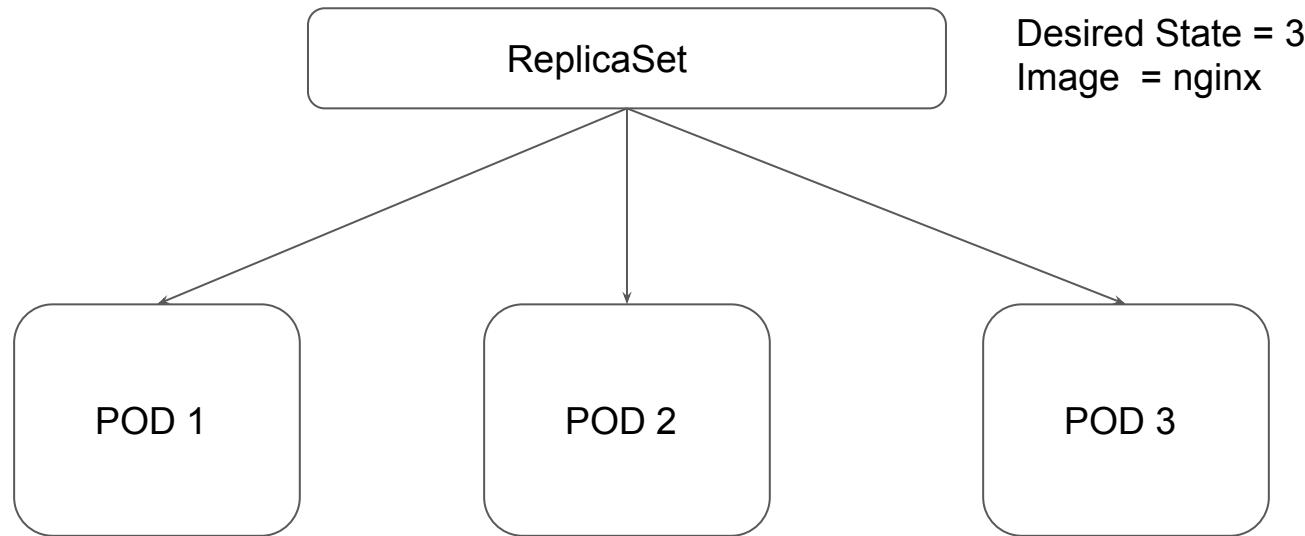


Deployments

Let's get started

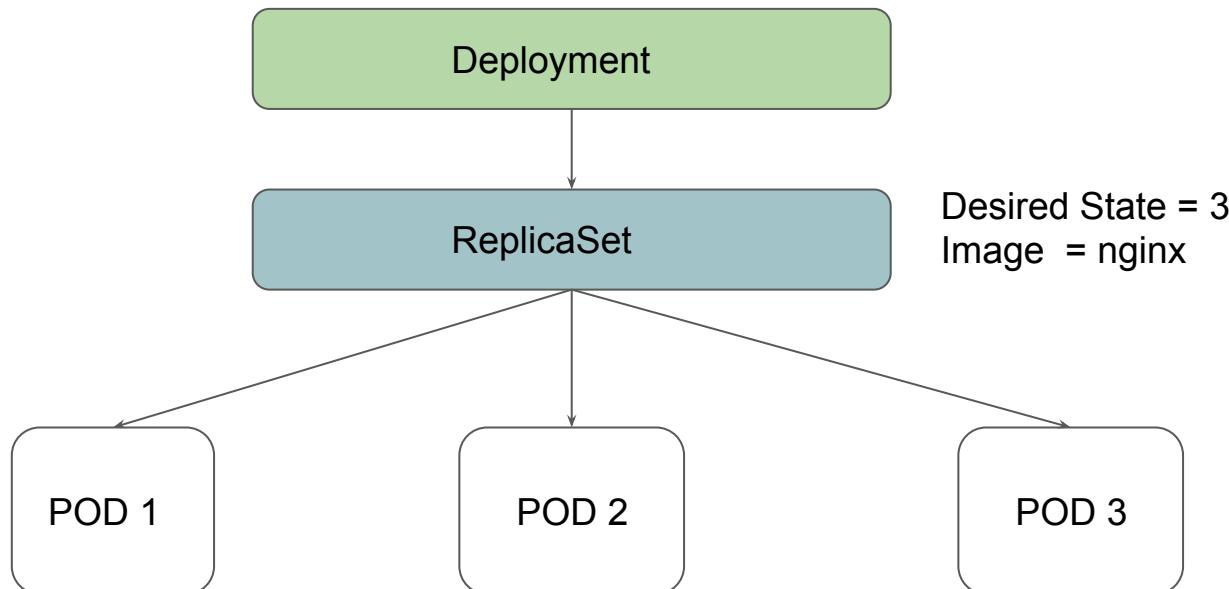
Challenges with ReplicaSets

ReplicaSets works well in basic functionality like managing pods, scaling pods and similar.



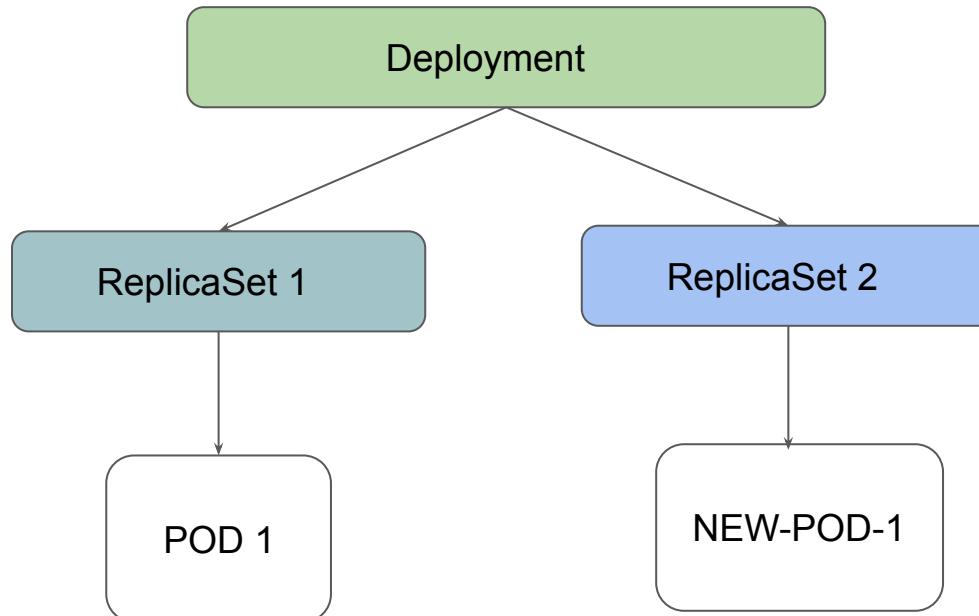
Understanding Deployments

Deployments provides replication functionality with the help of ReplicaSets, along with various additional capability like rolling out of changes, rollback changes if required.



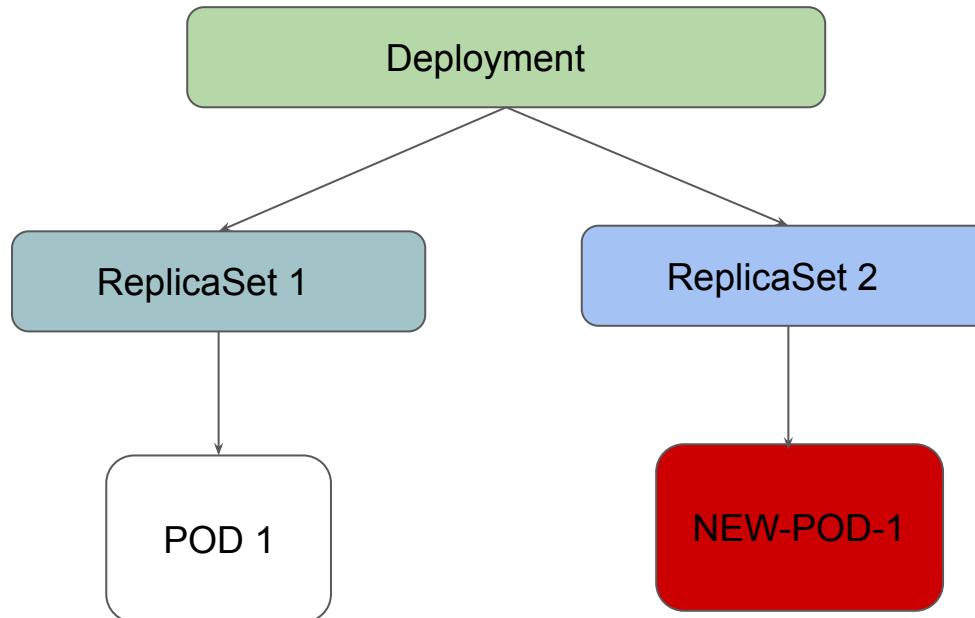
Benefits of Deployments - Rollout Changes

- We can easily rollout new updates to our application using deployments.
- Deployments will perform update in rollout manner to ensure that your app is not down.



Benefits of Deployments - RollBack Changes

- Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping

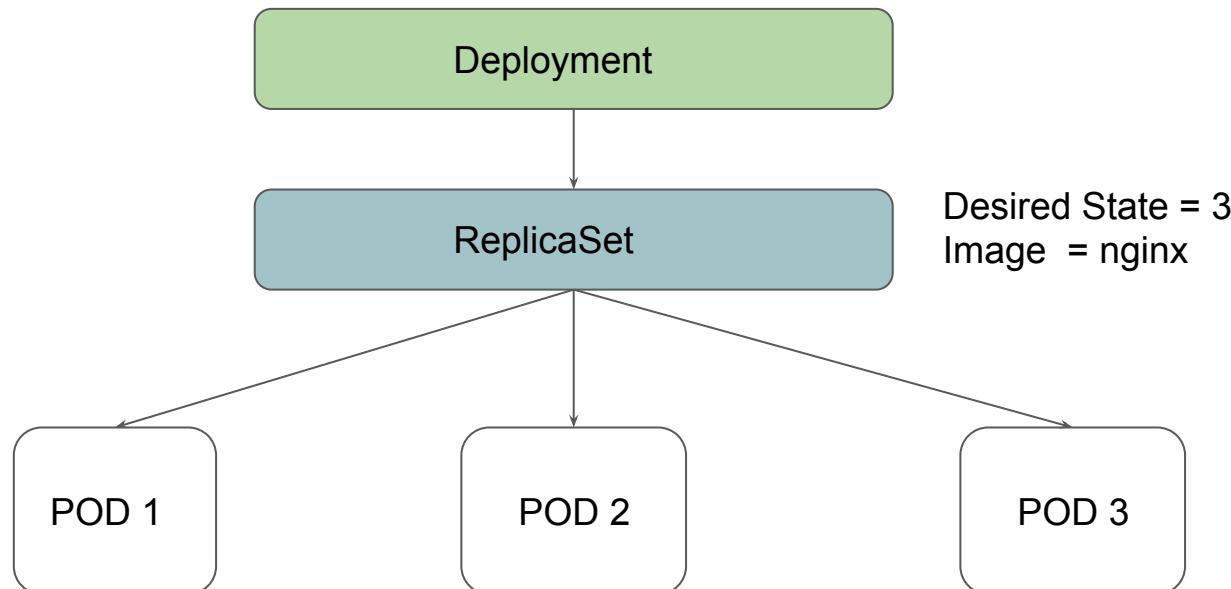


Creating First Deployment

Let's get started

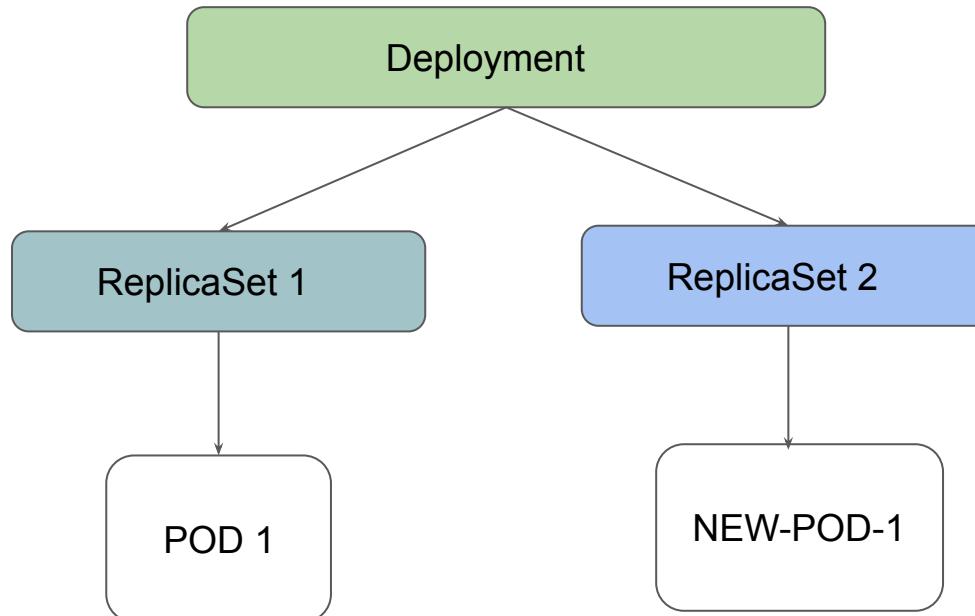
Understanding Deployments

Deployments provides replication functionality with the help of ReplicaSets, along with various additional capability like rolling out of changes, rollback changes if required.



Benefits of Deployments - Rollout Changes

- We can easily rollout new updates to our application using deployments.
- Deployments will perform update in rollout manner to ensure that your app is not down.



Important Pointers

Deployment ensures that only a certain number of Pods are down while they are being updated.

By default, it ensures that at least 25% of the desired number of Pods are up (25% max unavailable).

Deployments keep the history of revision which had been made.

maxSurge and maxUnavailable

Deployment Related Options

Overview of Deployment Configuration

While performing a rolling update, there are two important configuration to know.

Configuration Parameter	Description
maxSurge	Maximum Number of PODS that can be scheduled above original number of pods.
maxUnavailable	Maximum number of pods that can be unavailable during the update

Example Use-Case

Total PODS in Deployment 8.

maxSurge	maxUnavailable
25%	25%

maxSurge = 2 PODS

maxUnavailable = 2 PODS

At Most of 10 PODS (8 current pods + 2 maxSurge pods)

At Least of 6 PODS (6 current pods - 2 maxUnavailable pods)

Note

You can define the maxSurge and maxUnavailable in both the numeric and percentage terms;

maxUnavailable: 0

maxSurge: 10%

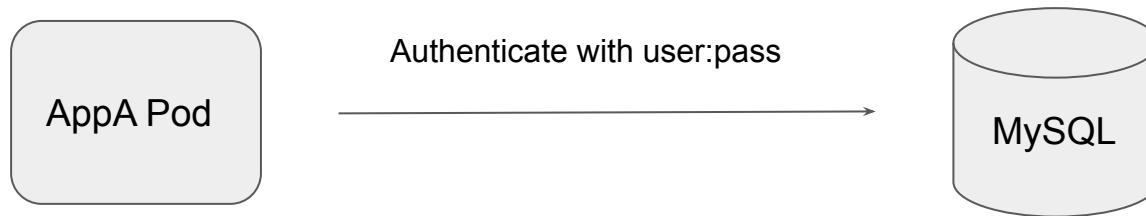
Kubernetes Secrets

Security Aspect

Let's understand the challenge

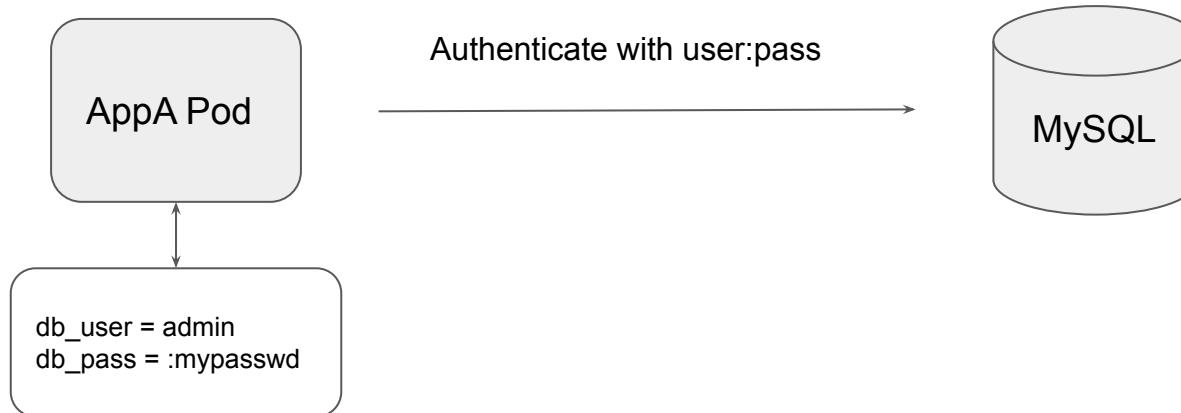
Use-Case:

AppA runs on a pod and it needs to connect to a MySQL DB to start working properly.



Common Approach: Hardcoding Credentials

Many a times you will find that developer have hardcoded credentials within their container image.



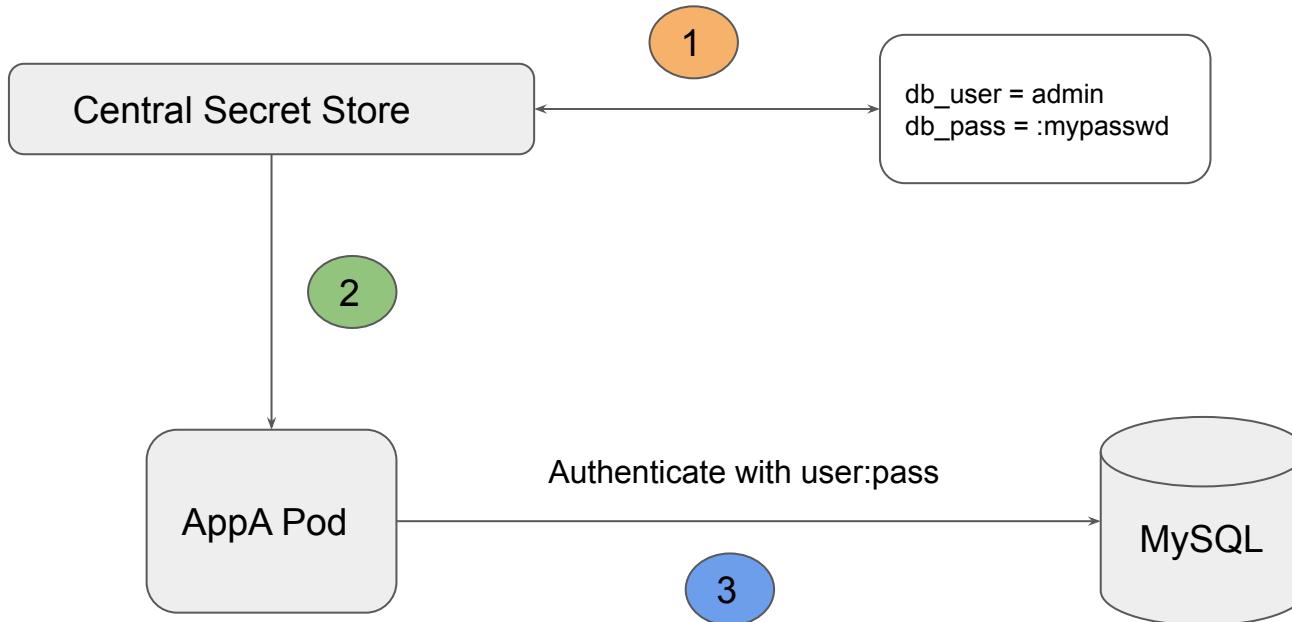
Common Approach: Risks

There are multiple risks of hard coding credentials:

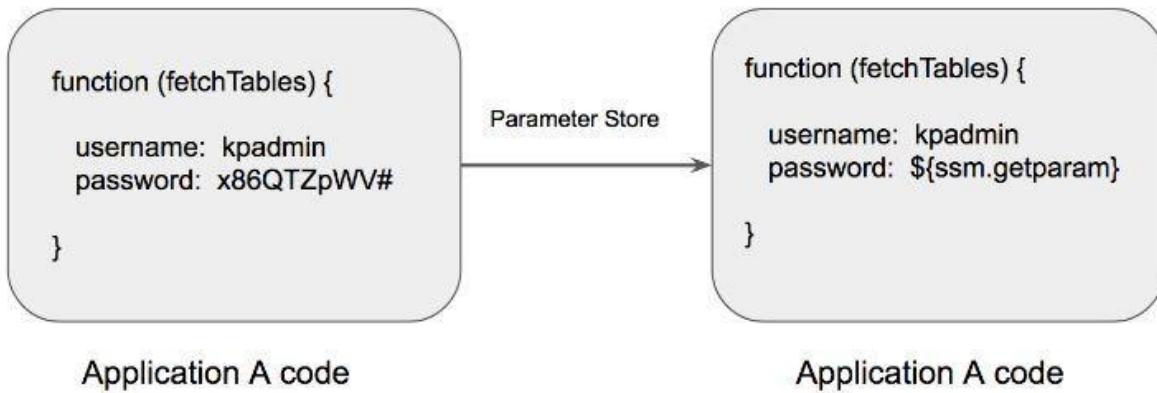
1. Anyone having access to the container repository can easily fetch the credentials.
2. Developer needs to have credentials of production systems.
3. Update of credentials will lead to new docker image being built.



Storing Credentials Centrally



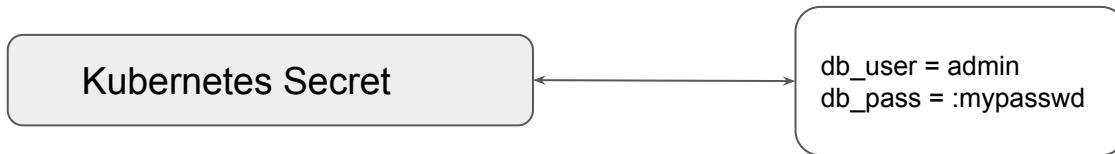
AppA Code Needs to be Modified



Overview of Kubernetes Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.

- Allows customers to store secrets centrally to reduce risk of exposure.
- Stored in ETCD database.



CLI Syntax for Creating Secret

```
kubectl create secret [TYPE] [NAME] [DATA]
```

Elaborating Type:

i) Generic:

- File (--from-file)
- directory
- literal value

ii) Docker Registry

iii) TLS

Mounting Secrets in Containers

Security Aspect

Mounting Secrets in Containers

Once a secret is created, it is necessary to make it available to containers in a pod.

There are two approaches to achieve this:

1. Volumes
2. Environment Variables.

ConfigMaps

Storing Configurations Centrally

Sample Use-Case

We have an AppA container and depending on the environment, its settings needs to be differed.

Example Properties:

Dev Environment: app.env=dev app.mem=2048m app.properties=dev.env.url

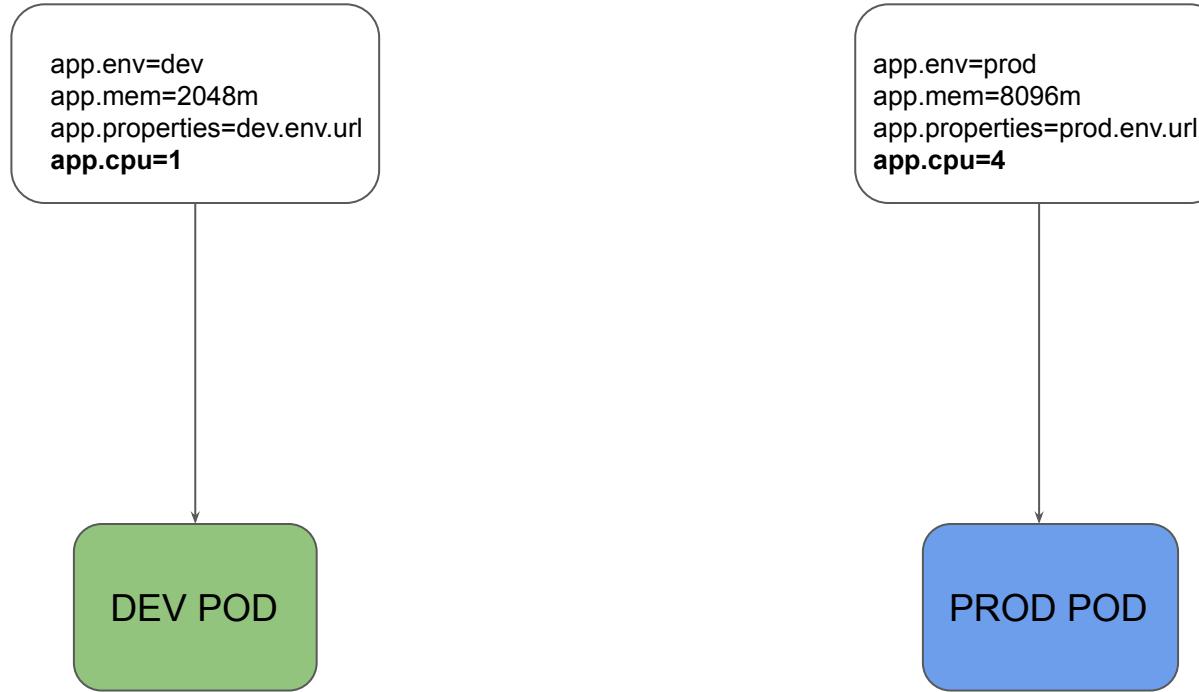
Prod Environment: app.env=prod app.mem=8096m app.properties=prod.env.url



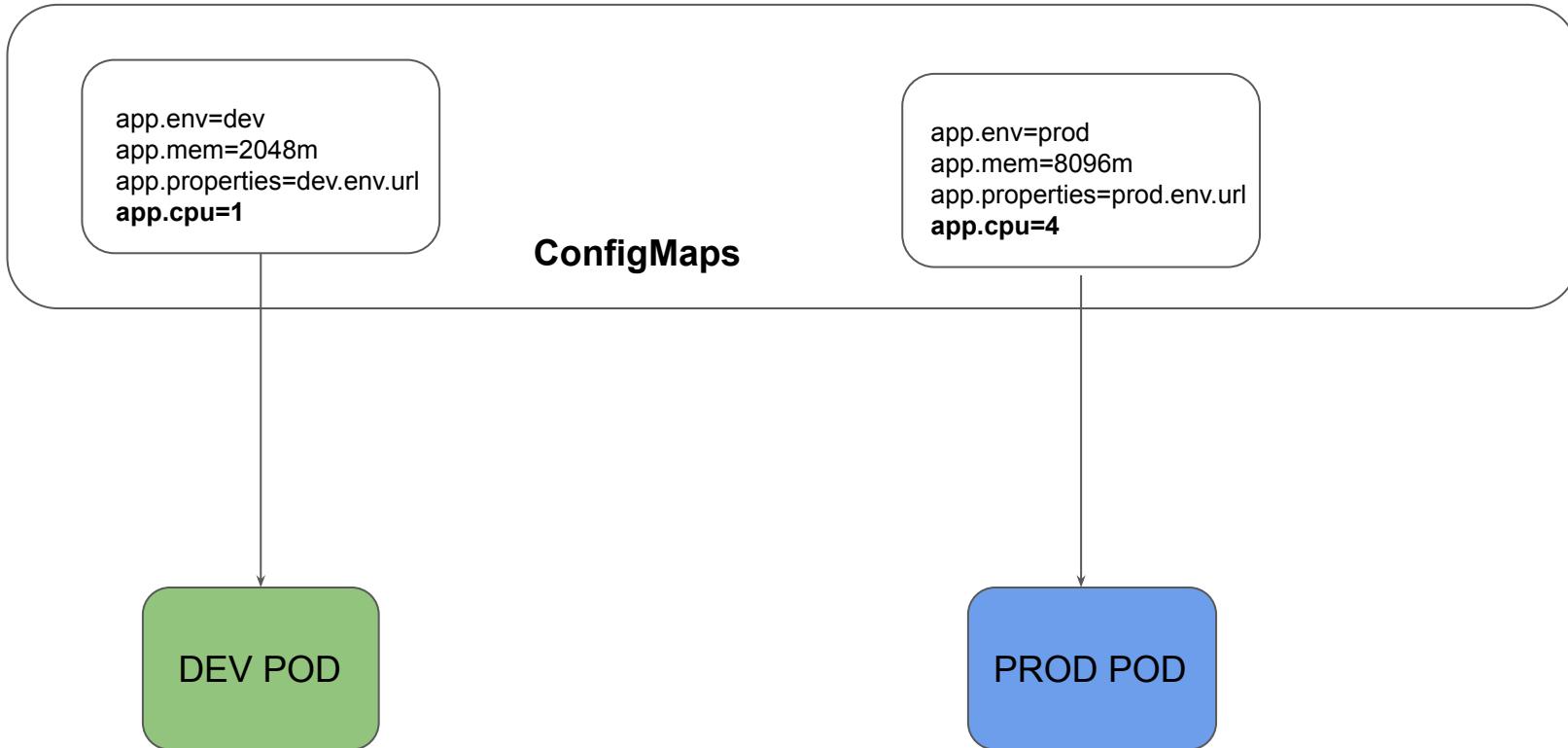
Centrally Store Data



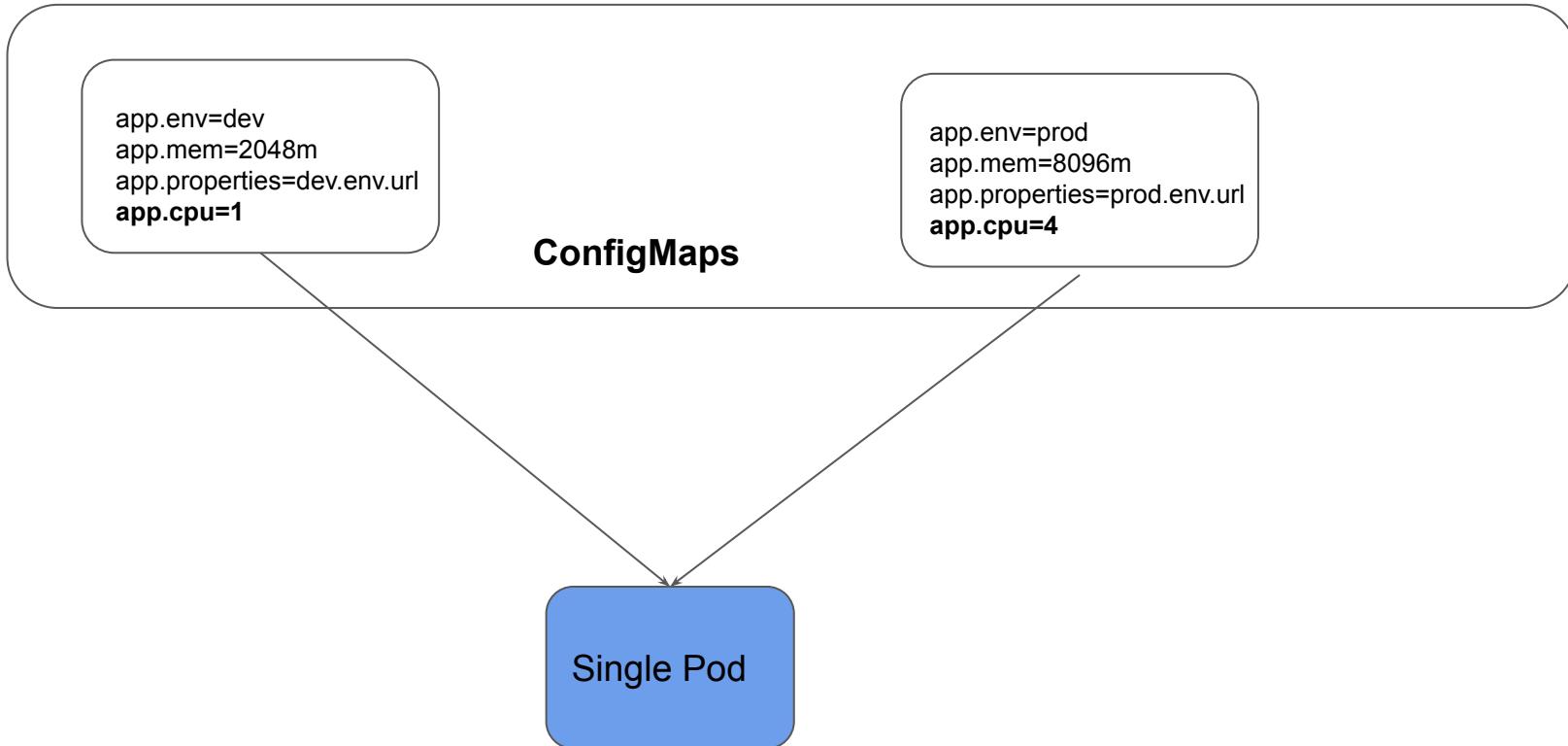
Dynamically Change the Values



ConfigMaps



ConfigMaps



CLI Syntax for Creating ConfigMap

```
kubectl create configmap [NAME] [DATA-SOURCE]
```

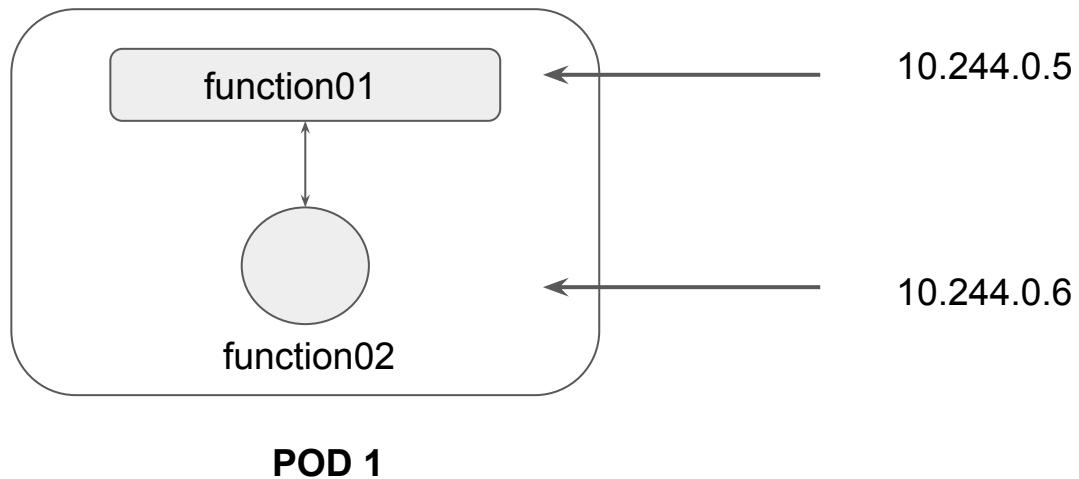
- File
- directory
- literal value

Overview of Service

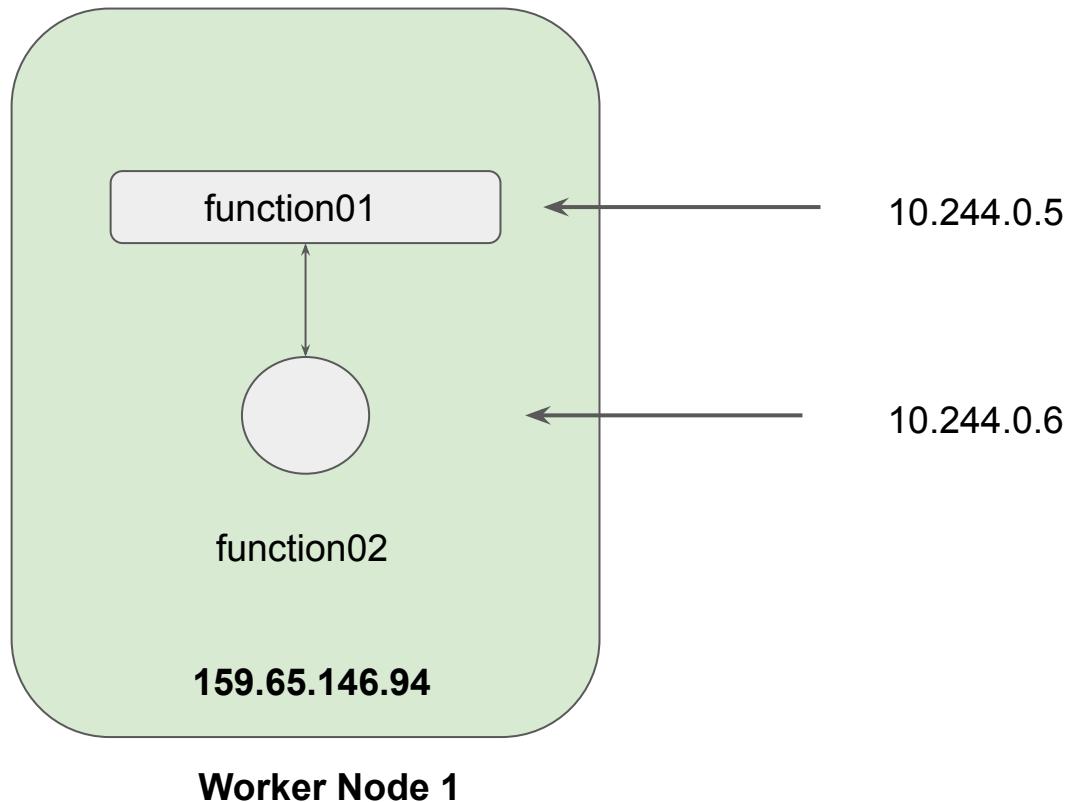
Let's get started

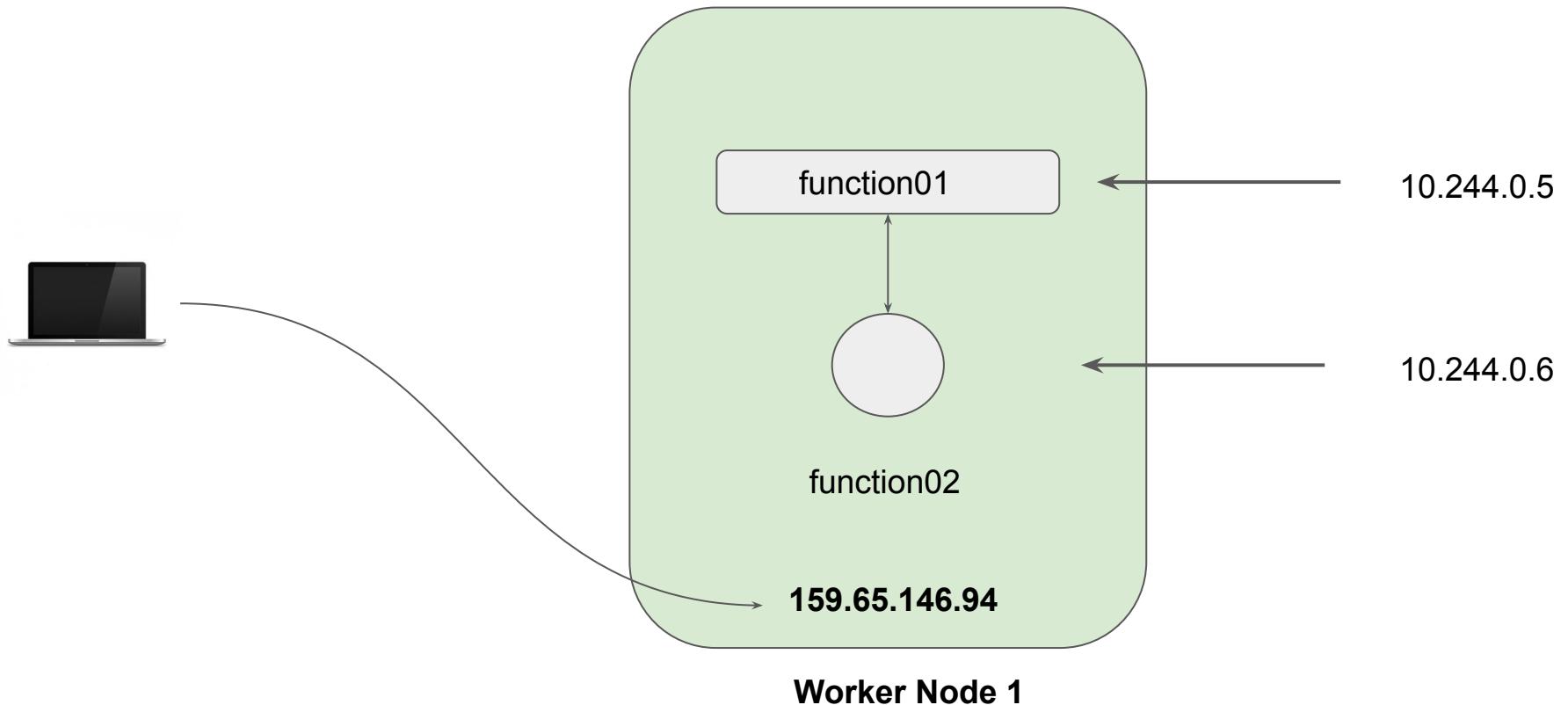
Overview of Kubernetes Pods

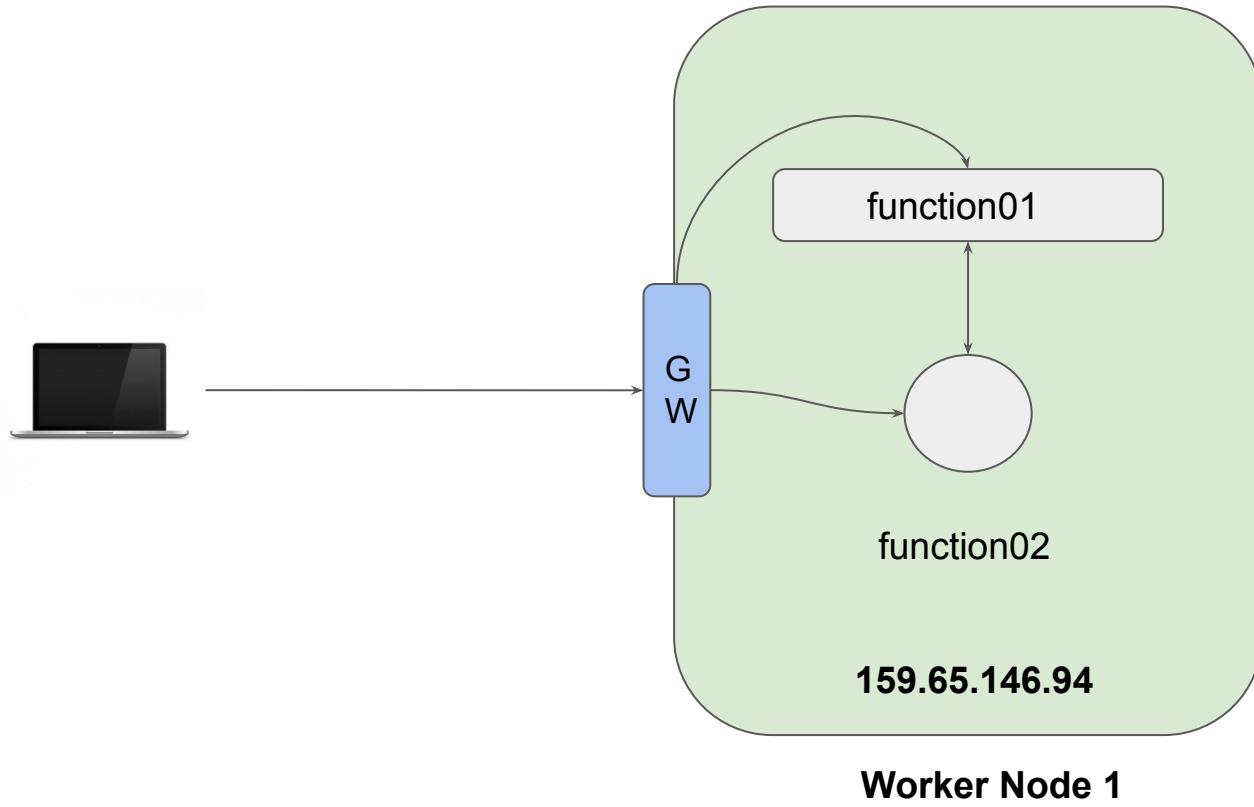
Whenever you create a Pod, the containers created will have Private IP addresses.

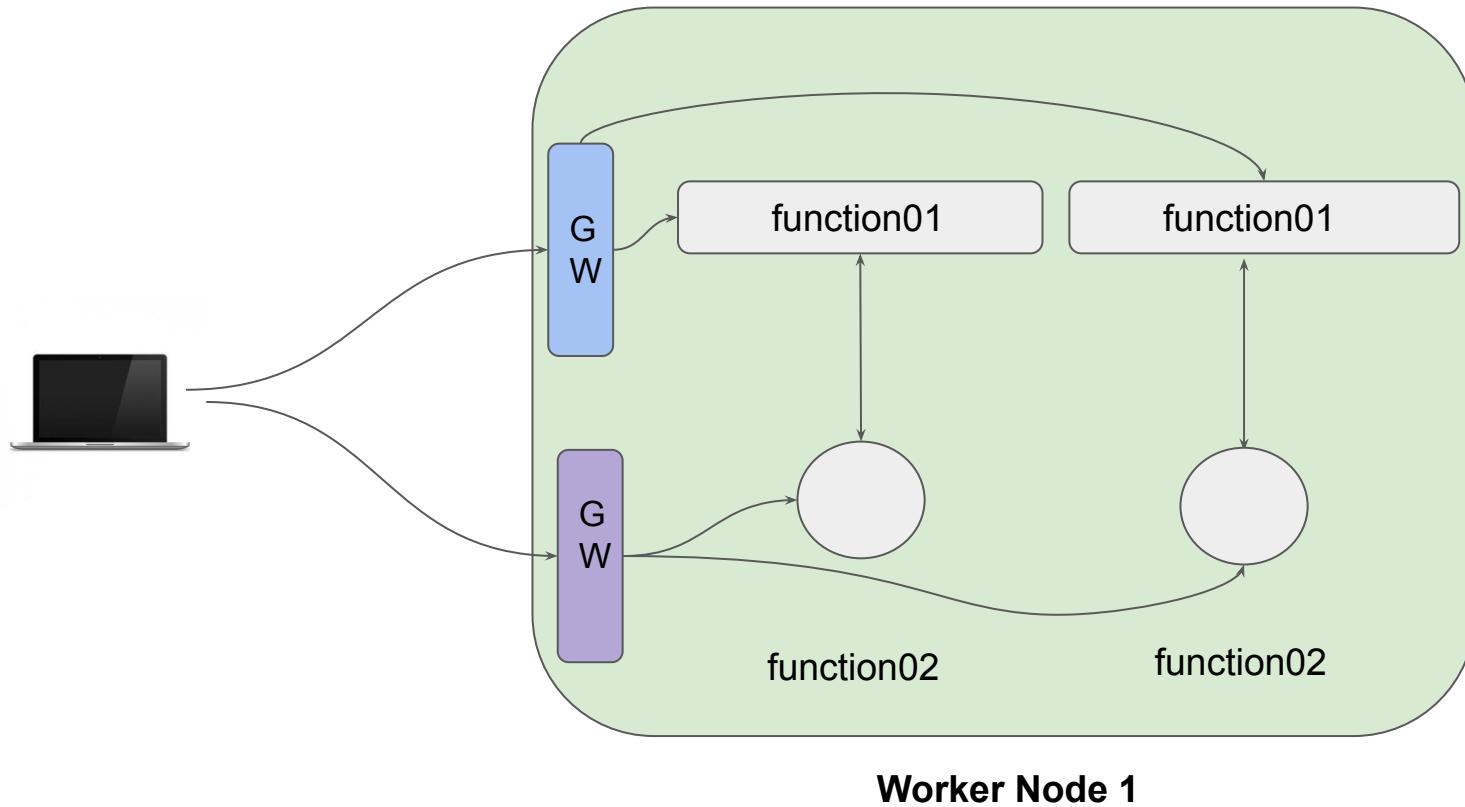


Bigger Network Picture









Worker Node 1

Importance of Service in Kubernetes

In a Kubernetes cluster, each Pod has an internal IP address.

Pods are generally ephemeral, they can come and go anytime.

We can make use of service which acts as a gateway and can get us connected with right set of pods.

Types of Kubernetes Service

Service is an abstract way of exposing application running in the pods as a network service.

There are several types of Kubernetes Services which are available:

- NodePort
- ClusterIP
- LoadBalancer
- ExternalName

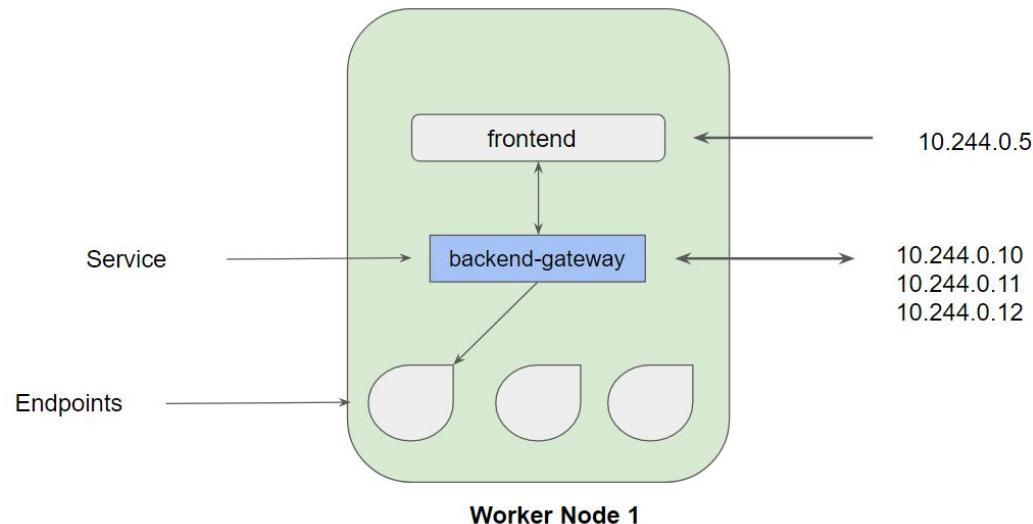
Creating Service and Endpoints

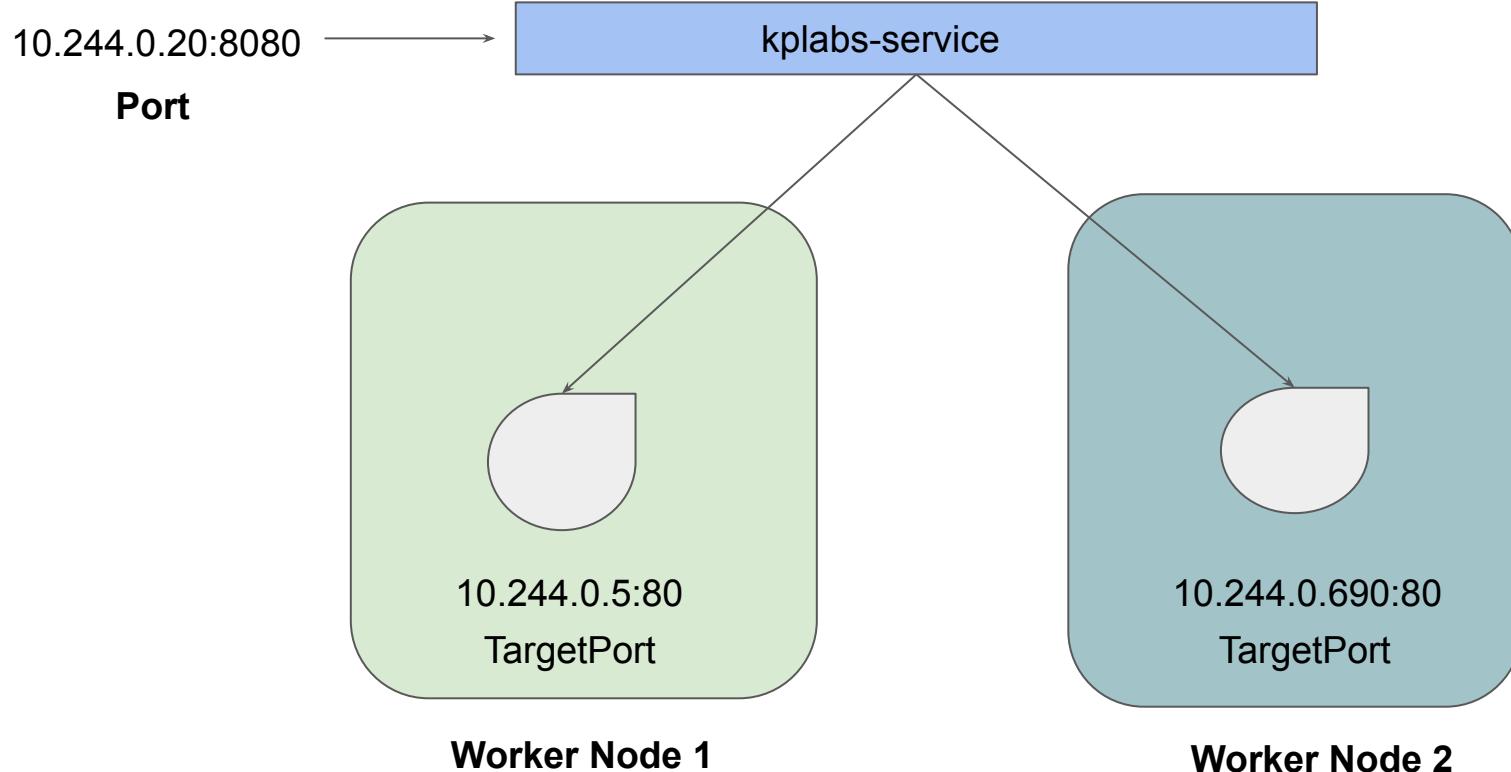
Networking Aspect

Service and Endpoints

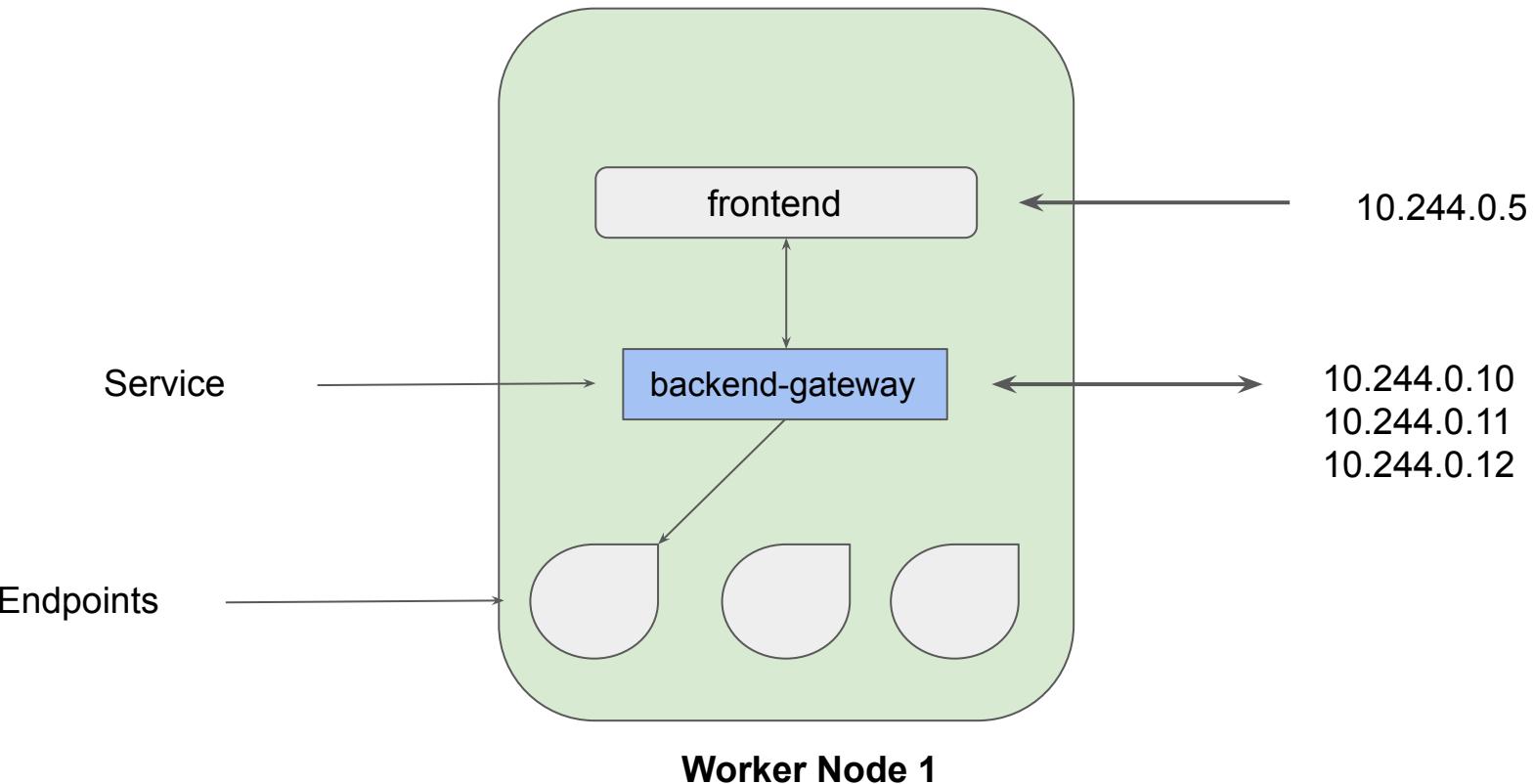
Service is a gateway that distributes the incoming traffic between its endpoints

Endpoints are the underlying PODS to which the traffic will be routed to.





Services and Endpoints



Service Type - ClusterIP

Networking Aspect

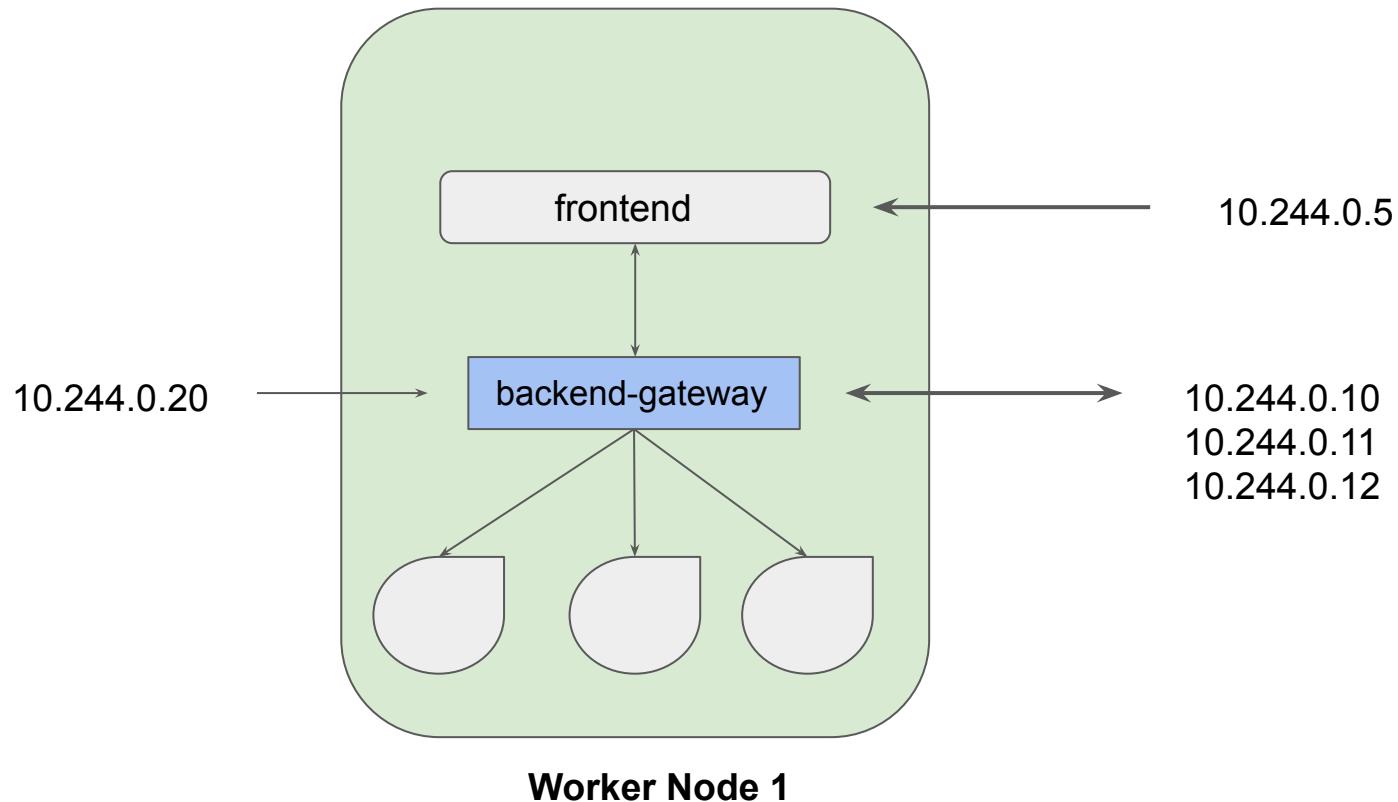
Understanding ClusterIP

Whenever service type is ClusterIP, an internal cluster IP address is assigned to the service.

Since an internal cluster IP is assigned, it can only be reachable from within the cluster.

This is a default ServiceType.

Overview of Services



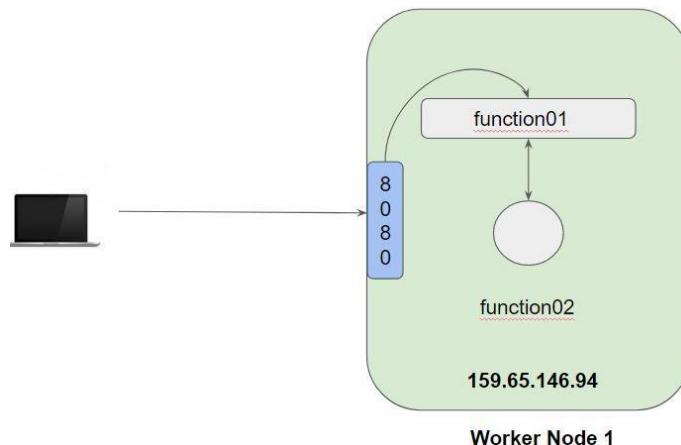
NodePort Service

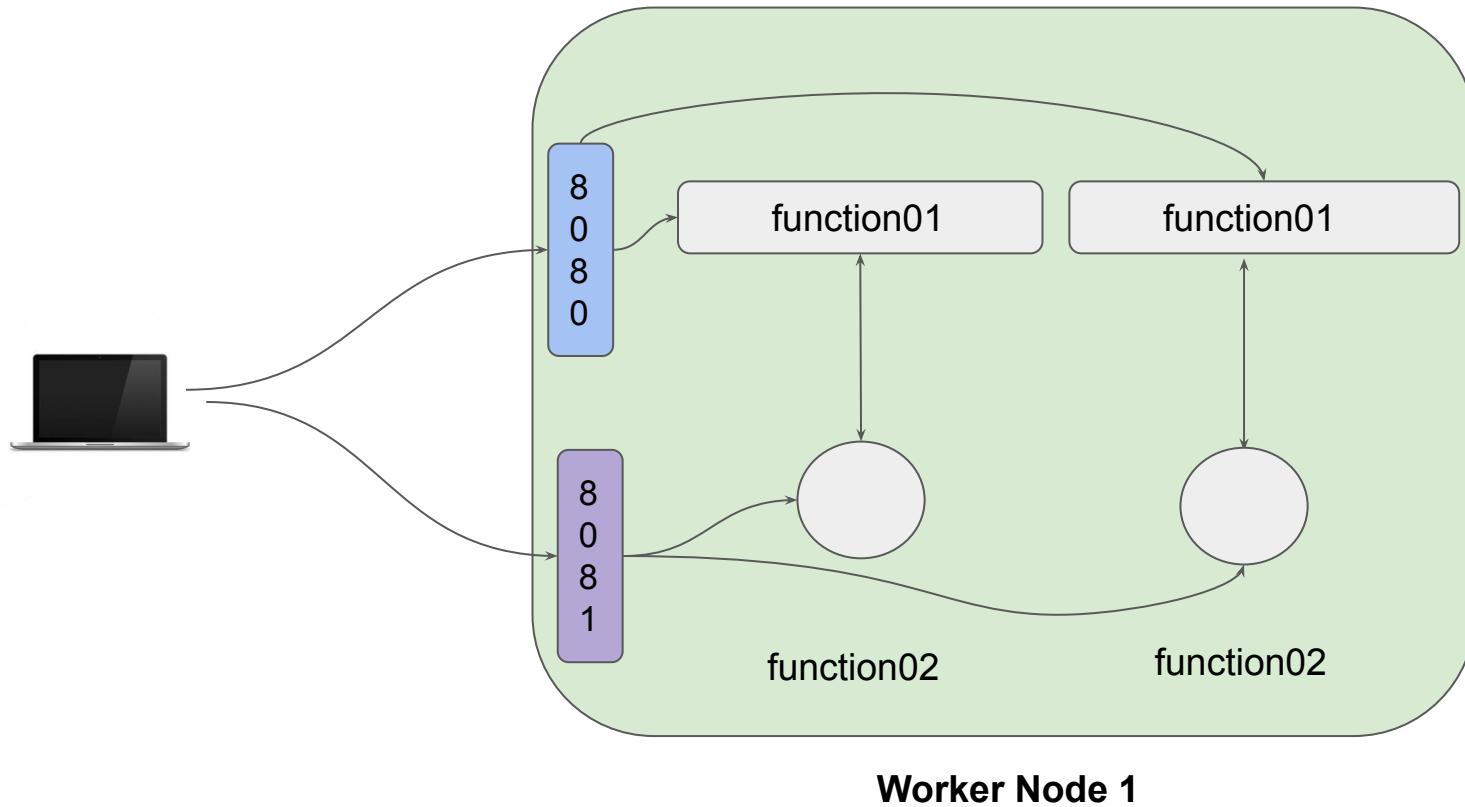
Let's get started

Overview of Kubernetes Pods

Exposes the Service on each Node's IP at a static port.

You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

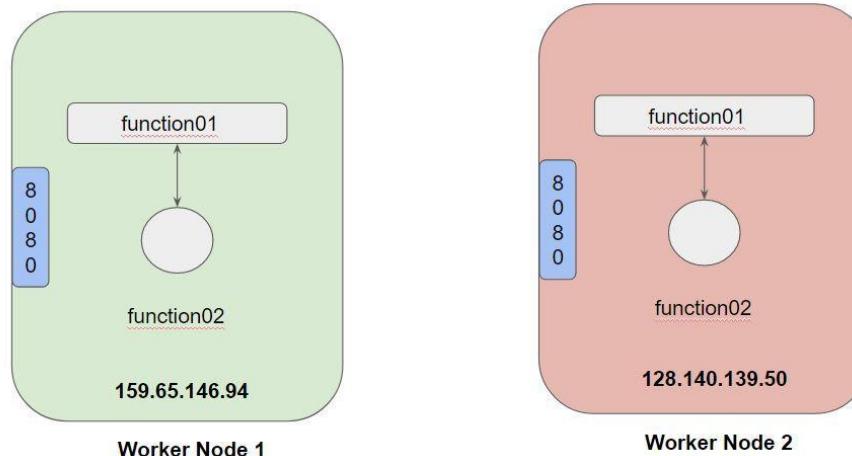




Types of Kubernetes Service

In a NodePort service type, Kubernetes will allocate a port from a range specified by --service-node-port-range flag (default: 30000-32767).

Each node proxies that port (the same port number on every Node) into your Service.



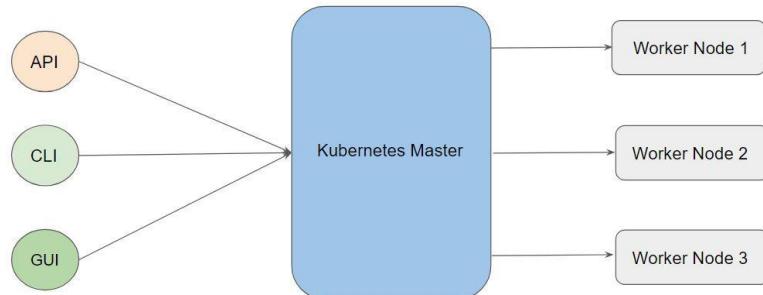
K8s Networking Model

Kubernetes In Detail

Understanding the Challenges

Kubernetes was built to run on distributed systems where there can be hundreds of worker nodes in which Pods would be running.

This makes networking a very important part component and with the understanding of Kubernetes networking model, it will allow administrators to properly run, monitor as well as troubleshoot applications in K8s clusters.



Overview of the Networking Model

Kubernetes imposes the following fundamental requirements on any networking implementation

1. pods on a node can communicate with all pods on all nodes without NAT
2. all Nodes can communicate with all Pods without NAT.
3. the IP that a Pod sees itself as is the same IP that others see it as.

Challenges & Solutions

Based on the constraints set, there are four different networking challenges that needs to be solved:

- Container-to-Container Networking
- Pod-to-Pod Networking
- Pod-to-Service Networking
- Internet-to-Service Networking

1 - Container to Container Networking

Container to Container networking primarily happens inside a pod.

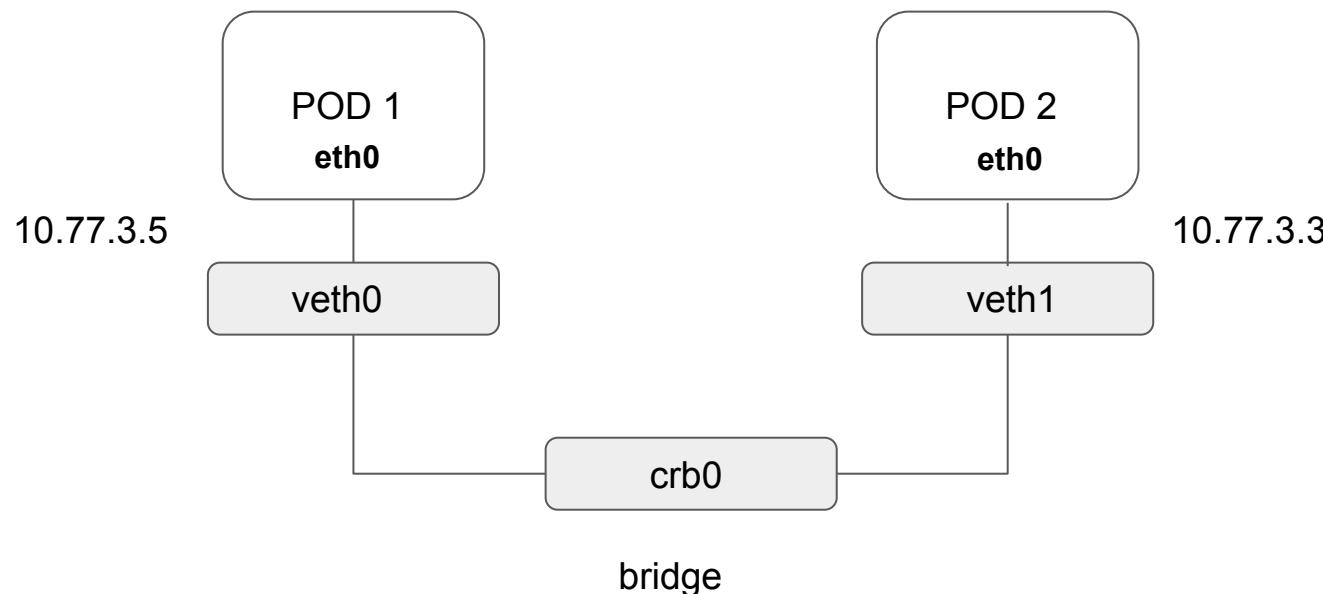
PODs can contain group of containers with the same IP address.

Communication between the containers inside pods happens via localhost



2 - Pod to Pod Networking

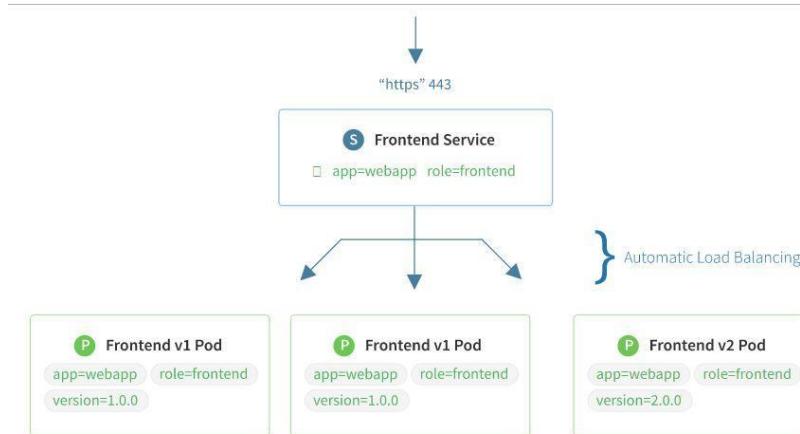
Primary aim is to understand how Pod to Pod communication works.



3 - Pod to Service Communication

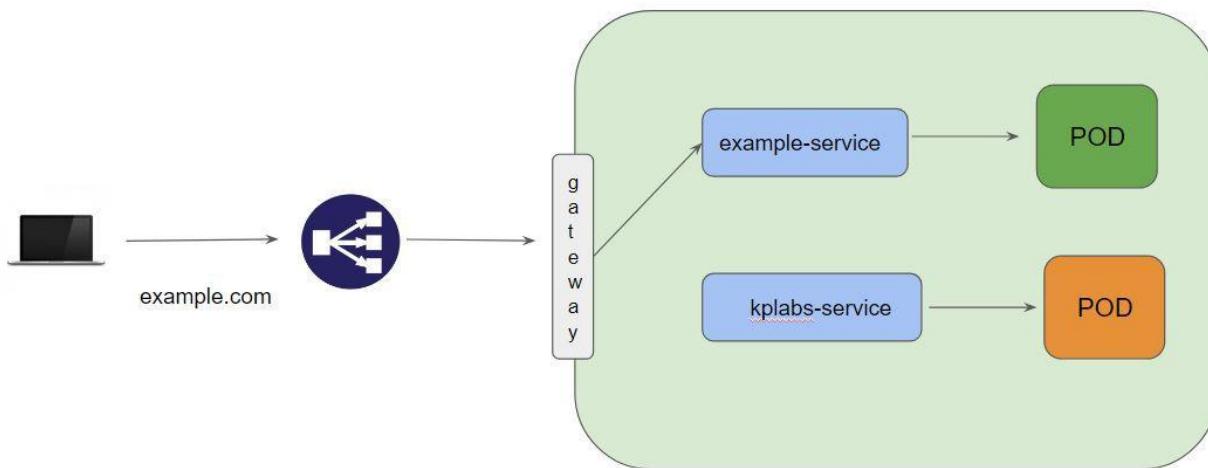
Kubernetes Service can act as an abstraction which can provide a single IP address and DNS through which pods can be accessed.

Endpoints tracks the IP address of the objects that service can send traffic to.



4 - Internet to Service Networking

Kubernetes Ingress is a collection of routing rules which governs how external users access the services running within the Kubernetes cluster.



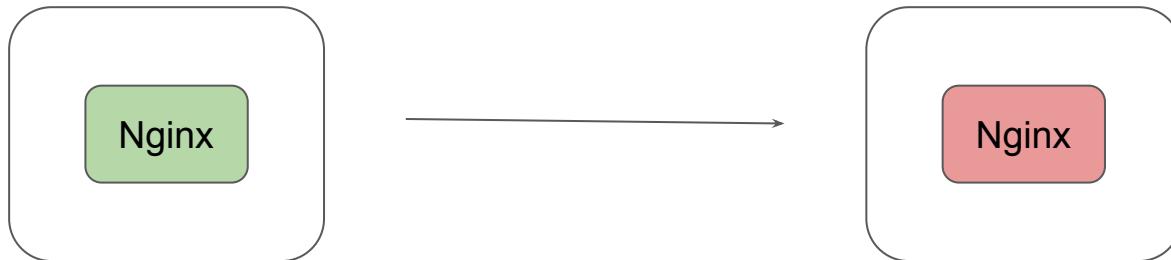
Liveness Probe

Health Checks

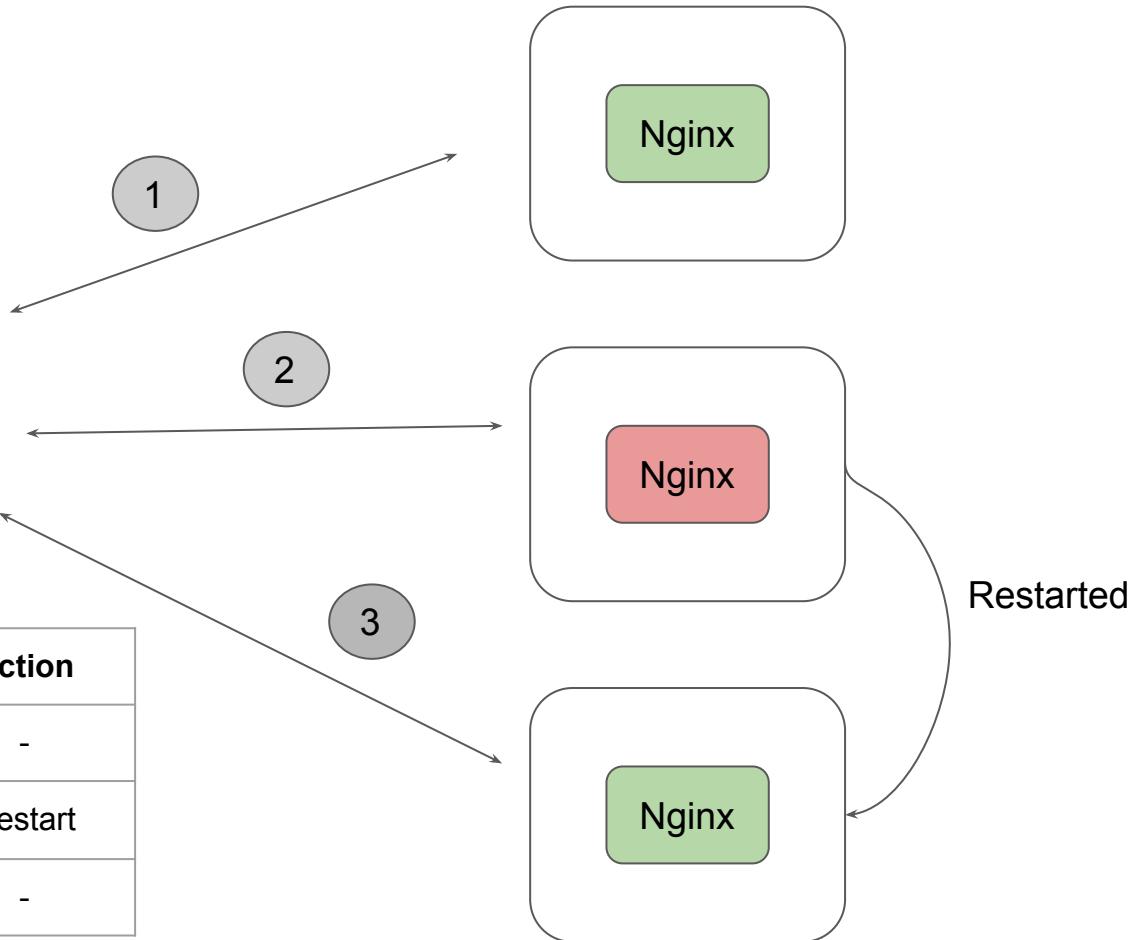
Overview of Liveness Probe

Many applications running for long periods of time eventually transition to broken states, and cannot recover except by being restarted.

Kubernetes provides liveness probes to detect and remedy such situations.



Probe	Status	Action
1	Heathy	-
2	Unhealthy	Restart
3	Heathy	-



Types of Probes

There are 3 types of probes which can be used with Liveness

- HTTP
- Command
- TCP

In this demo we had taken an example based on command.

Readiness Probe

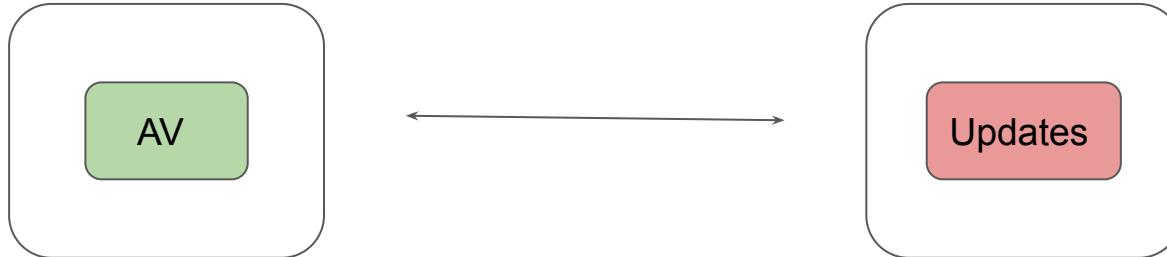
Readiness Checks

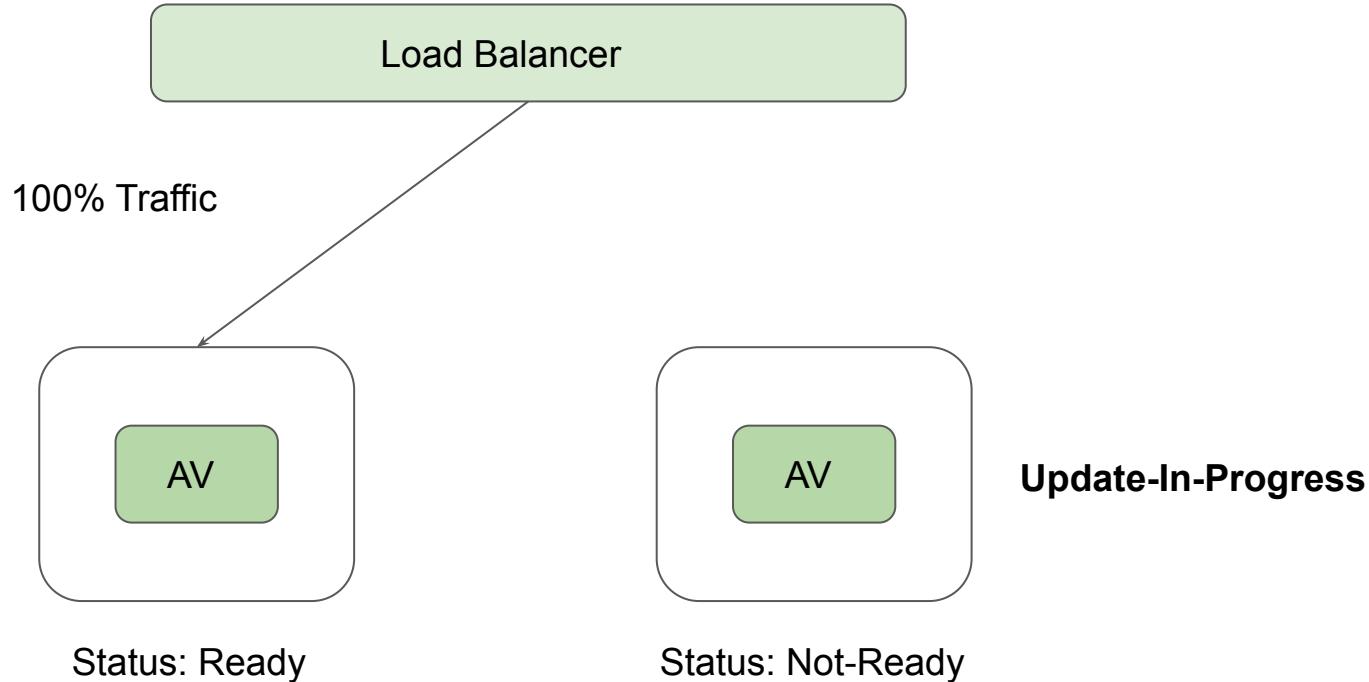
Overview of Readiness Probe

It can happen that an application is running but temporarily unavailable to serve traffic.

For example, application is running but it is still loading its large configuration files from external vendor.

In such-case, we don't want to kill the container however we also do not want it to serve the traffic.





Overview of Readiness Probe

Syntax of Readiness Probe:

readinessProbe:

exec:

command:

- cat

- /tmp/healthy

initialDelaySeconds: 5

periodSeconds: 5

DaemonSet

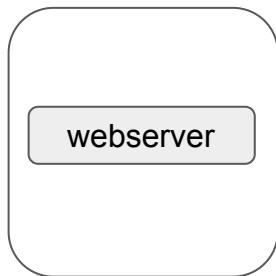
Global Service

Understanding the Use-Case

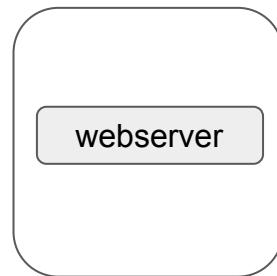
Sample: Use-Case:

Let's say that we have three nodes and we want to run a single copy of pod in every node.

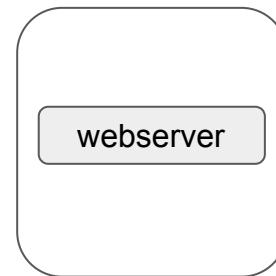
Worker Nodes



Node 01



Node 02



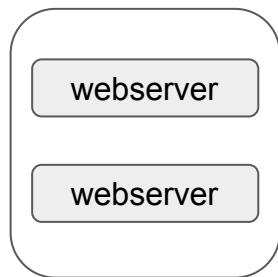
Node 03

Understanding the Use-Case

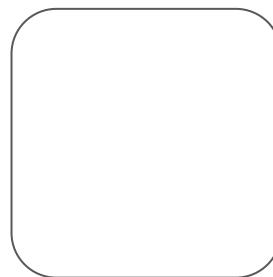
Sample: Use-Case:

Let's say that we have three nodes and we want to run a copy of pod in every node.

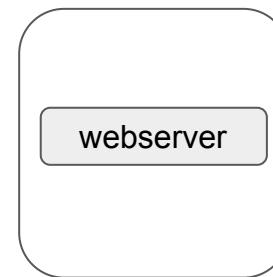
Worker Nodes



Node 01



Node 02



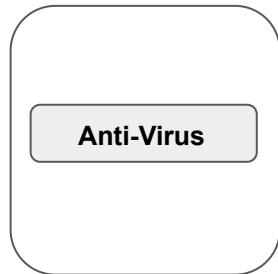
Node 03

Understanding DaemonSet

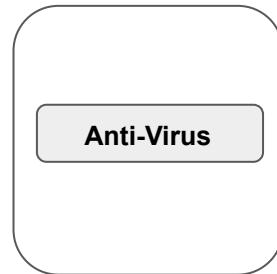
A DaemonSet can ensure that all Nodes run a copy of a Pod.

As nodes are added to the cluster, Pods are added to them.

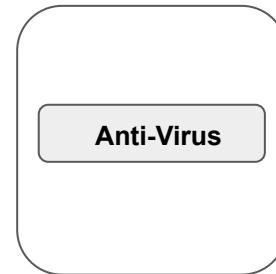
Worker Nodes



Node 01



Node 02



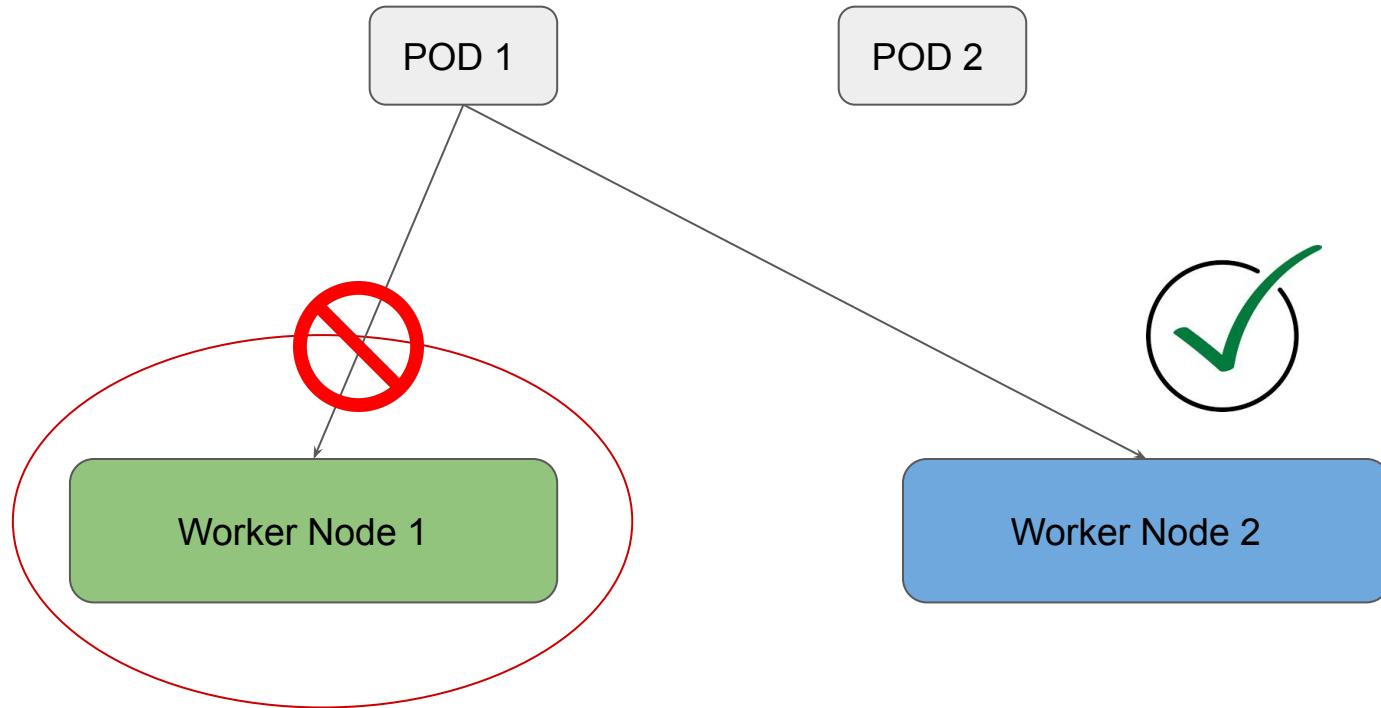
Node 03

Taints and Tolerations

Advanced Scheduling

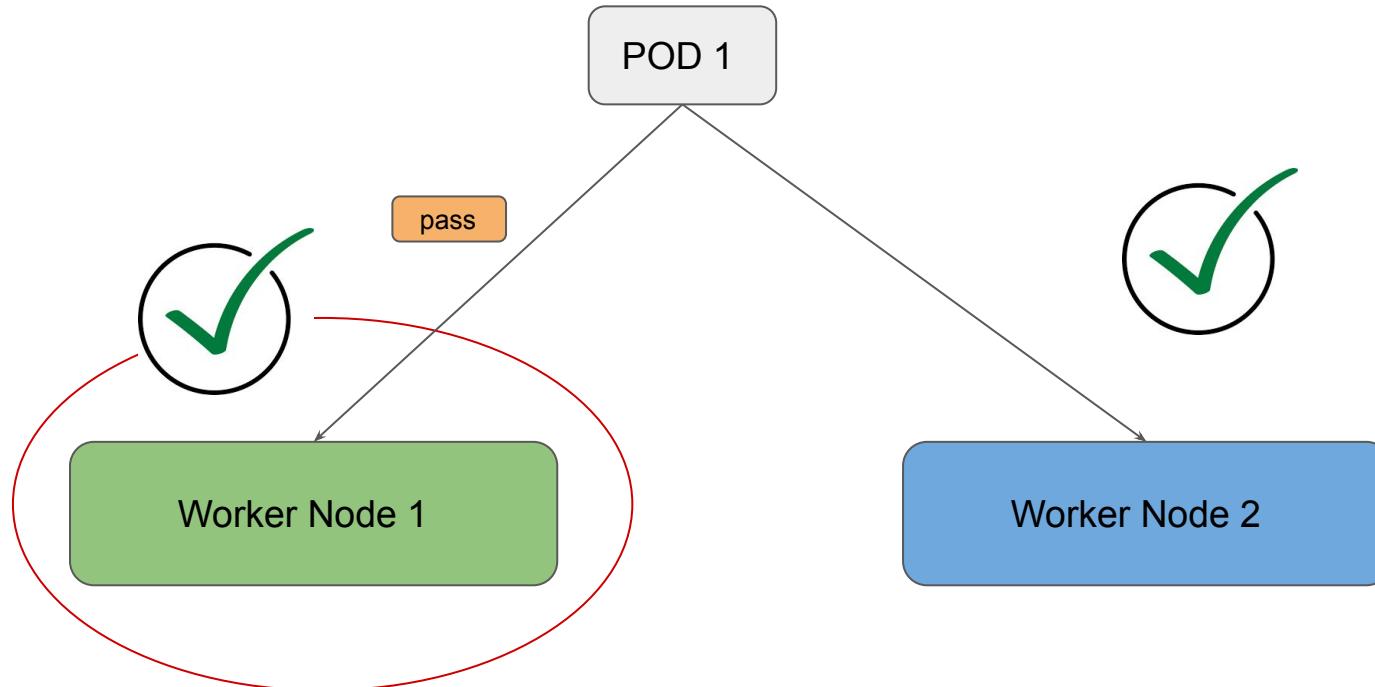
Understanding Taints

Taints are used to repel the pods from a specific nodes.



Understanding Tolerations

- In order to enter the taint worker node, you need a special pass.
- This pass is call toleration.

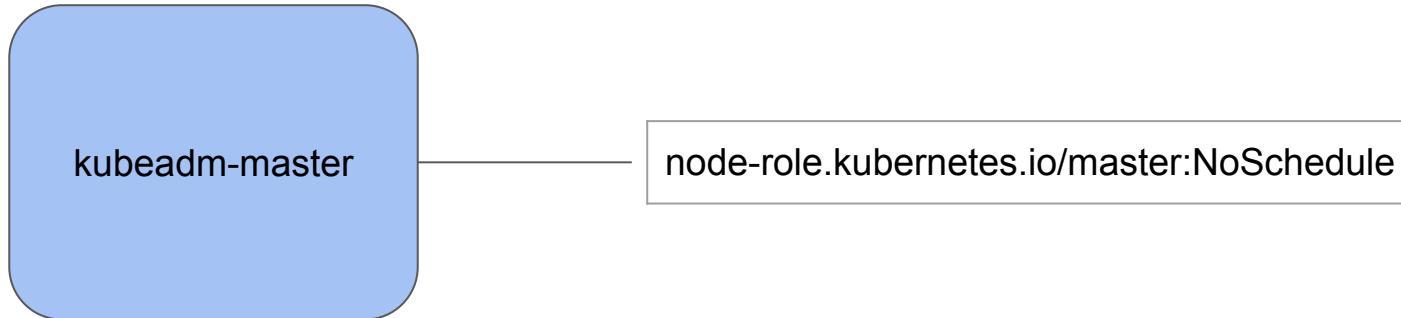


Our Setup

As of now, we have one server where we have kubeadm installed.

The master node has a Taint of NoSchedule

2 approaches: Create Pods with Toleration OR Remove the Taint.

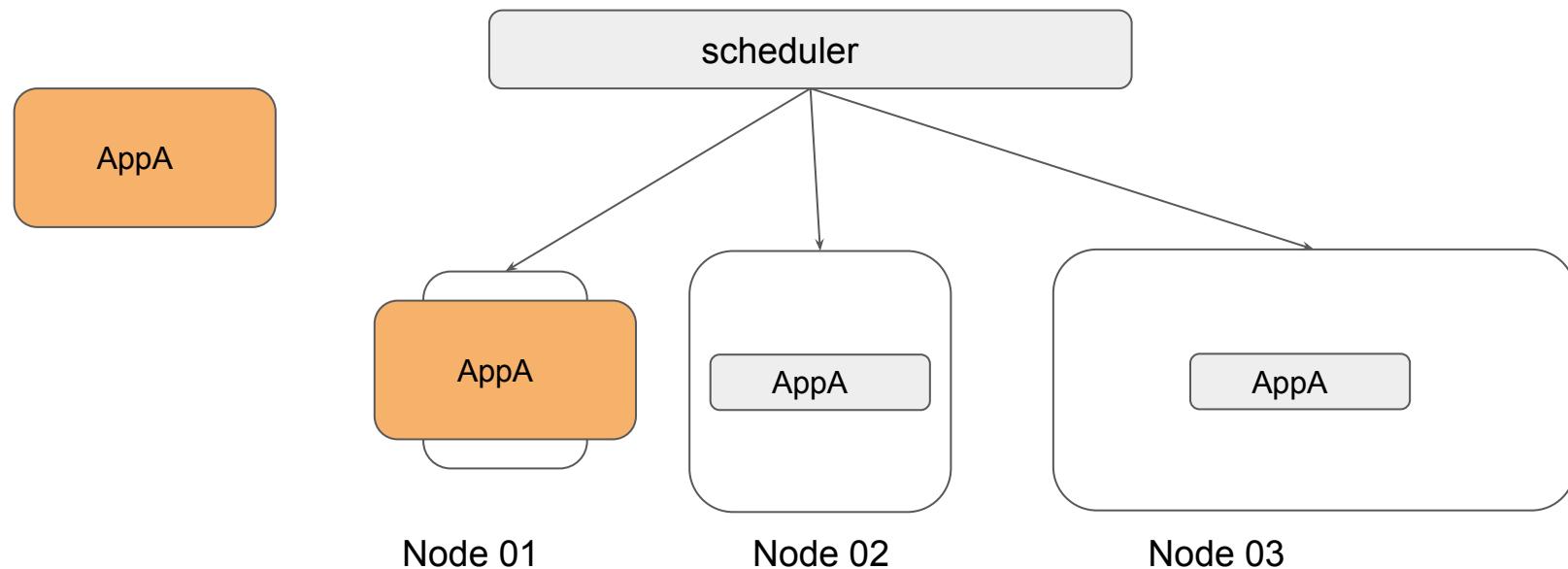


Resource Requests and Limits

Scheduling Objects

Understanding the Basics

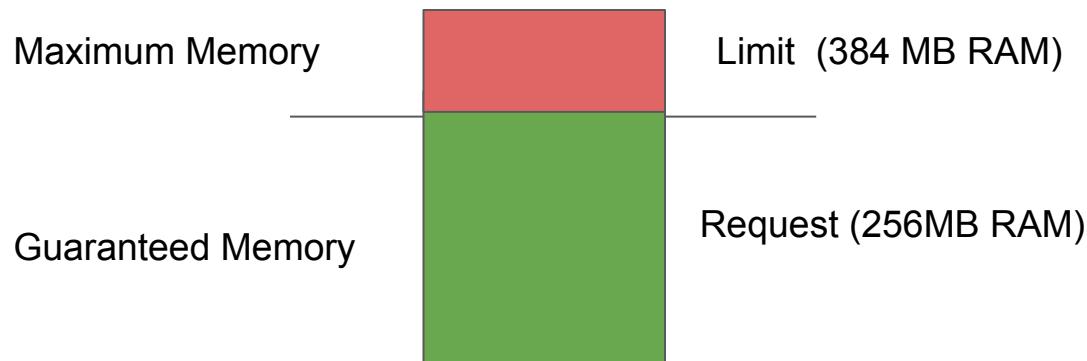
If you schedule a large application in a node which has limited resource, then it will soon lead to OOM or others and will lead to downtime.



Requests and Limits

Requests and Limits are two ways in which we can control the amount of resource that can be assigned to a pod (resource like CPU and Memory)

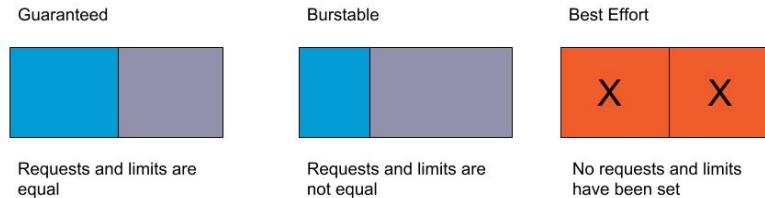
- Requests: Guaranteed to get.
- Limits: Makes sure that container does not take node resources above a specific value.



Requests and Limits

Kubernetes Scheduler decides the ideal node to run the pod depending on the requests and limits.

If your POD requires 8GB of RAM, however there are no nodes within your cluster which has 8GB RAM, then your pod will never get scheduled.



Network Policies

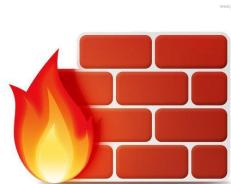
Security Aspect

Understanding the Challenge

By default, pods are non-isolated; they accept traffic from any source.

Example:

- Pod1 can communicate with Pod 2.
- Pod 1 in namespace DEV can communicate with Pod 3 in namespace Staging.



Overview of Network Policy

A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.

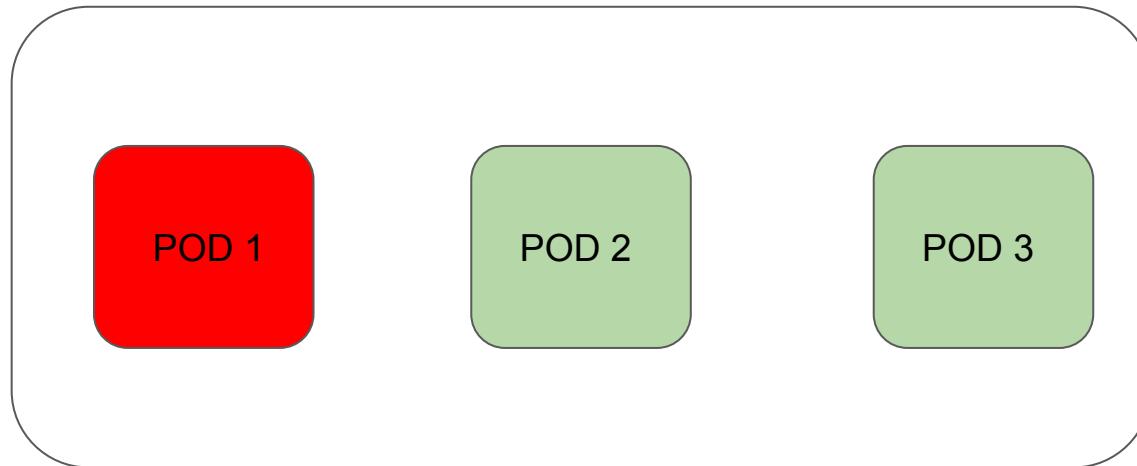
Example:

- POD 1 can only communicate with Pod 5 in the same namespace.
- POD 2 can only communicate with Pod 10 residing in namespace Security.
- No one should be able to communicate with Pod 3.

Use-Case 1 - Pod Compromise

There is a POD named AppA which is compromised.

Security Policy says that the pod should be isolated so that no communication happens.



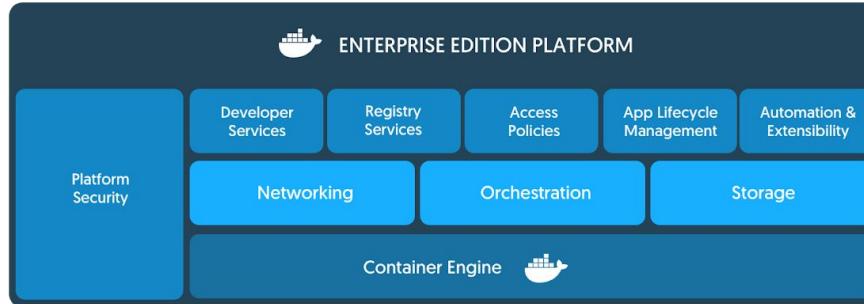
Docker Enterprise Edition (EE)

Build once, use anywhere

Overview of Docker EE

Docker Enterprise Edition (also referred as Docker EE) is designed for enterprise development as well as IT teams who build, ship, and run business-critical applications in production and at scale.

Docker EE is officially supported by the Docker Team.



Docker EE Tiers

There are multiple tiers available for Docker Enterprise edition. Referred as Basic, Standard and Advanced.

Capabilities	Docker Engine - Community	Docker Engine - Enterprise	Docker Enterprise
Container engine and built in orchestration, networking, security	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓
Image management			✓
Container app management			✓
Image security scanning			✓

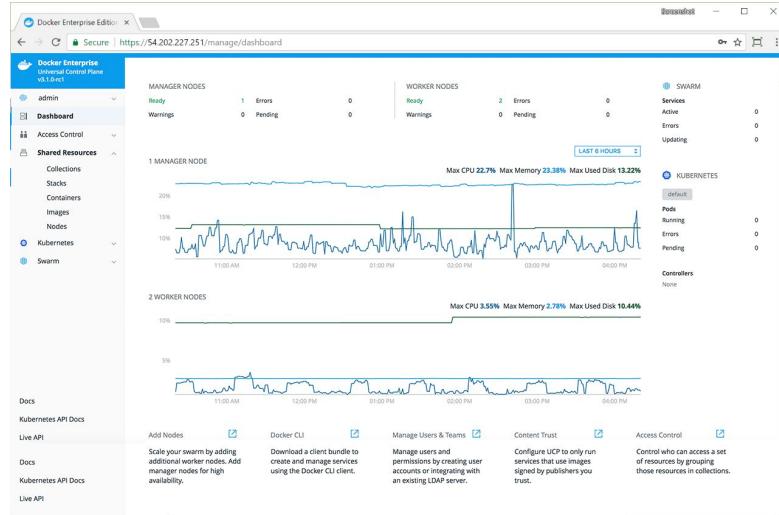
Universal Control Plane

Build once, use anywhere

Overview of Docker UCP

Docker Universal Control Plane (UCP) is the enterprise-grade cluster management solution from Docker.

UCP helps you manage your Docker cluster and applications through a single interface.



UCP Architecture

Universal Control Plane is a containerized application that runs on Docker EE.

Once Universal Control Plane (UCP) instance is deployed, developers and IT operations no longer interact with Docker Engine directly, but interact with UCP instead.



System Requirements for UCP

Minimum Requirements for UCP:

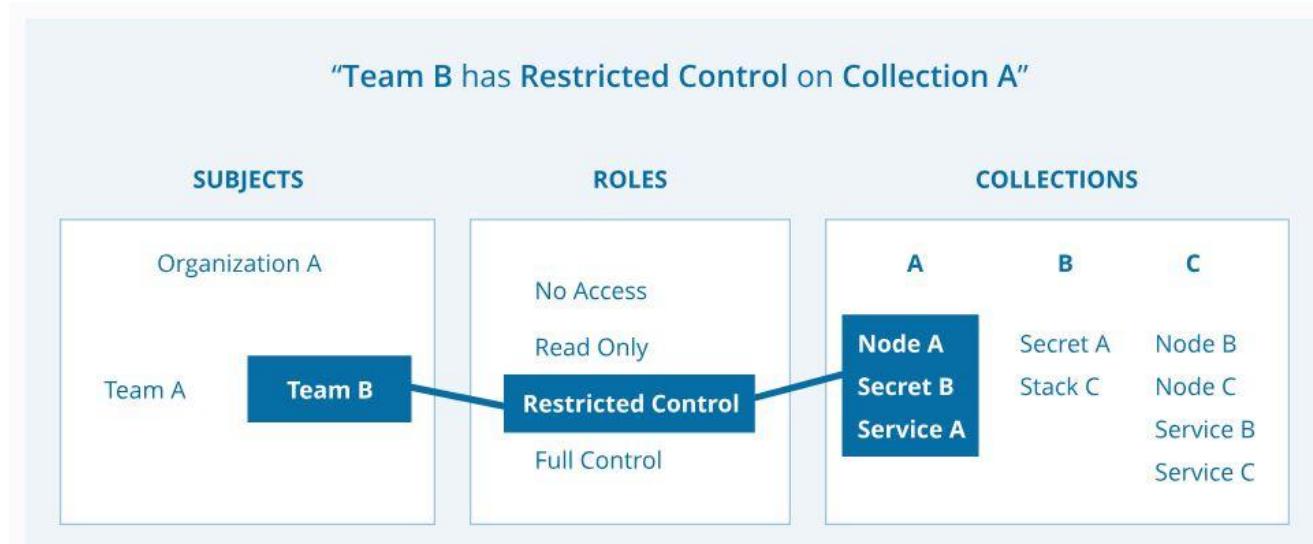
- Docker EE Engine
- 8GB of RAM for Manager Nodes
- 4GB of RAM for Worker nodes
- 5GB of free disk space for the /var partition for manager nodes
- 500MB of free disk space for the /var partition for worker nodes

UCP - Access Control Model

Build once, use anywhere

Getting Started

Docker UCP allows us provides **granular access control** feature which allows us to define various aspects like who can create and edit container resources in your swarm and others.



Understanding Subject

A subject represents a user, team, or organization.

A subject is in-turn granted an appropriate role.

User: Individual User.

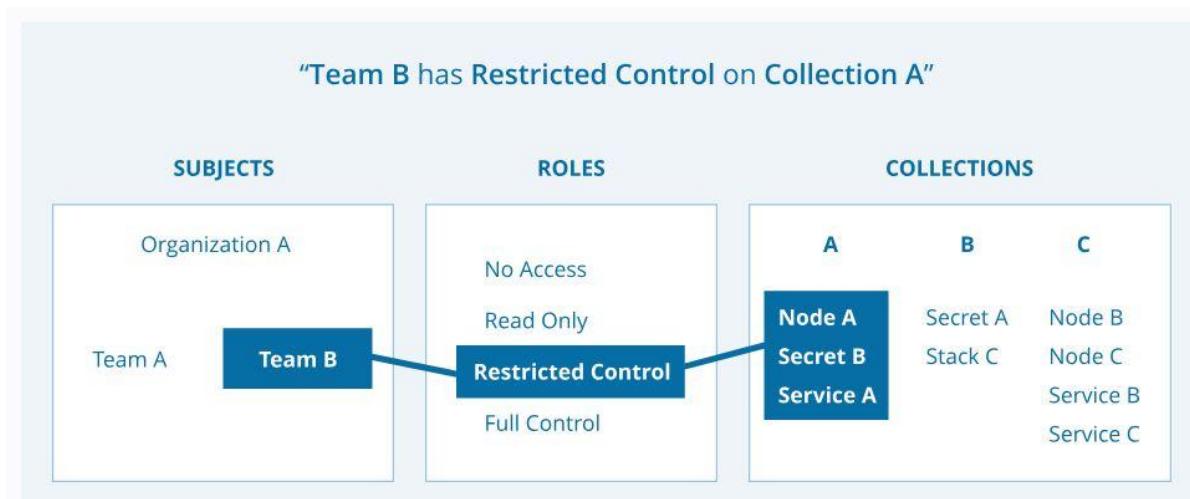
Organization: A group of users that share a specific set of permissions, defined by the roles of the organization.

Team: A group of users that share a set of permissions defined in the team itself.

A team exists only as part of an organization

Understanding Roles

A role is a set of permitted API operations that you can assign to a specific subject and collection



Understanding Collections

Docker EE enables controlling access to swarm resources by using collections.

Swarm resources that can be placed in to a collection include:

- Physical or virtual nodes
- Containers
- Services
- Networks
- Volumes
- Secrets
- Application configs

System Requirements for UCP

Minimum Requirements for UCP:

- Docker EE Engine
- 8GB of RAM for Manager Nodes
- 4GB of RAM for Worker nodes
- 5GB of free disk space for the /var partition for manager nodes
- 500MB of free disk space for the /var partition for worker nodes

DTR Backup

Build once, use anywhere

Overview of DTR Backup

Docker DTR has a backup command that allows us to take the backup.

The backup command doesn't cause downtime for DTR, so you can take frequent backups without affecting your users.

```
docker run --log-driver none -i --rm docker/dtr backup --ucp-url https://172.31.40.237  
-ucp-insecure-tls --ucp-username admin --ucp-password YOUR-PASSWORD-HERE >  
backup.tar
```

Important Pointer related to DTR Backup

This command only creates backups of configurations, and image metadata.

It does not back up users and organizations. Users and organizations can be backed up during a UCP backup.

It also does not back up Docker images stored in your registry.

Backup DTR Images

Build once, use anywhere

Backing up DTR Image

DTR supports multiple backends to store the images.

Depending on the backend that is being used, the backup procedure for images differs.

For local volumes based setup, you can backup the volume associated with DTR registry.

Routing Mesh

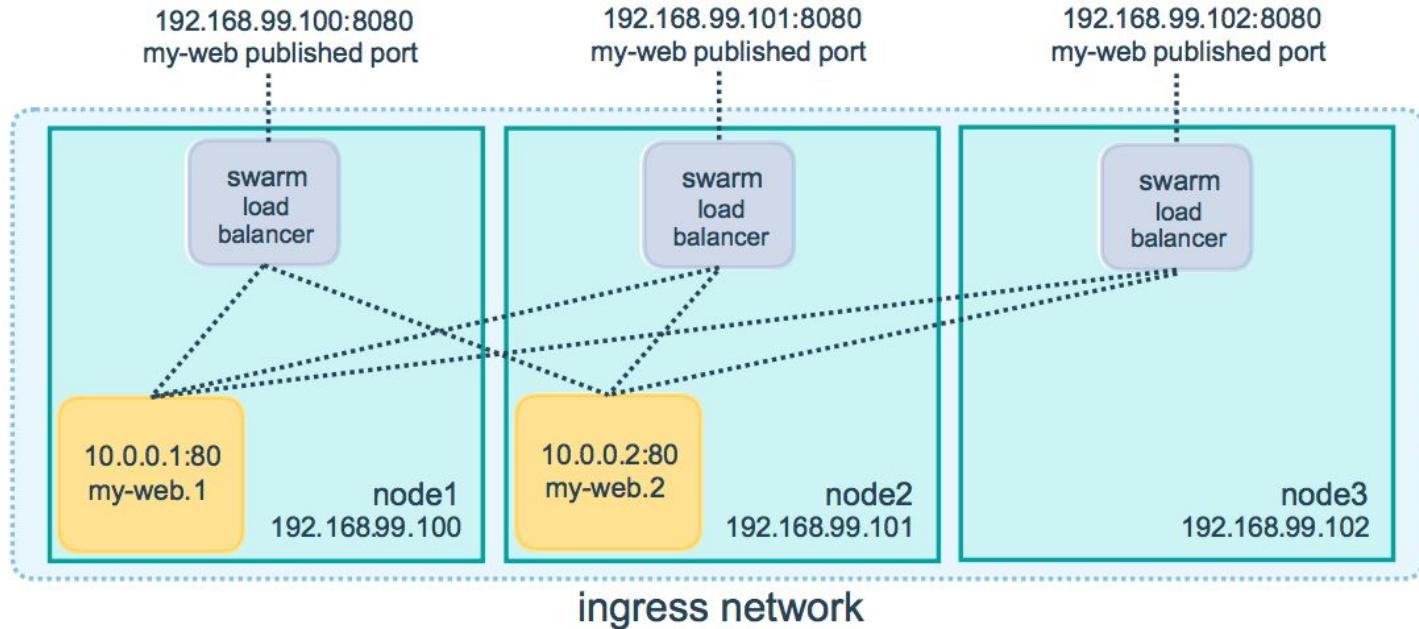
Build once, use anywhere

Overview of Routing Mesh

There can be multiple nodes within a swarm cluster.

When creating a service, the task can be assigned to nodes depending on various factors.

Routing mesh enables each node in the swarm to accept connections on published ports for any service running in the swarm, even if there's no task running on the node.



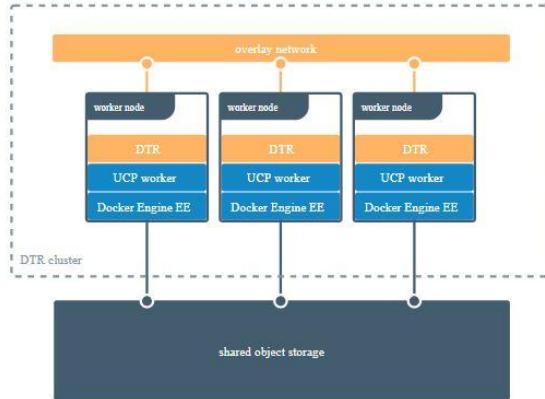
DTR Storage Driver

Build once, use anywhere

Overview of Storage driver

By default, Docker Trusted Registry stores images on the filesystem of the node where it is running, but you should configure it to use a centralized storage backend.

You can configure DTR to use an external storage backend, for improved performance or high availability.



Supported Storage Systems

Some of the supported storage systems in DTR are:

Local:

- NFS
- Bind Mount
- Volume

Cloud Storage Provider:

- AWS S3
- Azure
- Google Cloud

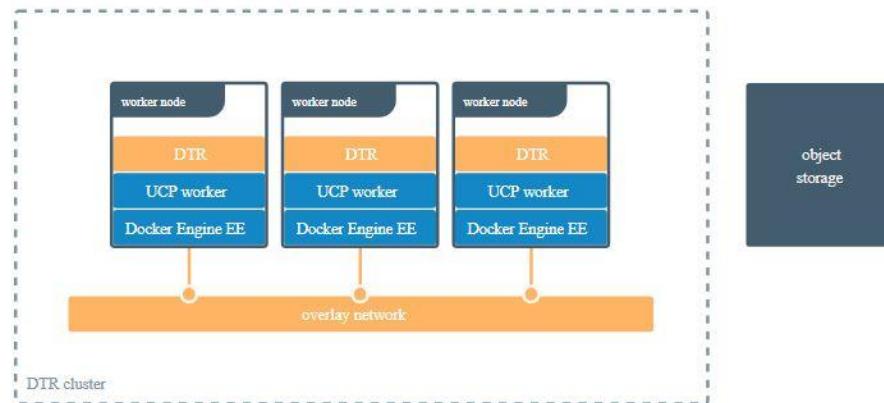
DTR High-Availability

Disaster Recovery

Overview of High Availability Setup

Docker Trusted Registry is designed to scale horizontally as your usage increases. You can add more replicas to make DTR scale to your demand and for high availability.

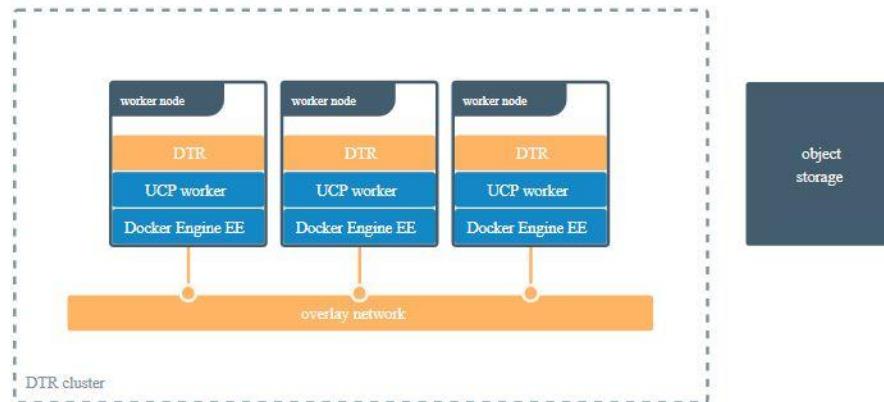
All DTR replicas run the same set of services and changes to their configuration are automatically propagated to other replicas.



Central Storage Backend

If your DTR deployment only has a single replica, you can continue using the local filesystem to store your Docker images.

If your DTR deployment has multiple replicas, for high availability, you need to ensure all replicas are using the same storage backend.



Fault Tolerance

Ensure sufficient amount of replicas are added to provide fault tolerance.

DTR Replicas	Fault Tolerated
1	0
3	1
5	2
7	3

Health Check Endpoints

DTR also exposes several endpoints you can use to assess if a DTR replica is healthy or not:

Endpoints	Description
/_ping:	Checks if the DTR replica is healthy, and returns a simple json response. This is useful for load balancing or other automated health check tasks
/api/v0/meta/cluster_status	Returns extensive information about all DTR replicas.

Important Pointers

To have high-availability on UCP and DTR, you need a minimum of:

- 3 dedicated nodes to install UCP with high availability,
- 3 dedicated nodes to install DTR with high availability,

When a replica fails, the number of failures tolerated by your cluster decreases. Don't leave that replica offline for long.

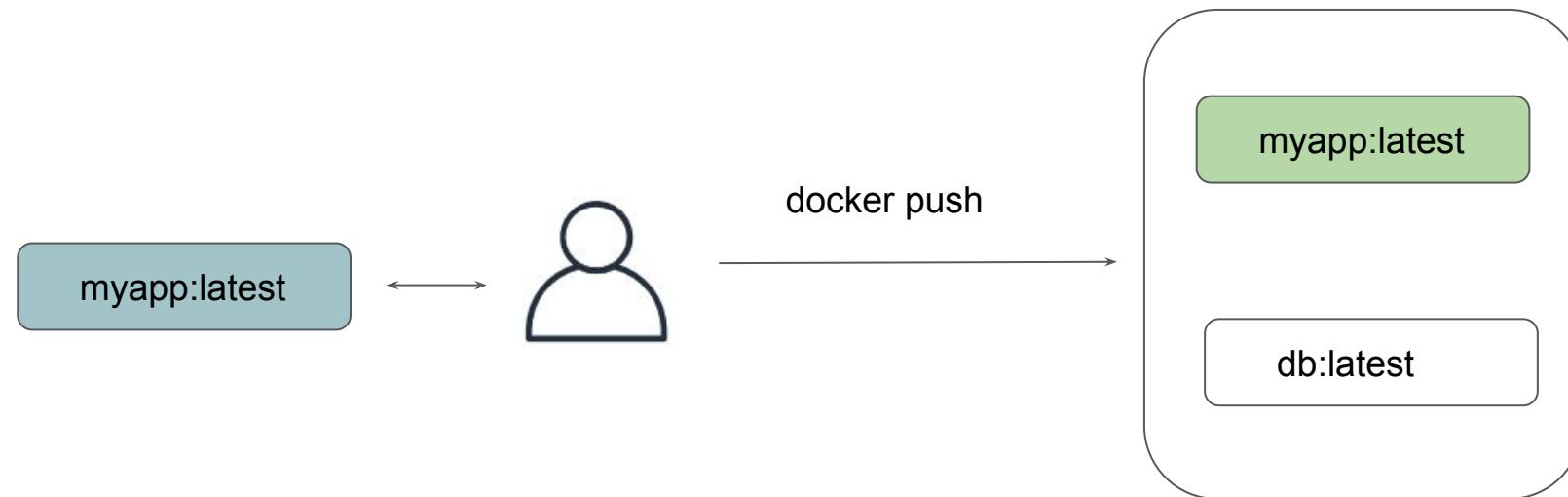
Adding too many replicas to the cluster might also lead to performance degradation, as data needs to be replicated across all replicas.

Immutable Tags in DTR

DTR In-Detail

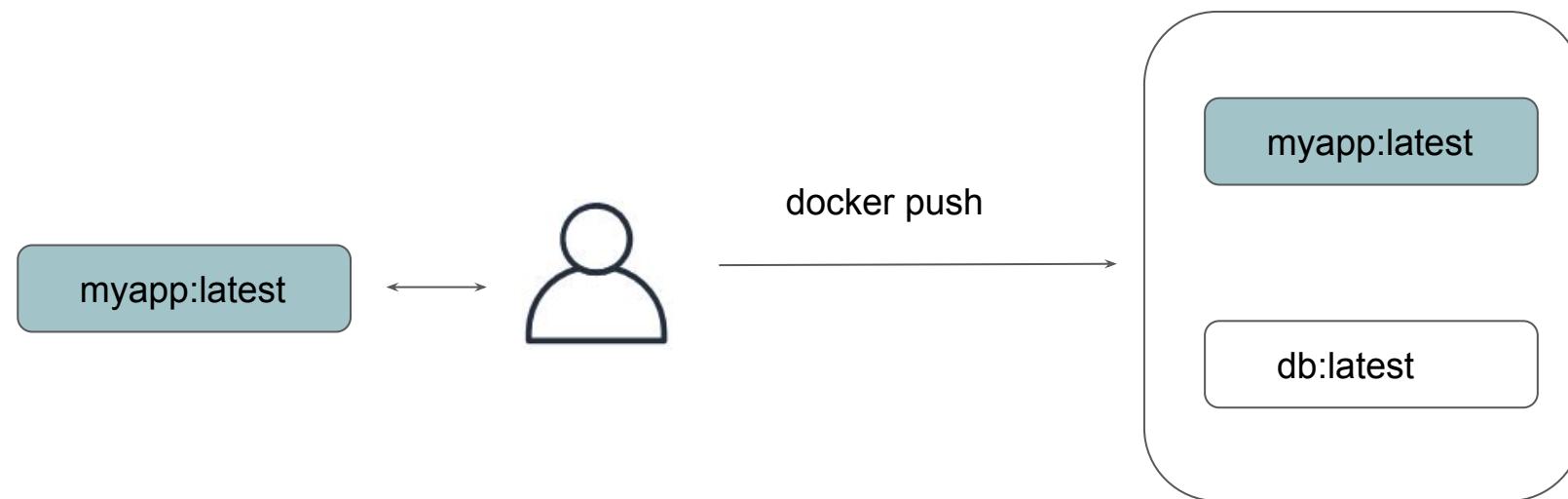
Overview of the Challenge

By default, users with read and write access to a repository can push the same tag multiple times to that repository.



Overview of the Challenge

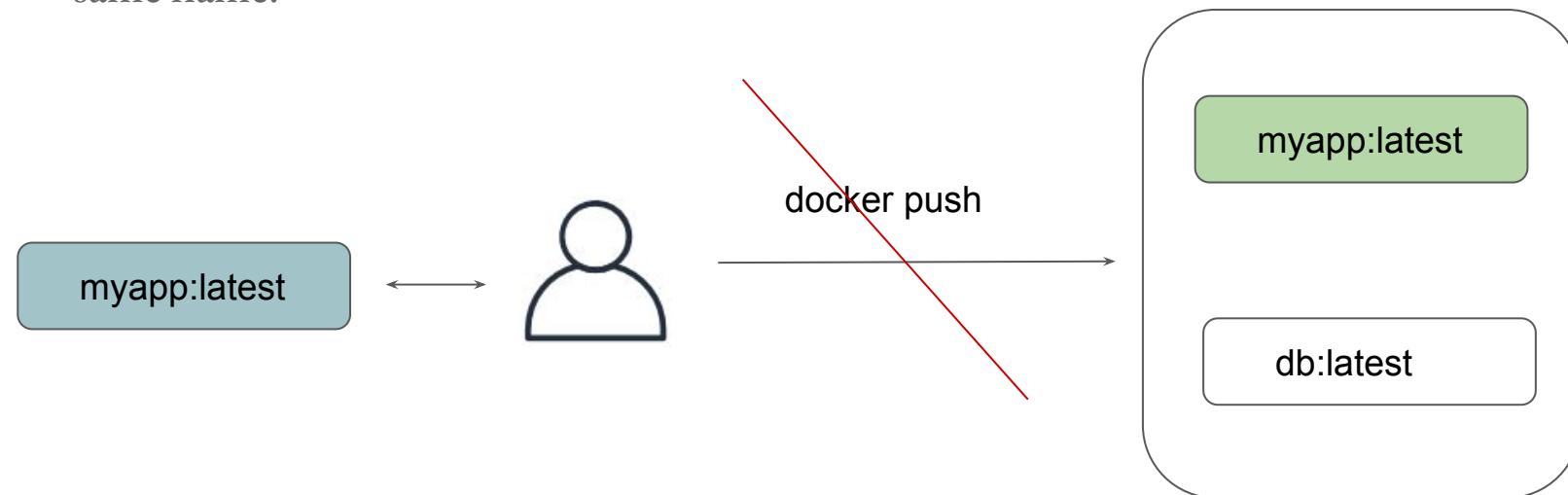
By default, users with read and write access to a repository can push the same tag multiple times to that repository.



Immutable Tags

To prevent tags from being overwritten, you can configure a repository to be immutable.

Once configured, DTR will not allow anyone else to push another image tag with the same name.



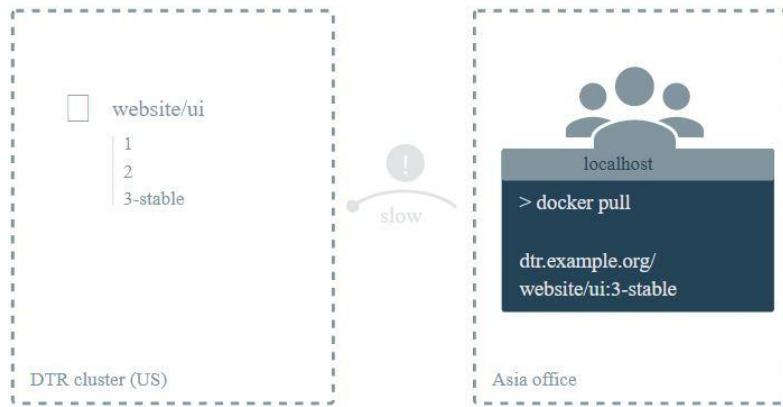
DTR Cache

DTR In Detail

Understanding the Challenge

The further away you are from the geographical location where DTR is deployed, the longer it will take to pull and push images.

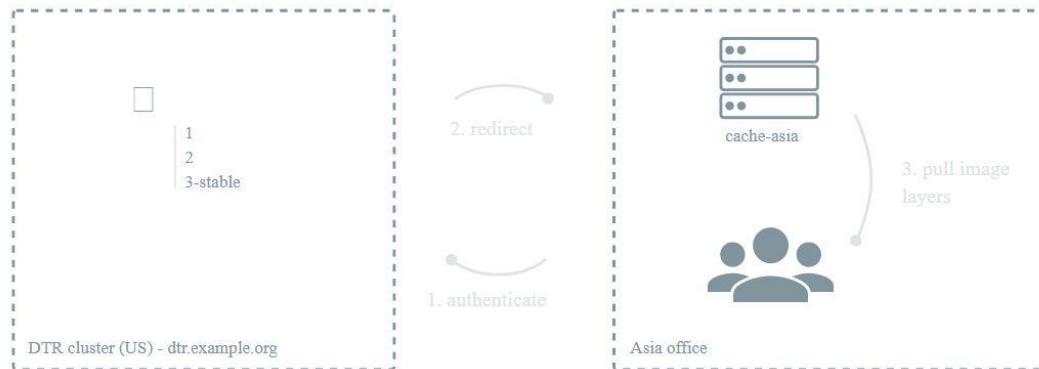
This happens because the files being transferred from DTR to your machine need to travel a longer distance, across multiple networks.



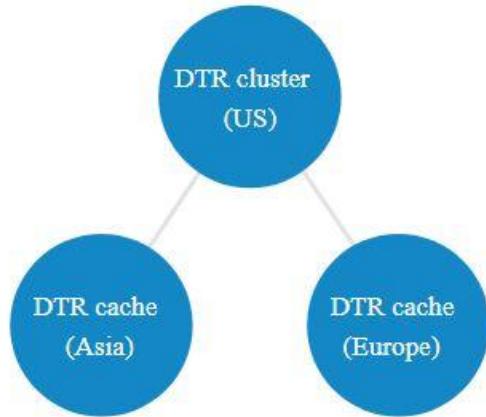
Implementing Cache at Location

To decrease the time to pull an image, you can deploy DTR caches geographically closer to users.

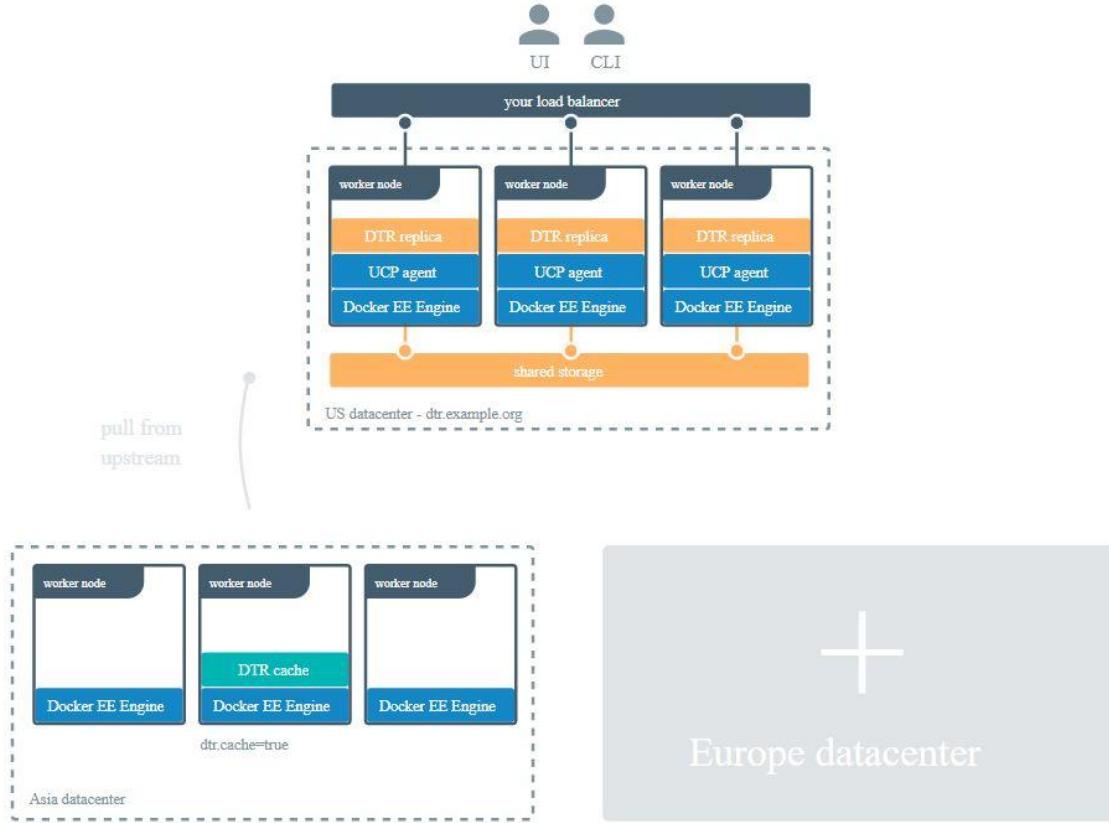
Caches are transparent to users, since users still log in and pull images using the DTR URL address. DTR checks if users are authorized to pull the image, and redirects the request to the cache.



Implementing Cache at Location



Architecture



Setting Orchestrator in UCP

UCP In-Detail

Orchestrator Types in UCP

Docker UCP supports both Swarm and Kubernetes.

When you add a node to the cluster, the node's workloads are managed by a default orchestrator, either Docker Swarm or Kubernetes.

When you install Docker Enterprise, new nodes are managed by Docker Swarm, but you can change the default orchestrator to Kubernetes in the administrator settings.

Scheduler

SET ORCHESTRATOR TYPE FOR NEW NODES



Swarm



Kubernetes

Orchestrator Types For Nodes

You can change the current orchestrator for any node that's joined to a Docker Enterprise cluster. The available orchestrator types are Kubernetes, Swarm, and Mixed.

The Mixed type enables workloads to be scheduled by Kubernetes and Swarm both on the same node.

When you change the orchestrator type for a node, existing workloads are evicted, and they're not migrated to the new orchestrator automatically

Change Orchestrator Type via CLI

To change the orchestrator type for a node from Swarm to Kubernetes:

```
docker node update --label-add com.docker.ucp.orchestrator.kubernetes=true <node-id>  
docker node update --label-add com.docker.ucp.orchestrator.kubernetes=true <node-id>
```

Container Security Scanning

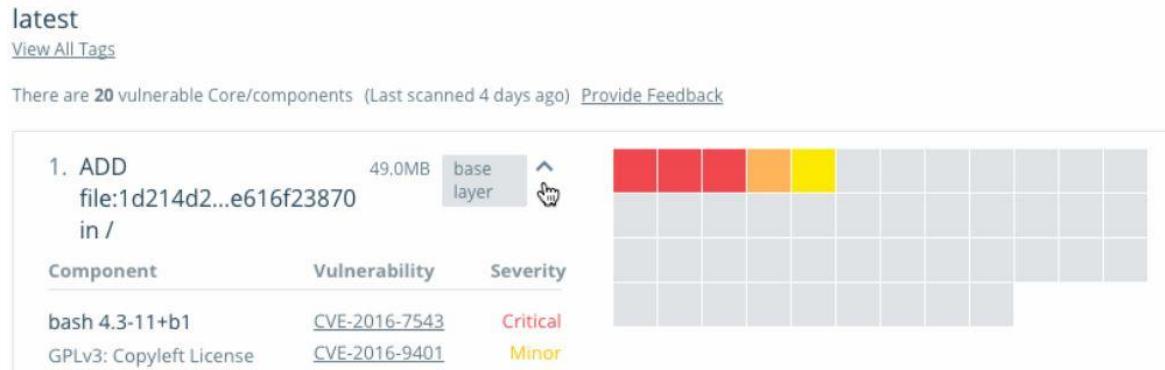
Container Security



Getting Started

Docker Containers can have security vulnerabilities.

If blindly pulled and if containers are running in production, it can result in breach.



Overview of Security Scanning in DTR

DTR allows us to perform security scan for the containers.

These scan can perform “On Push” or even manually.

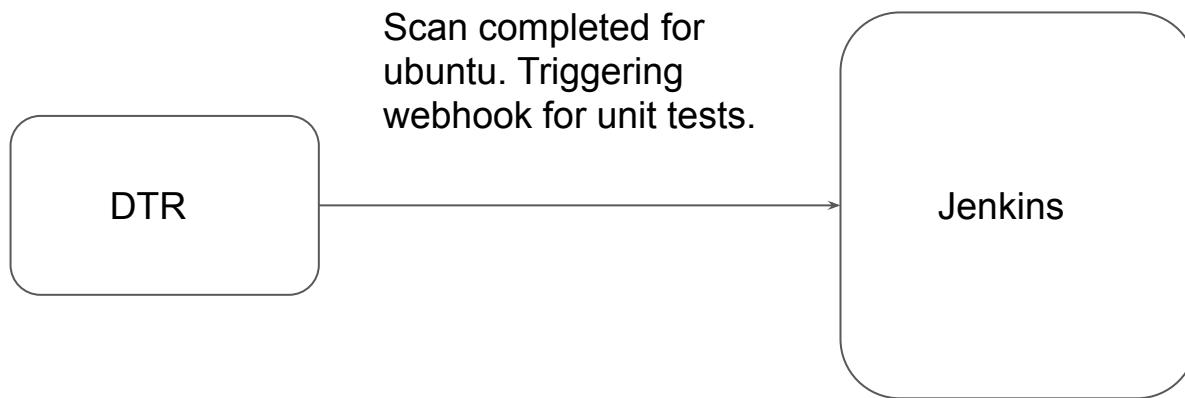
admin / webserver								
	Info	Tags	Webhooks	Promotions	Pruning	Mirrors	Settings	Activity
	Image	Os/Arch	Image ID	Size (Compressed)	Signed	Last Pushed	Vulnerabilities	
<input type="checkbox"/>	ubuntu	linux / amd64	9361ce633ff1	43.56 MB		12 minutes ago by admin	5 Critical 31 Major 18 Minor	View details
<input type="checkbox"/>	v1	linux / amd64	d8233ab899d4	755.9 kB		29 minutes ago by admin	Clean	View details

DTR Webhooks

Build once, use anywhere

Overview of the Webhooks

You can configure DTR to automatically post event notifications to a webhook URL of your choosing



Overview of the Webhooks

```
{  
  "type": "TAG_PUSH",  
  "createdAt": "2019-05-15T19:39:40.607337713Z",  
  "contents": {  
    "namespace": "foo",  
    "repository": "bar",  
    "tag": "latest",  
    "digest": "sha256:b5bb9d8014a0f9b1d61e21e796d78dccdf1352f23cd32812f4850b878ae4944c",  
    "imageName": "foo/bar:latest",  
    "os": "linux",  
    "architecture": "amd64",  
    "author": "",  
    "pushedAt": "2015-01-02T15:04:05Z"  
  },  
  "location": "/repositories/foo/bar/tags/latest"  
}
```

UCP Client Bundles

Build once, use anywhere

Overview of UCP Client Bundles

A client bundle is a group of certificates downloadable directly from the Docker Universal Control Plane (UCP).

Depending on the permission associated with the user, you can now execute docker swarm commands from your remote machine that take effect on the remote cluster.

For example:

- You can create a new service in UCP from your laptop.
- Login to remote container from your laptop without SSH (via API)

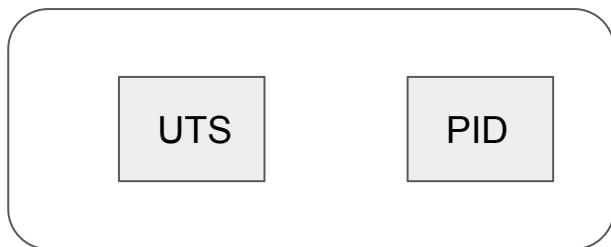
Linux Namespaces

Build once, use anywhere

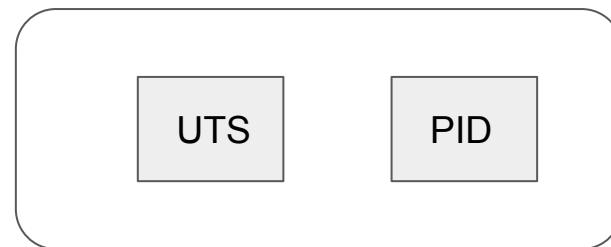
Overview of Namespaces

Docker uses a technology called namespaces to provide the isolated work space called the container

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.



Namespace A



Namespace B

Namespaces in Linux

Currently Linux provides six different types of namespaces as follows:

- Inter-Process Communication (IPC)
- Process (pid)
- Network (net)
- User Id (user)
- Mount (mnt)
- UTS

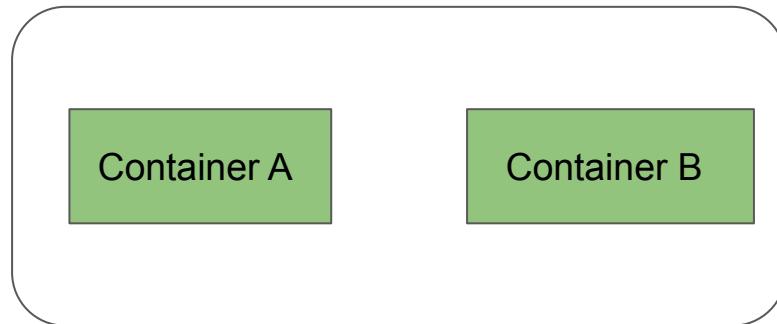
Control Groups

Build once, use anywhere

Overview of Control Groups

Control Groups (cgroups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

1 CPU core
1 GB RAM



Docker Host

Overview of Control Groups

Control Groups (cgroups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

1 CPU core
1 GB RAM



Docker Host

Limits CPU for Containers

There are various ways in which you can limit the CPU for containers, these include:

Approaches	Description
--cpus=<value>	If host has 2 CPUs and if you set --cpus=1, then container is guaranteed at most one CPU. You can even specify --cpus=0.5
--cpuset-cpus	Limit the specific CPUs or cores a container can use. A comma-separated list or hyphen-separated range of CPUs a container can use. 1st CPU is numbered 0 2nd CPU is numbered 1. Value of 0-3 means usage of first, second, third and fourth CPU. Value of 1,3 means second and fourth CPU.

Reservation vs Limit

Build once, use anywhere

Getting Started

By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows.

It is important not to allow a running container to consume too much of the host machine's memory.

On Linux hosts, if the kernel detects that there is not enough memory to perform important system functions, it throws an **Out Of Memory** Exception, and starts killing processes to free up memory.

Reservation vs Limit

Limit imposes an upper limit to the amount of memory that can potentially be used by a Docker container.

Reservations, on the other hand, is less rigid.

When the system is running low on memory and there is contention, reservation tries to bring the container's memory consumption at or below the reservation limit.

Swarm MTLS

Build once, use anywhere

Getting Started

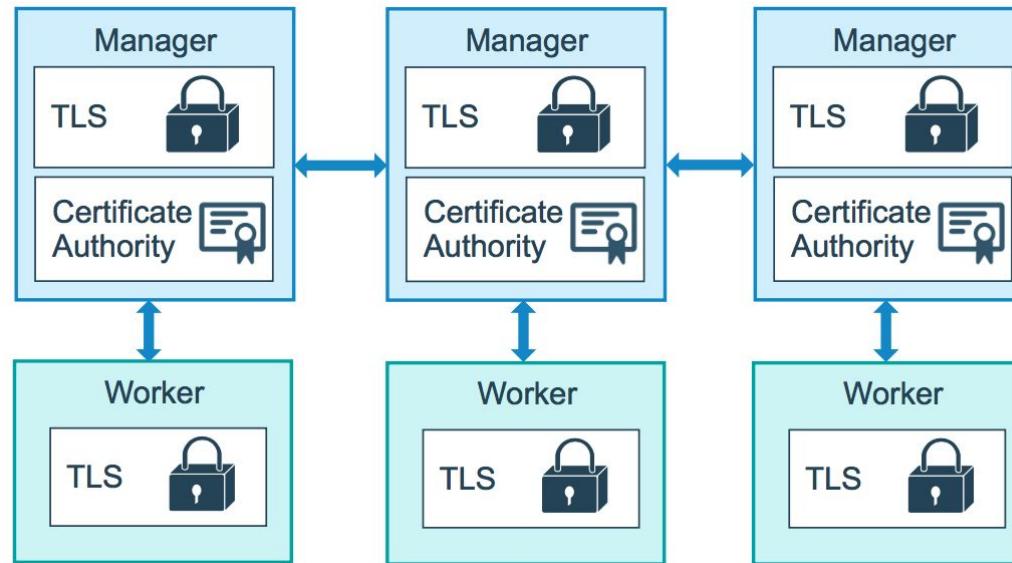
The nodes in a swarm use mutual Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the swarm.

By default, the manager node generates a new root Certificate Authority (CA) along with a key pair, which are used to secure communications with other nodes that join the swarm.

You can specify your own externally-generated root CA, using the `--external-ca` flag of the docker swarm init command.

Overview of Generated Certificate

Each time a new node joins the swarm, the manager issues a certificate to the node.



Overview of Tokens

The manager node also generates two tokens to use when you join additional nodes to the swarm: one **worker** token and one **manager** token

Each token includes the digest of the root CA's certificate and a randomly generated secret.

When a node joins the swarm, the joining node uses the digest to validate the root CA certificate from the remote manager.

Certificate Details

By default, each node in the swarm renews its certificate every three months.

You can configure this interval by running the `docker swarm update --cert-expiry <TIME PERIOD>`

In the event that a cluster CA key or a manager node is compromised, you can rotate the swarm root CA so that none of the nodes trust certificates signed by the old root CA anymore.

Rotating the CA Certificate

Run `docker swarm ca --rotate` to generate a new CA certificate and key.

In the above command, docker generated a cross-signed certificate signed by previous old CA.

After every node in the swarm has a new TLS certificate signed by the new CA, Docker forgets about the old CA certificate and key material, and tells all the nodes to trust the new CA certificate only.

Join tokens also changes accordingly.

Managing Secrets in Docker Swarm

Build once, use anywhere

Overview of Docker Secrets

Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it.

Secrets are encrypted during transit and at rest in a Docker swarm.

A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

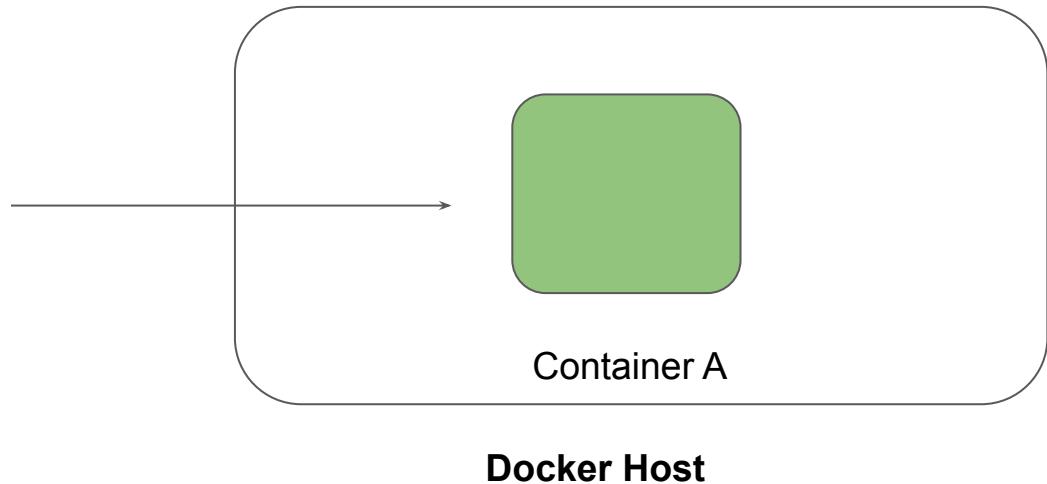
Linux Capabilities

Security Aspect

Understanding Capabilities

There are several types of capabilities which Linux provides to have granular access at the application level.

Capability	Action
SETPCAP	Yes
CHOWN	Yes
SYS_MODULE	No
LINUX_IMMUTABLE	No



Privileged container

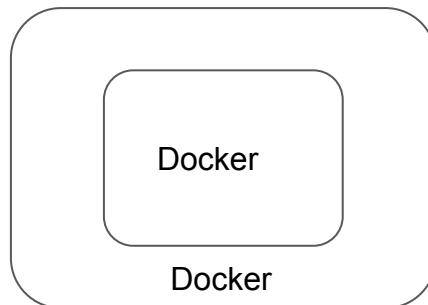
Security Aspect

Understanding the Challenge

By default, docker container does not have many capabilities assigned to it.

- Docker containers are also not allowed to access any devices.

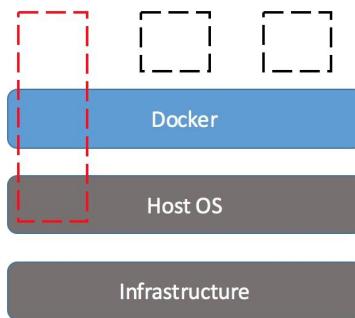
Hence, by default, docker container cannot perform various use-cases like run docker container inside a docker container



Privileged Containers

Privileged Container can access all the devices on the host as well as have configuration in AppArmor or SELinux to allow the container nearly all the same access to the host as processes running outside containers on the host.

- Limits that you set for privileged containers will not be followed.
- Running a privileged flag gives all the capabilities to the container



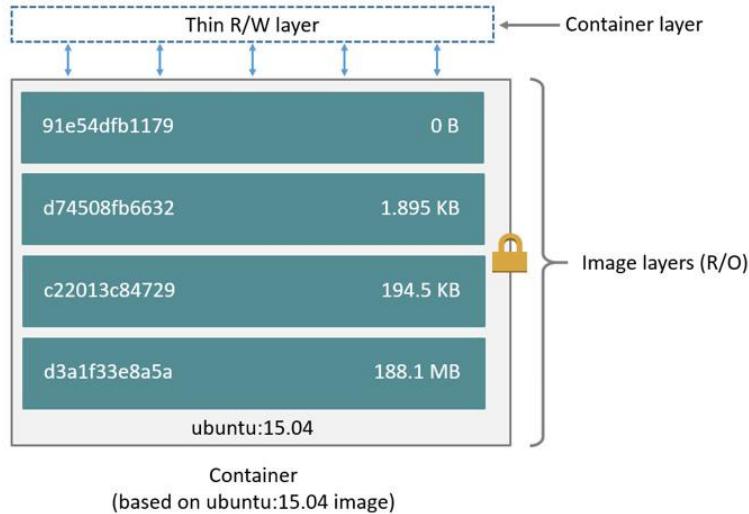
Storage Drivers

Build once, use anywhere

Overview of Storage Drivers

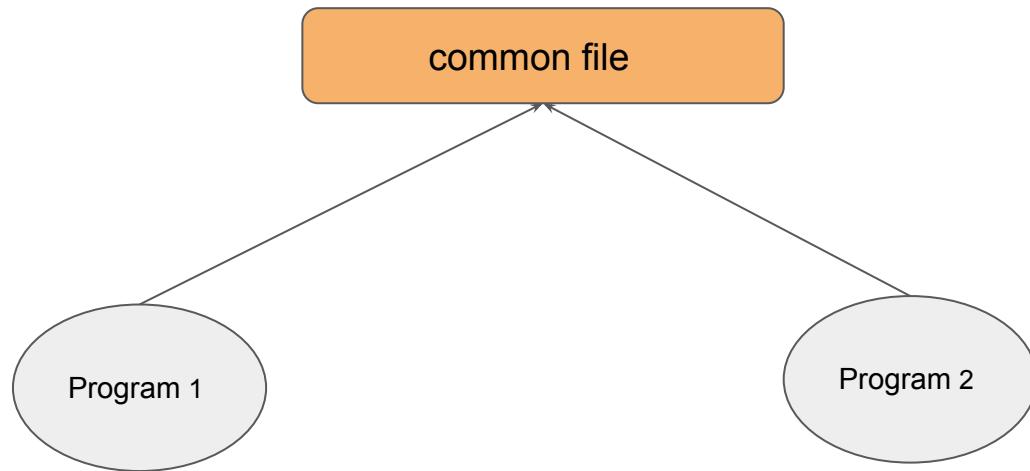
- A Docker image is built up from a series of layers.
- Storage Driver piece all these things together for you
- There are multiple storage drivers available with different trade-off..

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

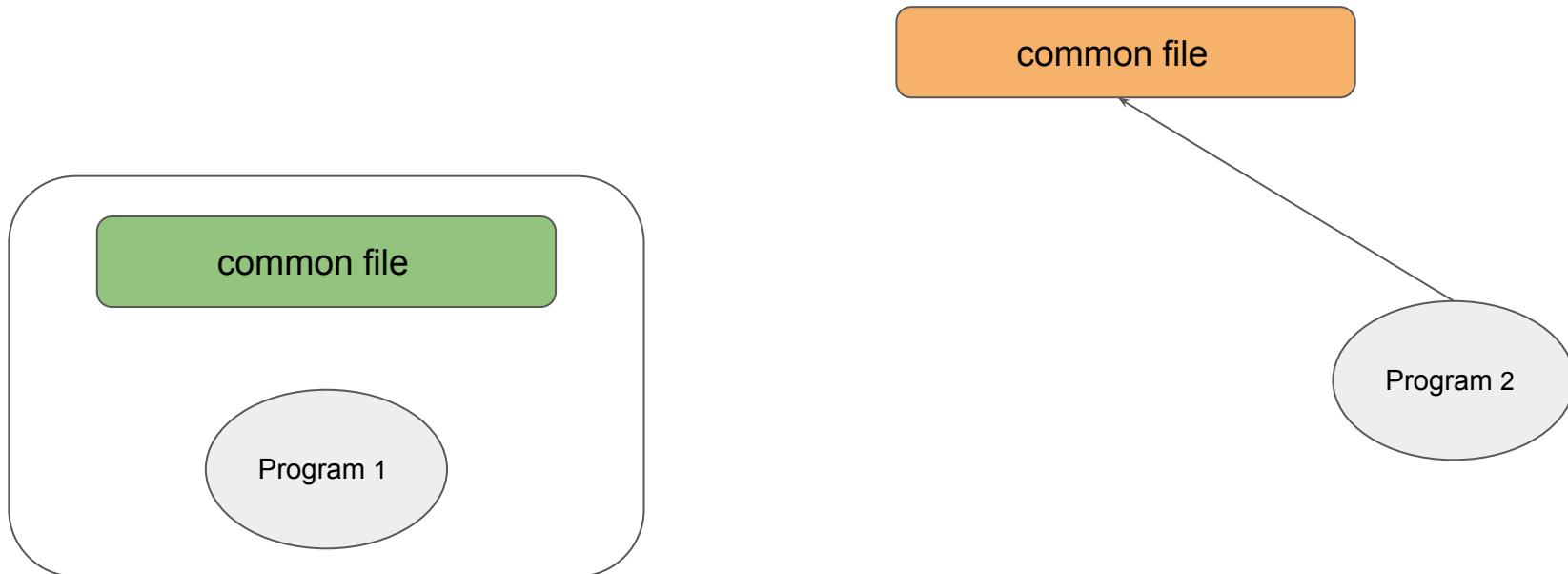


The copy-on-write (CoW) strategy

"Copy on write" means: everyone has a single shared copy of the same data until it's written, and then a copy is made.



The copy-on-write (CoW) strategy



Various Storage Drivers Available

There are various storage drivers available in Docker, these includes:

- AUFS
- Device Mapper
- OverlayFS
- ZFS
- VFS
- Brtfs

Block vs Object Storage

Everything can be lost

Block Storage

In block storage, the data is stored in terms of blocks

Data stored in blocks are normally read or written entirely a whole block at the same time

Most of the file systems are based on block devices.

Block Storage

Every block has an address and application can be called via SCSI call via it's address

There is no storage side meta-data associated with the block except the address.

Thus block has no description, no owner

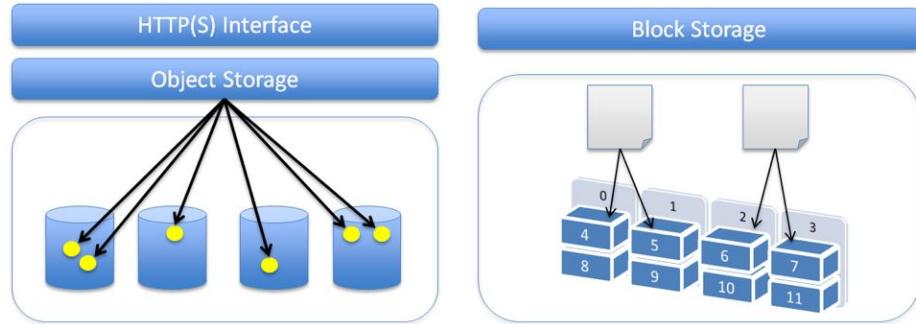
Object Storage

Object storage is a data storage architecture that manages data as objects as opposed to blocks of storage.

An object is defined as a data (file) along with all it's meta-data which is combined together as an object.

This object is given an ID which is calculated from the content of the object (from the data and metadata) . Application can then call object with the unique object ID.

Difference



- Store virtually unlimited files.
 - Maintain file revisions.
 - HTTP(S) based interface.
 - Files are distributed in different physical nodes.
- File is split and stored in fixed sized blocks.
 - Capacity can be increased by adding more nodes.
 - Suitable for applications which require high IOPS, database, transactional data.

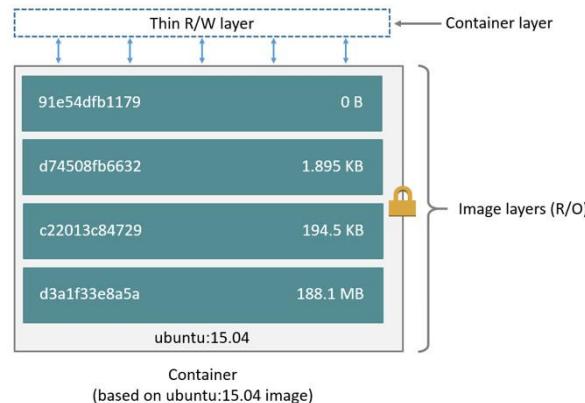
Docker Volumes

Build once, use anywhere

Challenges with files in Container Writable Layer

By default all files created inside a container are stored on a writable container layer. This means that:

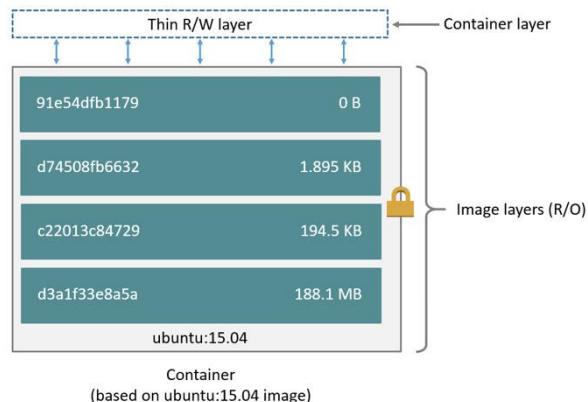
The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.



Challenges with files in Container Writable Layer - Part 2

Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel.

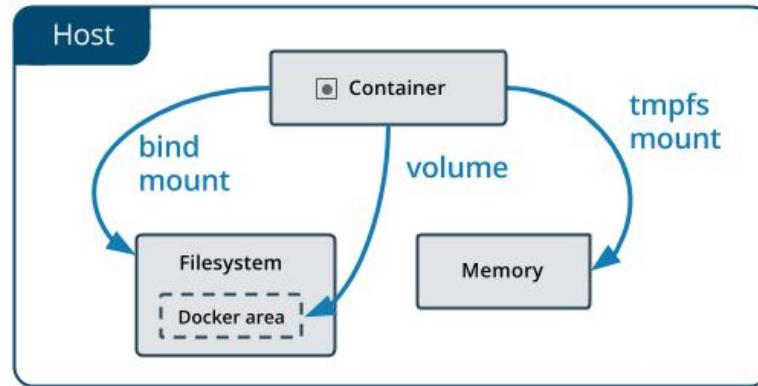
This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.



Ideal Approach for Persistent Data

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.

If you're running Docker on Linux you can also use a tmpfs mount.



Important Pointers to Remember

A given volume can be mounted into multiple containers simultaneously.

When no running container is using a volume, the volume is still available to Docker and is not removed automatically.

When you mount a volume, it may be named or anonymous. Anonymous volumes are not given an explicit name when they are first mounted into a container, so Docker gives them a random name that is guaranteed to be unique within a given Docker host.

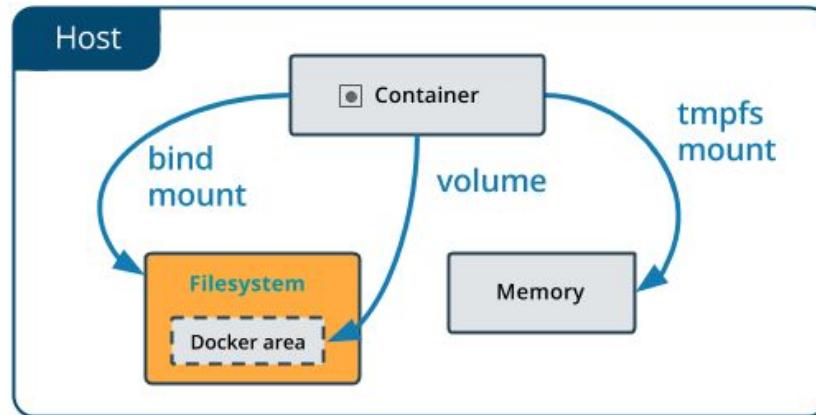
Bind Mounts

Build once, use anywhere

Understanding Bind Mounts

When you use a bind mount, a file or directory on the host machine is mounted into a container.

The file or directory is referenced by its full or relative path on the host machine.



Automatically Remove Volume on Container Removal

Build once, use anywhere

Getting Started

Whenever a container is created with -dt flag and if the main process completes/stops, then it goes into the Exited stage.

We can see list of all containers (Running/Exited) with `docker ps -a` command

Many a times, containers performs a job and we want container to automatically get deleted once it exits. This can be achieved with the `--rm` option.

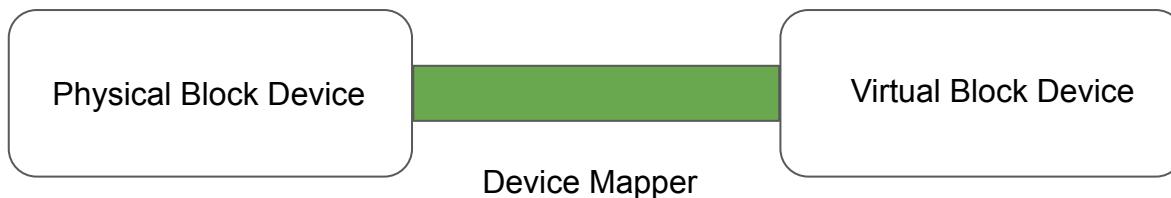
Device Mapper

Build once, use anywhere

Overview of Device Mapper

The device mapper is a framework provided by the Linux kernel for mapping physical block devices onto higher-level virtual block devices.

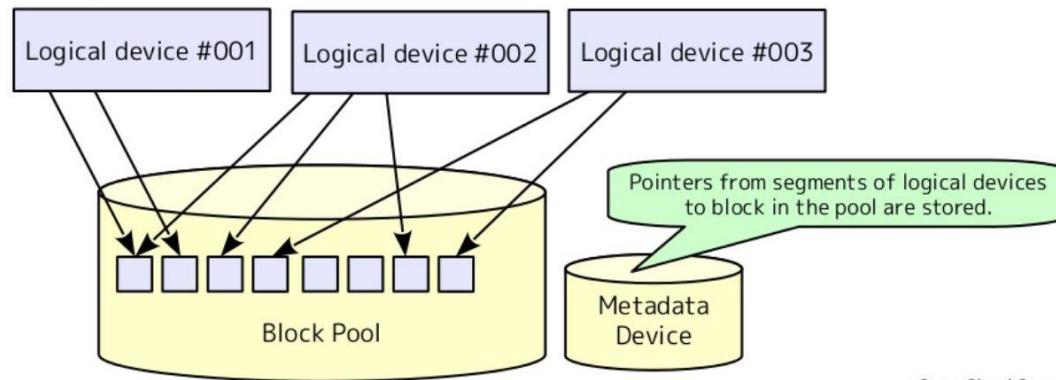
Device mapper works by passing data from a virtual block device, which is provided by the device mapper itself, to another block device.



Overview of Device Mapper

Device mapper creates logical devices on top of physical block device and provides feature like :-

- RAID, Encryption, Cache and various others.



Open Cloud Campus

Few Important Pointers

There are two modes for devicemapper storage driver:

i) loop-lvm mode:

- Should only be used in testing environment.
- Makes use of loopback mechanism which is generally on the slower side.

ii) direct-lvm mode:

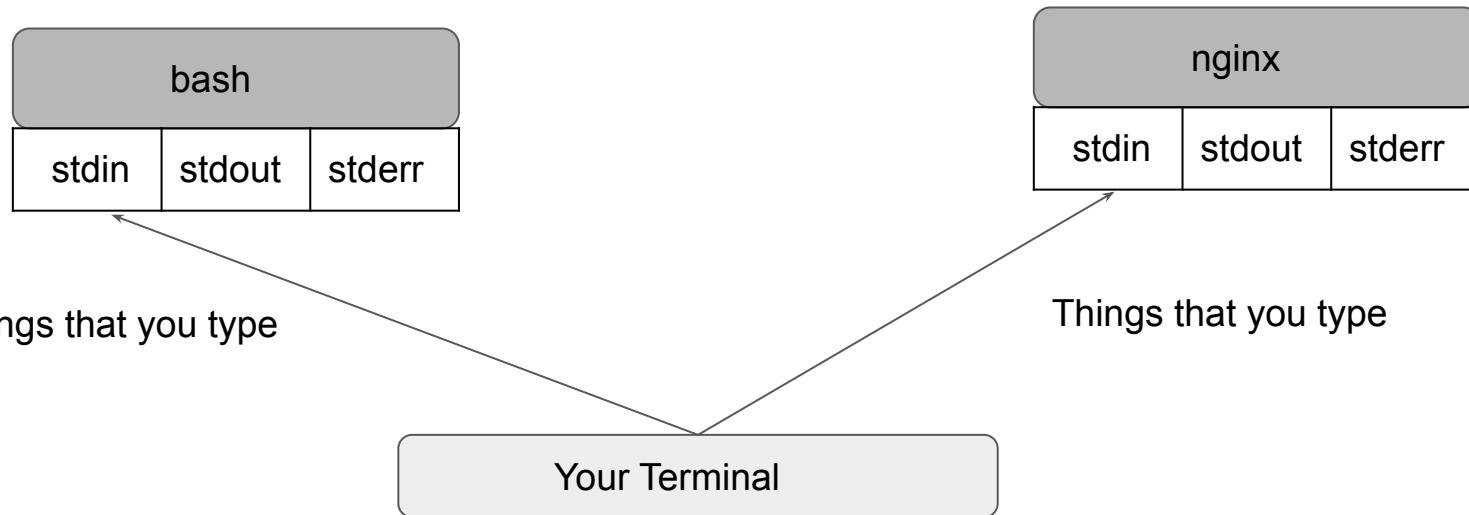
- Production hosts using the devicemapper storage driver must use direct-lvm mode.
- This is much more faster then the loop-lvm mode.

Logging Drivers

Build once, use anywhere

Getting Started

UNIX and Linux commands typically open three I/O streams when they run, called **STDIN**, **STDOUT**, and **STDERR**



Logging Drivers in Docker

There are a lot of logging driver options available in Docker, some of these include:

- json-file
- none
- syslog
- local
- journald
- splunk
- awslogs

The docker logs command is not available for drivers other than json-file and journald.

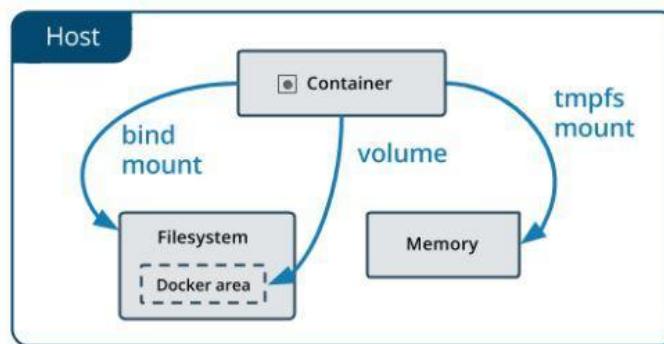
Volumes in Kubernetes

Security Aspect

Two Challenges

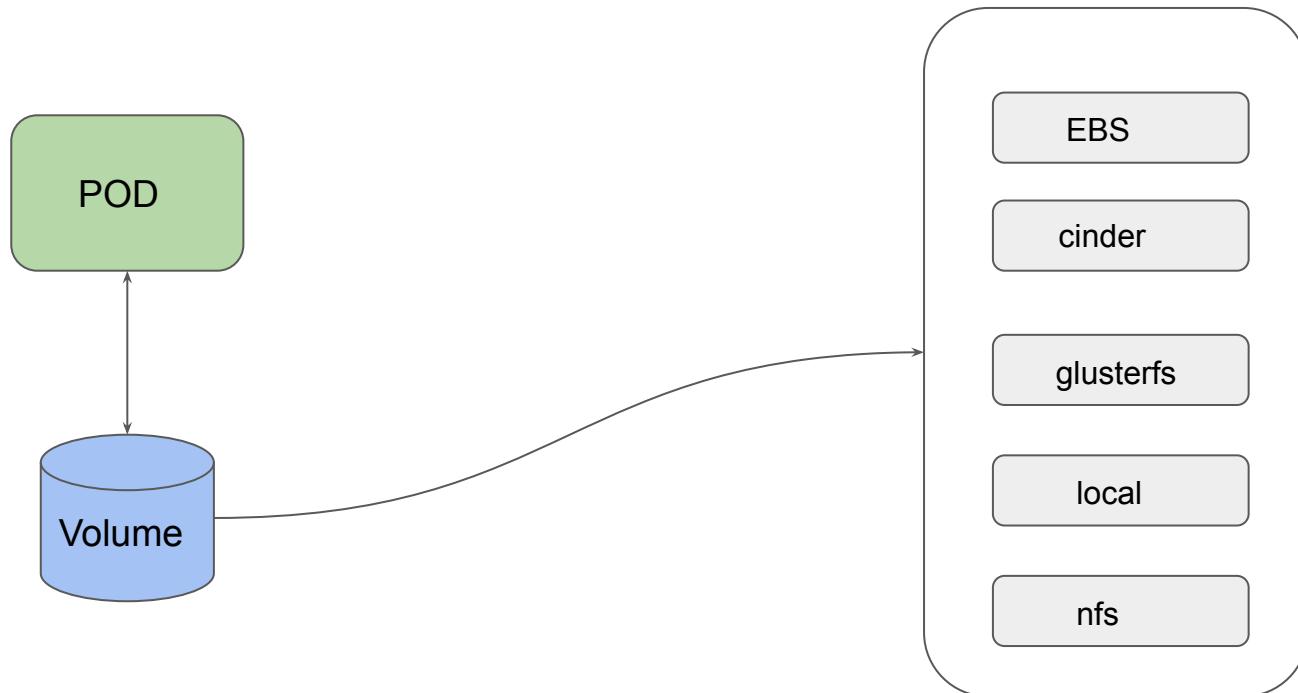
On-disk files in a Container are ephemeral.

When there are multiple containers who wants to share same data, it becomes a challenge.



Volumes in Kubernetes

One of the benefits of Kubernetes is that it supports multiple types of volumes.



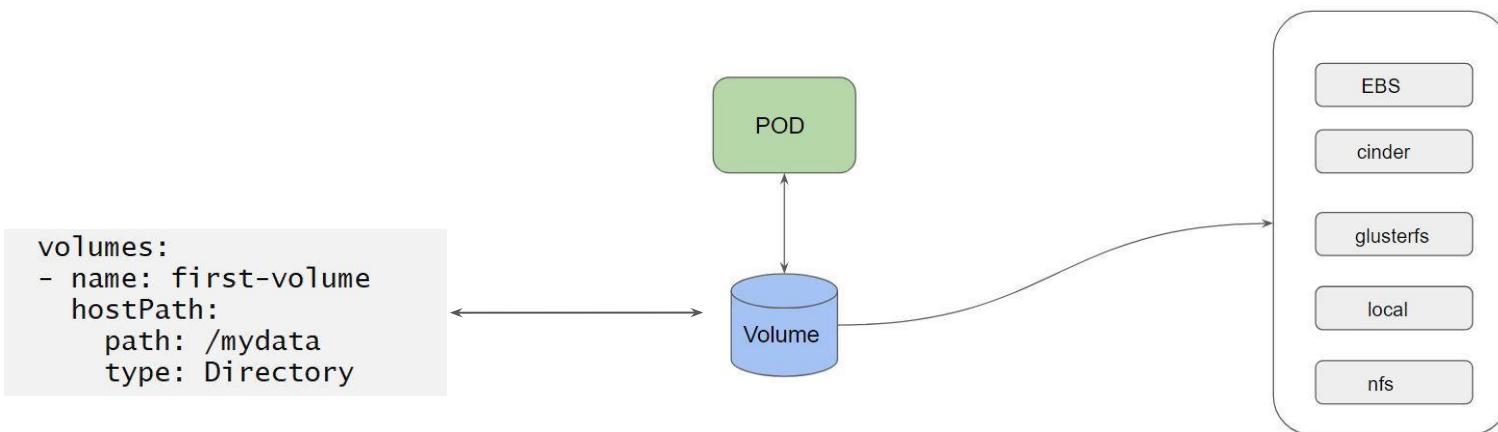
PersistentVolume and PersistentVolumeClaim

Storage Aspect

Volumes in Kubernetes

Generally developers creates the YAML files associated with pods that they want to deploy.

In a scenario of directly referencing to volumes, developer need to be aware of configurations



Understanding the Challenge



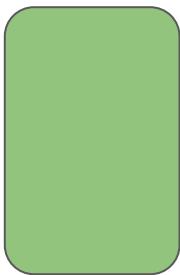
I am a Developer. I just want to write my code and pod.yaml file. I don't want to take care of storage provisioning, and its configurations.



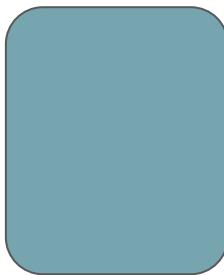
I am a Storage Administrator. I can take care of provisioning storage and its associated configurations. Developers can reference the storage within their podspec.

Persistent Volumes

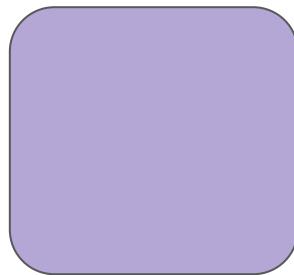
A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes



Volume 1
Size: Small
Speed: Fast



Volume 2
Size: Medium
Speed: Fast



Volume 3
Size: Big
Speed: Slow



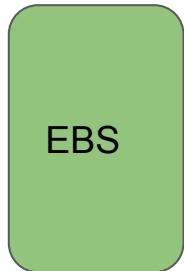
Volume 4
Size: VSmall
Speed: Ultra Fast



Volume 5
Size: Very Big
Speed: Very Slow

Different Types of Persistent Volumes

- Every Volume which is created can be of different type.
- This can be taken care by the Storage Administrator / Ops Team



Volume 1
Size: Small
Speed: Fast



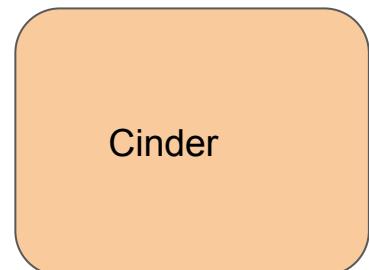
Volume 2
Size: Medium
Speed: Fast



Volume 3
Size: Big
Speed: Slow



Volume 4
Size: VSmall
Speed: Ultra Fast



Volume 5
Size: Very Big
Speed: Very Slow

PersistentVolumeClaim

A PersistentVolumeClaim is a request for the storage by a user.

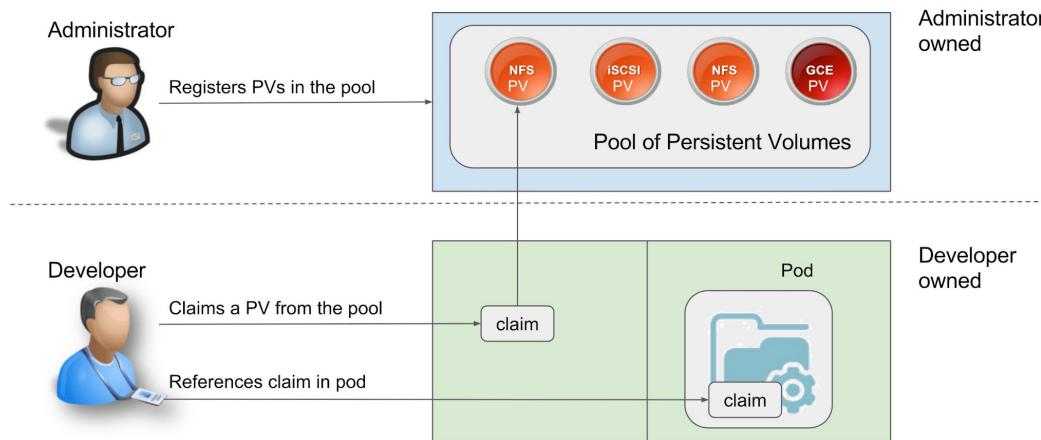
Within the claim, user need to specify the size of the volume along with access mode.

Developer:

I want a volume of size 10 GB which is has speed of Fast for my pod.

Overall Working Steps

1. Storage Administrator takes care of creating PV.
2. Developer can raise a “Claim” (I want a specific type of PV).
3. Reference that claim within the PodSpec file.

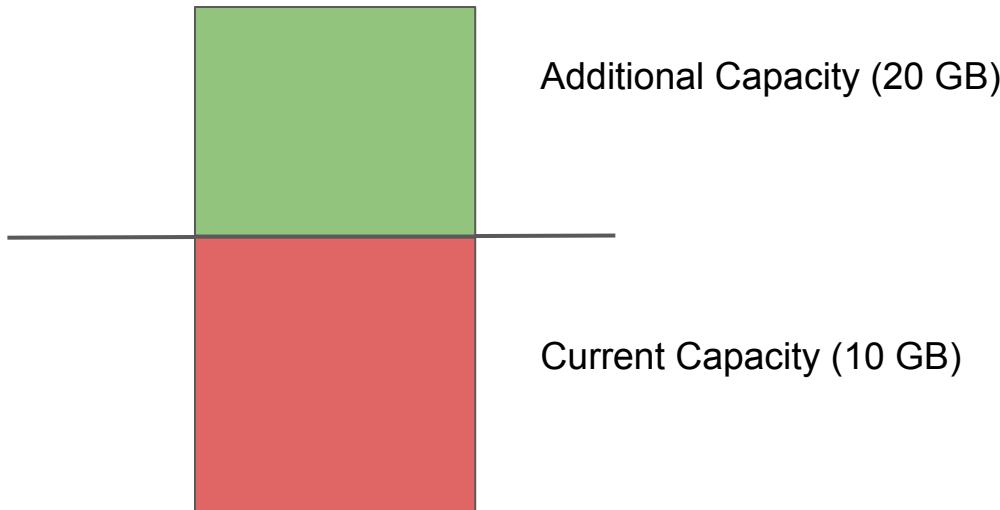


Volume Expansion in K8s

Mastering K8s

Understanding the Challenge

It can happen that your persistent volume has become full and you need to expand the storage for additional capacity.



Step 1 - Enable Volume Expansion

1. Enabling Volume Expansion in the Storage Class.

```
bash-4.2# kubectl describe storageclass do-block-storage
Name:           do-block-storage
IsDefaultClass: Yes
Annotations:   kubectl.kubernetes.io/last-applied-configuration={"allowVolumeExpansion":true,"apiVersion":"storage.k
8s.io/v1","kind":"StorageClass","metadata":{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"},"name
":"do-block-storage"},"provisioner":"dobs.csi.digitalocean.com","reclaimPolicy":"Delete"}
               ,storageclass.kubernetes.io/is-default-class=true
Provisioner:    dobs.csi.digitalocean.com
Parameters:     <none>
AllowVolumeExpansion: True
MountOptions:   <none>
ReclaimPolicy: Delete
VolumeBindingMode: Immediate
Events:        <none>
```

Step 2 - Resizing the PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"PersistentVolumeClaim",
       "metadata":{"name":"csi-pvc","namespace":"default","uid":2ec4833a-bdb7-49b1-8c32-6b73aaafea04},
       "spec":{"accessModes":["ReadWriteOnce"],"resources":{"requests":{"storage":15Gi}}}}
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: dobs.
  creationTimestamp: "2020-08-30T07:08:20Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: csi-pvc
  namespace: default
  resourceVersion: "2154"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/csi-pvc
  uid: 2ec4833a-bdb7-49b1-8c32-6b73aaafea04
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 15Gi
```

Step 3 - Restart the POD

Once PVC object is modified, you will have to restart the POD.

```
kubectl delete pod [POD-NAME]
```

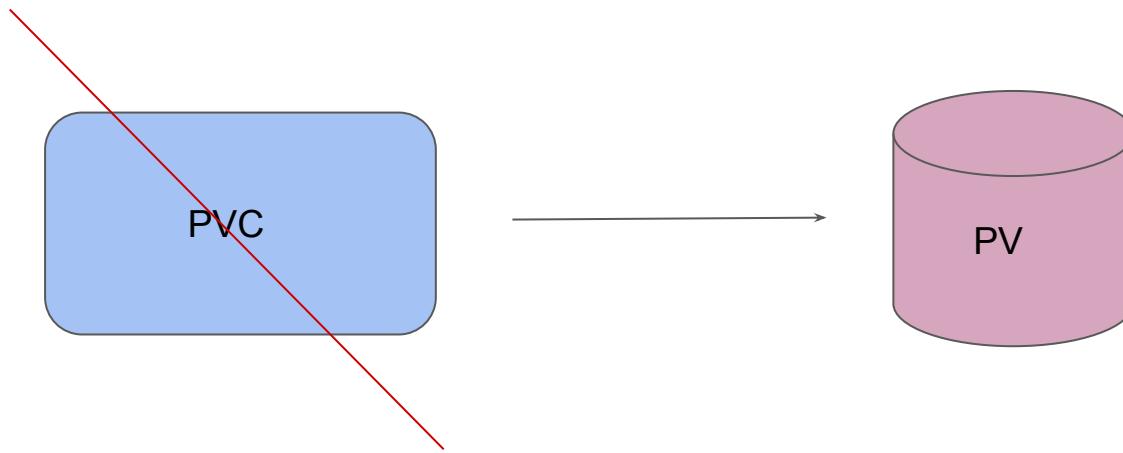
```
kubectl apply -f pod-manifest.yaml
```

Reclaim Policy

Mastering K8s Storage

Understanding the Challenge

The reclaim policy is responsible for what happens to the data in persistent volume when the kubernetes persistent volume claim has been deleted.



Types of Reclaim Policy

Reclaim Policy	Description
Retain	When the PersistentVolumeClaim is deleted, the PersistentVolume still exists and the volume is considered "released"
Delete	The persistent volume is deleted when the claim is deleted.
Recycle	If supported by the underlying volume plugin, the Recycle reclaim policy performs a basic scrub (<code>rm -rf /thevolume/*</code>) on the volume and makes it available again for a new claim.

Retain Policy

When PVC is deleted, the PersistentVolume still exists and the volume is considered "released".

It is not yet available for another claim because the previous claimant's data remains on the volume.

bash-4.2# kubectl get pv		CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
kNAME	STORAGECLASS	REASON	AGE			
pvc-41872ac3-55b7-412c-ac2c-a44e0a04fe33		5Gi	RWO	Retain	Released	default/kplabs-pvc

Steps for Reclamation

An administrator can manually reclaim the volume with the following steps.

Delete the PersistentVolume. The associated storage asset in external infrastructure (such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume) still exists after the PV is deleted.

Manually clean up the data on the associated storage asset accordingly.

Manually delete the associated storage asset, or if you want to reuse the same storage asset, create a new PersistentVolume with the storage asset definition.

S3 Storage Classes

Cloud Storage is Saviour

Use-Case - Netflix

Netflix offers various different subscription plans for various category of requirements.

Main Aim: Watch the Entertainment Content



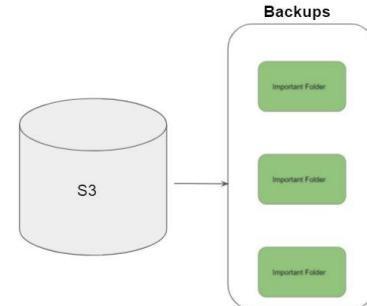
	Basic	Standard	Premium
Monthly cost* (Arab Emirates Dirham)	29 AED	39 AED	56 AED
Number of screens you can watch on at the same time	1	2	4
Number of phones or tablets you can have downloads on	1	2	4
Unlimited movies and TV shows	✓	✓	✓
Watch on your laptop, TV, phone or tablet	✓	✓	✓
HD available		✓	✓
Ultra HD available			✓

Initial Challenge - S3

AWS has millions of active customers.

Each customer might have different requirements for data storage.

Main Aim: Store Data.



S3 Storage Classes

Amazon S3 offers a range of storage classes designed for different use cases.

Storage Classes	Description
S3 Standard	Offers high durability, availability, and performance object storage for frequently accessed data
S3 Standard-Infrequent Access	For data that is accessed less frequently, but requires rapid access when needed.
Amazon S3 Glacier	Low-cost storage class for data archiving

AWS S3 Standard

S3 Standard offers high durability, availability, and performance object storage for frequently accessed data.

Designed for durability of 99.99999999% of objects (eleven nines)

Example :-

If we have 10,000 files stored in S3 (11 nines durability) then you can expect to lose one file every ten million years.

AWS S3 Standard IA

S3 Standard-IA is for data that is accessed less frequently, but requires rapid access when needed.

Comparing storage cost of 1TB data stored in S3 based on accessibility patterns.

Criteria	Amazon S3	Amazon S3 IA
Storage of 1TB Data	\$23.44	\$23.50
50% storage accessed in last 30 days	-	\$18.18
0% storage accessed in last 30 days	-	\$12.80

Amazon S3 Glacier

Glacier is meant to be for archiving and for storing long-term backups.

Ideally meant for data that needs to be archived for years without much requirement of access.

Criteria	Amazon S3	Glacier
Storage of 1TB Data	\$23.44	\$4.10

Multiple S3 Storage Classes

Performance across the S3 Storage Classes

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)					
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days

Durability vs Availability

- Durability is percent (%) over one year period of time that the file which is stored in S3 will not be lost.
- Availability is percent (%) over one year period of time that the file stored in S3 will not be available.

Example :-

For Servers, Availability is one of the key metric and any minute of downtime is a loss.
However what happens if component of server itself fails and server goes down ?

Overview of DCA Exam

Exam Experience & Important Tips

Overview of DCA Exam

DCA Exam is remotely proctored on your **Windows** or **Mac** computer

55 multiple choice questions in 90 minutes

USD \$195 or Euro €175 purchased online

Results are delivered immediately

Keep your ID card ready and be alone in the room.

DCA Initial Results



Docker Certified Associate

Examinee Details

First Name Zeal

Last Name Vora

Email sunzealvora@gmail.com

Company

Congratulations, you passed this exam

Docker Certified Associate

Overview of Imp Pointers for Exams

Exam Experience & Important Tips

Getting Started

The exam questions are generally small and you have more than sufficient time to complete the exam.

Certain questions can be tricky, so you should know how things work.

Why this section?

All the topics of this course are important for the exams.

The purpose of this section is to not skip the topics of course and directly jump to this section and sit for the exams.

The purpose is to make sure that your concepts are clear with 100% surity about the topics discussed henceforth.

Important Aspect to this section

1 topic can span across multiple domains.

In-case if that topic is not mentioned in Domain 1, it would be mentioned in other domain.

It is challenging to regularly update entire videos when a new kind of topic is asked in exam.

There is document called as “Updated Pointers” which has list of all such topics.

Make sure to complete the “Exam Preparation Quiz”

DCA Initial Results



Docker Certified Associate

Examinee Details

First Name Zeal

Last Name Vora

Email sunzealvora@gmail.com

Company

Congratulations, you passed this exam

Docker Certified Associate

Overview of DOMC Questions

Exam Experience & Important Tips

Overview of Exam Question Format

New exam question format is based on both DOMC and Multiple Choice

Type of Questions	Number of Questions
DOMC	42
Multiple Choice	13

DOMC Questions

MCQ

Overview of DOMC

DOMC stands for discrete option multiple choice.

Variations of DOMC

- Presented with only a single option, correct or incorrect, and then scoring the test item.
- Requiring more than one YES response to correct options.
- Presenting one or more non-scored options after the item has been scored.
- Increasing the number of correct and incorrect options

Approaching DOMC

If your concept of topic is clear, the DOMC questions would be easy to answer.

Make habit of calculating the answer of a question before reading any options from multiple-choice.

Example:

ADD Instruction provides which additional features over COPY?

Testing Against DOMC

Not so good news!

The DOMC item type is patented, and a license is needed to use the item in a quiz, test or exam.

Follow the tip that was previously suggested.

You cannot modify answers of DOMC questions (immutable)

Important Pointer - Part 1

Exam Experience & Important Tips

Important Pointers - Part 01

1. You should know how to create swarm Service

- docker service create --name webserver --replicas 1 nginx

2. Difference between replicated and global service

- Replicated service will have N number of containers defined with the --replica flag
- Global service will create 1 container in every node of the cluster.

Important Pointers - Part 02

3. Multiple approaches to scale swarm service

- docker service scale mywebserver=5
- docker service update --replicas 5 mywebserver

4. Difference between two approaches to scale swarm service

- docker service scale allows us to specify multiple services in same command.
- docker service update command only allows us to specify one service per command.

Important Pointers - Part 03

5. Draining Swarm Node

- docker node update --availability drain swarm03
- docker node update --availability active swarm03

6. Understanding the use-case of Docker Stack CLI

- docker stack deploy --compose-file docker-compose.yml

Important Pointers - Part 04

7. Placement Constraints

```
docker service create --name webserver --constraint node.label.region==blr nginx  
docker service create --name webserver --constraint node.label.region!=blr nginx
```

8. Adding Labels to a node

```
docker node update --label-add region=mumbai worker-node-id
```

Important Pointers - Part 05

9. Overlay Networks & Security

Overlay network allows containers spreaded across different servers to communicate.

The communication between containers can be made secured with IPSec tunnels.

```
docker network create --opt encrypted --driver overlay my-overlay-secure-network
```

Important Pointers - Part 06

10. Revise and Double Revise the Quorum

Cluster Size	Majority	Fault Tolerance
1	0	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

Important Pointers - Part 07

11. Troubleshooting aspect

- docker system events provides you real-time even information from your server.
- Service Deployment would be in pending state if node is drained.
- Service deployments can also be in pending state due to placement constraints.
- You can further inspect the task to see more information.

Important Pointers 8 - Docker Stack Commands

Child Commands	Description
<code>docker stack deploy</code>	Deploy a new stack or update an existing stack
<code>docker stack ls</code>	List stacks
<code>docker stack ps</code>	List the tasks in the stack
<code>docker stack services</code>	List the services in the stack

Important Pointers 9 - Docker Service Commands

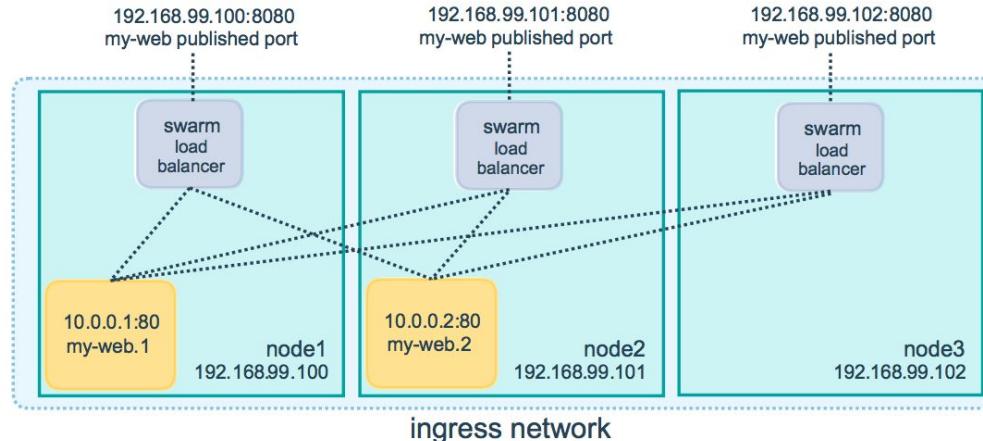
Child Commands	Description
docker service create	Create a new service
docker service inspect	Display detailed information on one or more services
docker service ps	List the tasks of one or more services
docker service update	Update a service
docker service scale	Scale one or multiple replicated services

Important Pointers 10 - Swarm Routing Mesh

Routing mesh enables each node in the swarm to accept connections on published ports for any service running in the swarm, even if there's no task running on the node.

```
docker service create --name my_web --replicas 3 --publish published=8080,target=80 nginx
```

8080 goes to 80



Important Pointers 11 - Updating Image of Swarm Service

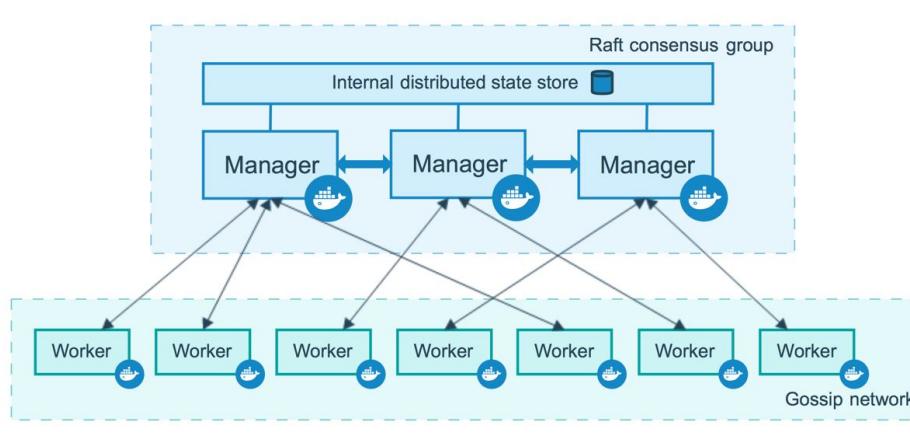
To update an image of a swarm service, following command can be used:

```
docker swarm update --image myimage:tag servicename
```

Important Pointers 12 - Manager and Worker Nodes

- Manager Nodes handles cluster management tasks like scheduling services.
- Raft implementation is used to maintain consistent internal state.
- Sole purpose of worker nodes is to execute containers.

To prevent the scheduler from placing tasks on a manager nodeset the availability for the manager node to Drain



Important Pointers 13 - Join Tokens in Swarm

Join tokens are secrets that allow a node to join the swarm.

There are two different join tokens available, one for the worker role and one for the manager role.

```
docker swarm join-token worker
```

Important Pointers 14 - Initializing Swarm

Initialize a swarm. The docker engine targeted by this command becomes a manager in the newly created single-node swarm.

```
docker swarm init
```

Important Pointers 15 - Miscellaneous Swarm

Leaving Docker Swarm Cluster:

If a worker intends to leave swarm, he can run the following command:

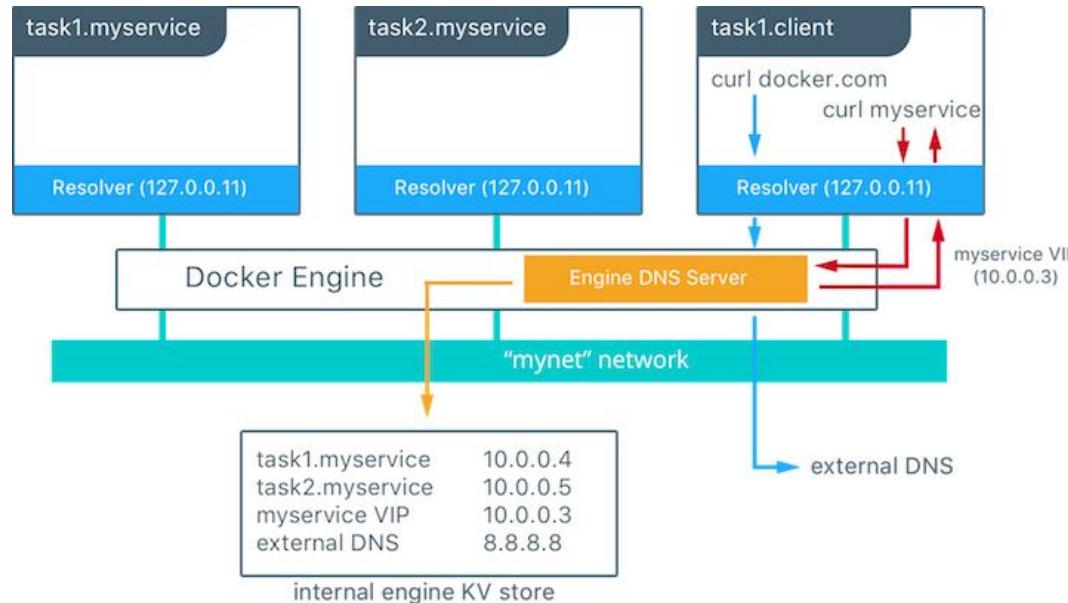
```
docker swarm leave
```

You can use the --force option on a manager to remove it from the swarm.

Only use --force in situations where the swarm will no longer be used after the manager leaves, such as in a single-node swarm.

Important Pointers 16 - Service Discovery in Swarm

Docker uses embedded DNS to provide service discovery for containers running on a single Docker engine and tasks running in a Docker swarm



Miscellaneous Pointer - 1

The docker system df command displays information regarding the amount of disk space used by the docker daemon.

```
C:\Users\Zeal Vora>docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	15	7	6.092GB	2.575GB (42%)
Containers	18	1	23.21GB	18.33GB (78%)
Local Volumes	88	3	32.09GB	30.85GB (96%)
Build Cache	0	0	0B	0B

Miscellaneous Pointer - 2

Use docker system events to get real-time events from the server. These events differ per Docker object type.

```
$ docker system events --filter 'event=stop'

2017-01-05T00:40:22.880175420+08:00 container stop 0fdb...ff37 (image=alpine:latest, name=test)
2017-01-05T00:41:17.888104182+08:00 container stop 2a8f...4e78 (image=alpine, name=kickass_brattain)

$ docker system events --filter 'image=alpine'

2017-01-05T00:41:55.784240236+08:00 container create d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:41:55.913156783+08:00 container start d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:42:01.106875249+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=15)
2017-01-05T00:42:11.111934041+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=9)
2017-01-05T00:42:11.119578204+08:00 container die d9cd...4d70 (exitCode=137, image=alpine, name=happy_meitner)
2017-01-05T00:42:11.173276611+08:00 container stop d9cd...4d70 (image=alpine, name=happy_meitner)
```

Miscellaneous Pointer - 3

dockerd is the persistent process that manages containers. Docker uses different binaries for the daemon and client. To run the daemon you type dockerd

For specifying the configuration options, you can make use of daemon.json file.

The default location of the configuration file on Linux is /etc/docker/daemon.json

The default location of the configuration file on Windows is
%programdata%\docker\config\daemon.json

Miscellaneous Pointer - 4

`docker container inspect` display detailed information on one or more containers.

It can also show you list of volumes that are attached to the container.

```
"Mounts": [
  {
    "Type": "bind",
    "Source": "/host_mnt/d/containers/git",
    "Destination": "/gitplace",
    "Mode": "",
    "RW": true,
    "Propagation": "rprivate"
  }
],
```

Miscellaneous Pointer 5 - Inspecting Container

`docker container inspect` display detailed information on one or more containers.

It can also show you list of volumes that are attached to the container.

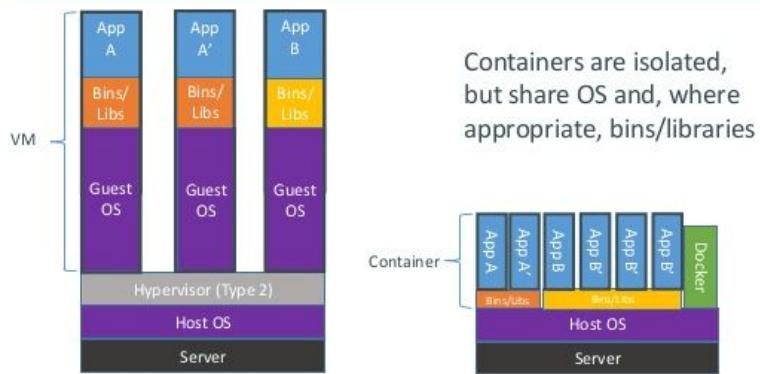
```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/host_mnt/d/containers/git",
        "Destination": "/gitplace",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

Containers vs Virtual Machines

Virtual Machine contains entire Operating System.

Container uses the resource of the host operating system (primarily the kernel)

Containers vs. VMs



Important Pointer - Part 2

Image Creation, Management, and Registry

Important Pointers - Part 01

1. Have thorough understanding about various Dockerfile instructions.

- ADD
- COPY
- RUN
- ENTRYPOINT
- WORKDIR
- ENV
- VOLUMES
- CMD
- HEALTHCHECK

Important Pointers - Part 02

2. Understand difference between ADD and COPY

COPY allows us to copy files from local source to destination.

ADD allows the same in addition to using URL & extraction capabilities of TAR files.

3. Know difference between CMD and ENTRYPOINT

- CMD can be overridden.
- ENTRYPOINT cannot be overridden.

Important Pointers - Part 03

4. Use-cases of using ADD vs wget/curl

- Because image size matters, using ADD to fetch packages from remote URLs is strongly discouraged; you should use curl or wget instead.

```
ADD http://example.com/big.tar.xz /usr/src/things/  
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things  
RUN make -C /usr/src/things all
```

```
RUN mkdir -p /usr/src/things \  
    && curl -SL http://example.com/big.tar.xz \  
    | tar -xJC /usr/src/things \  
    && make -C /usr/src/things all
```

Important Pointers 04 - WORKDIR

The **WORKDIR** instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile

The WORKDIR instruction can be used multiple times in a Dockerfile

Sample Snippet:

```
WORKDIR /a  
WORKDIR b  
WORKDIR c  
RUN pwd
```

Output = /a/b/c

Important Pointers - Part 05

5. Understanding the format option

Format option allows us to format the output based on various criteria that we have defined with the command

```
[root@ip-172-31-45-184 ~]# docker images --format "{{.ID}}: {{.Repository}}"
87159635da2d: aml-ssh
d131e0fa2585: ubuntu
eaa8f22fecc5: registry.aquasec.com/enforcer
27a188018e18: nginx
af2f74c517aa: busybox
01da4f8f9748: amazonlinux
```

Important Pointers - Part 06

6. Understanding the filter option

Filter option allows us to filter output based on condition provided.

Sample Use-Case:

- Show all the dangling images
- Show all the images created after N image.
- Show all the containers which are in the exited stage.

Important Pointers - Part 07

7. Accessing in-secure docker registries

By default, docker will not allow you to perform operation with an insecure registry. You can override by adding following stanza within the /etc/docker/daemon.json file

```
{  
  "insecure-registries" : ["myregistrydomain.com:5000"]  
}
```

Important Pointers - Part 08

8. Pushing an image to a private repository

In order to push an image to private repository, you will have to tag it with the DNS name.

For example:

Registry Name: example.com

```
docker tag ubuntu:latest example.com/myrepo:ubuntu
```

Important Pointers - Part 09

9. Login to a private repository

docker login example.com

10. Searchability Aspect of Images

- docker search nginx
- docker search nginx --filter “is-official=true”

Important Pointer 10 - Moving Images Across Hosts

The docker save command will save one or more images to a tar archive

```
docker save busybox > busybox.tar
```

The docker load command will load an image from a tar archive

```
docker load < busybox.tar
```

Important Pointer 11 - Container Management

The docker commit can be used to save the current state of a container to an image.

```
docker commit c3f279d17e0a container-image:v2
```

You can use docker export command to export a container to another system as an image tar file.

```
docker export my-container > container.tar
```

When we export a container, all the resultant imported image will be flattened.

Important Pointer 12 - Dockerfile ENV

The ENV instruction sets the environment variable <key> to the value <value>.

```
ENV NGINX 1.2
RUN curl -SL http://example.com/web-\$NGINX.tar.xz
RUN tar -xzvf web-$NGINX.tar.xz
```

You can use the -e, --env, and --env-file flags to set simple environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.

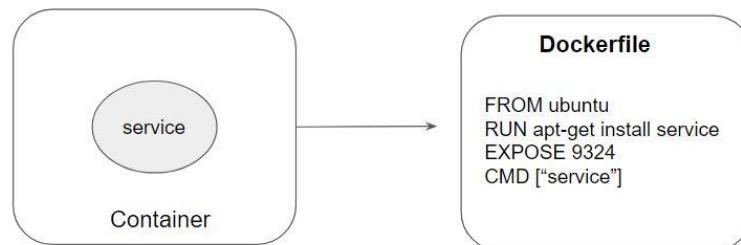
```
docker run --env VAR1=value1 --env VAR2=value2 ubuntu env | grep VAR
```

Important Pointer 13 - EXPOSE

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

The EXPOSE instruction does not actually publish the port.

It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.



Important Pointer 14 - HEALTHCHECK

HEALTHCHECK instruction in Docker allows us to tell the platform on how to test that our application is healthy.

```
HEALTHCHECK CMD curl --fail http://localhost || exit 1
```

That uses the curl command to make an HTTP request inside the container, which checks that the web app in the container does respond.

It exits with a 0 if the response is good, or a 1 if not - which tells Docker the container is unhealthy.

Important Pointer 15 - Tagging Docker Images

Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Syntax:

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

Example Snippet:

```
docker tag httpd fedora/httpd:version1.0
```

Important Pointer 16 - Pruning Docker Images

`docker image prune` command allows us to clean up unused images.

By default, the above command will only clean up dangling images.

Dangling Images = Image without Tags and Image not referenced by any container

Downloading Images from Private Registry

If your image is available on a private registry which requires login, use the `--with-registry-auth` flag with `docker service create`, after logging in.

```
docker login registry.example.com
```

```
docker service create --with-registry-auth \--name my_service  
registry.example.com/acme/my_image:latest
```

This passes the login token from your local client to the swarm nodes where the service is deployed, using the encrypted WAL logs. With this information, the nodes are able to log into the registry and pull the image.

Important Pointer 18 - Build Cache

Docker creates container images using layers.

Each command that is found in a Dockerfile creates a new layer.

Docker uses a layer cache to optimize the process of building Docker images and make it faster.

```
[root@swarm02 build-cache]# docker build -t demo2
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM python:3.7-slim-buster
--> 87b1022604d5
Step 2/4 : COPY . .
--> Using cache
--> fe355c27a8ff
```

Important Pointer - Domain 3

Installation & Configuration

Important Pointers 01 - DTR Backup Process

When we backup DTR, there are certain things which are not backed.

User/Organization backup should be taken from UCP.

Data	Description	Backed up
Raft keys	Used to encrypt communication among Swarm nodes and to encrypt and decrypt Raft logs	yes
membership	List of the nodes in the cluster	yes
Services	Stacks and services stored in Swarm-mode	yes
(overlay) networks	The overlay networks created on the cluster	yes
configs	The configs created in the cluster	yes
secrets	Secrets saved in the cluster	yes
Swarm unlock key	Must be saved on a password manager !	no

Important Pointers 02 - Namespaces

2. Know what namespaces are all about.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

User namespace is not enabled by default

Important Pointers 03 - Cgroups

Have clear understanding about cgroups.

Control Groups (cgroups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

Approaches	Description
--cpus=<value>	If host has 2 CPUs and if you set --cpus=1, than container is guaranteed at most one CPU. You can even specify --cpus=0.5
--cpuset-cpus	Limit the specific CPUs or cores a container can use.A comma-separated list or hyphen-separated range of CPUs a container can use. 1st CPU is numbered 0 2nd CPU is numbered 1. Value of 0-3 means usage of first, second, third and fourth CPU. Value of 1,3 means second and fourth CPU.

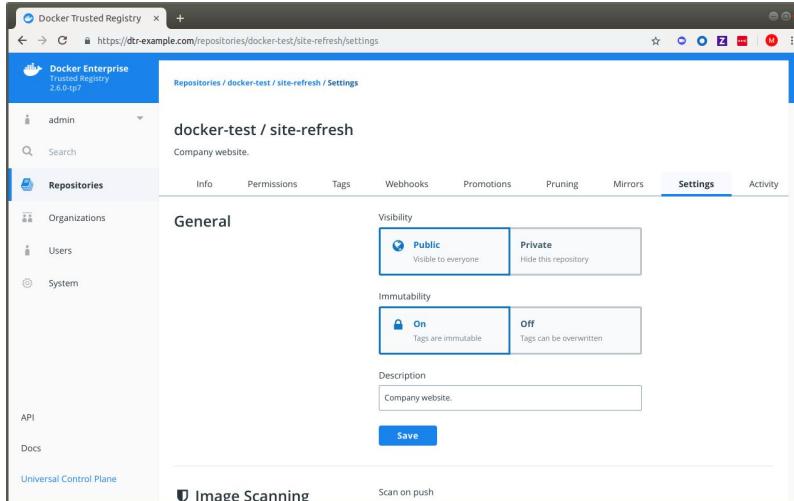
Important Pointers 04 - Reservation and Limits

- Limit is a hard limit.
 - Reservation is a soft limit.
-
- docker container run -dt --name container01 --memory-reservation 250m ubuntu
 - docker container run -dt --name container02 -m 500m ubuntu

--memory and --memory-reservation

Important Pointers 05 - Immutable Tags in DTR

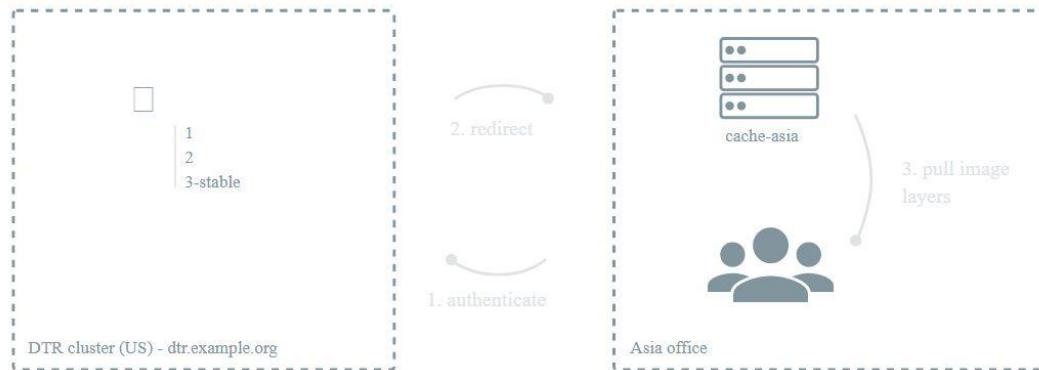
- By default, users with read and write access can overwrite tags.
- To prevent tags from being overwritten, we can configure repository to be immutable.



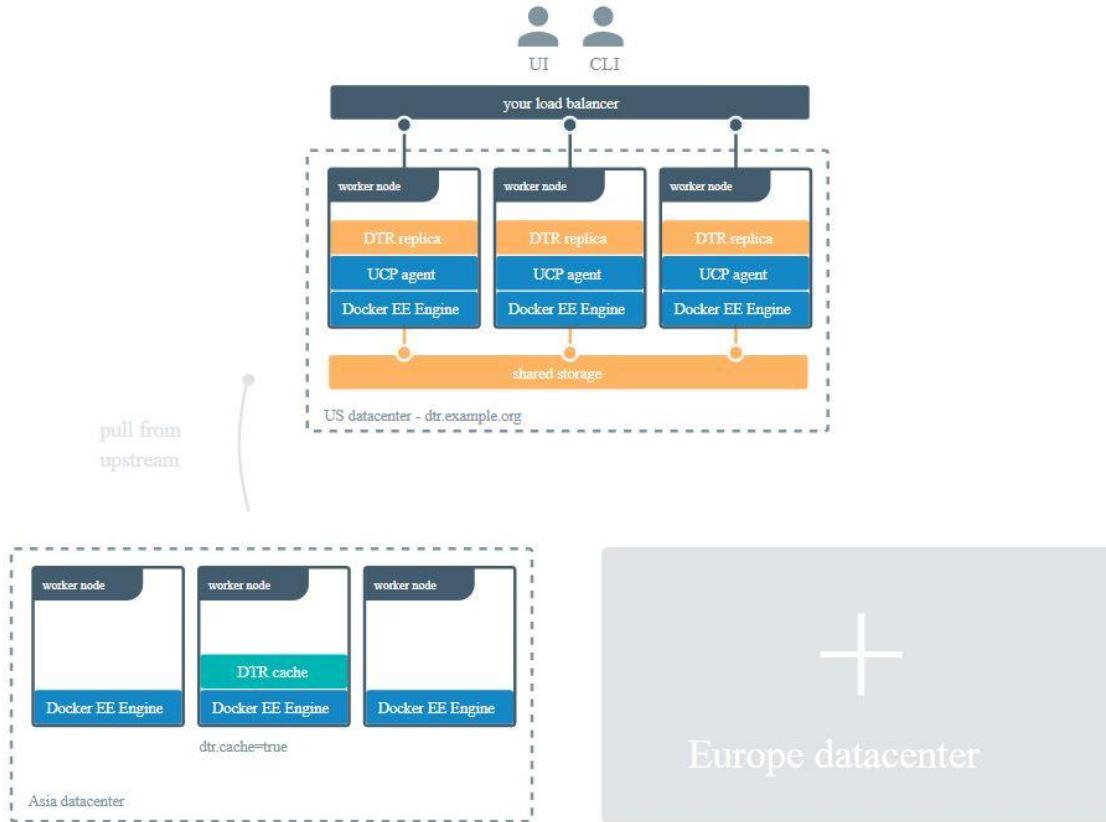
Important Pointers 06 - DTR Cache

To decrease the time to pull an image, you can deploy DTR caches geographically closer to users.

Caches are transparent to users, since users still log in and pull images using the DTR URL address. DTR checks if users are authorized to pull the image, and redirects the request to the cache.



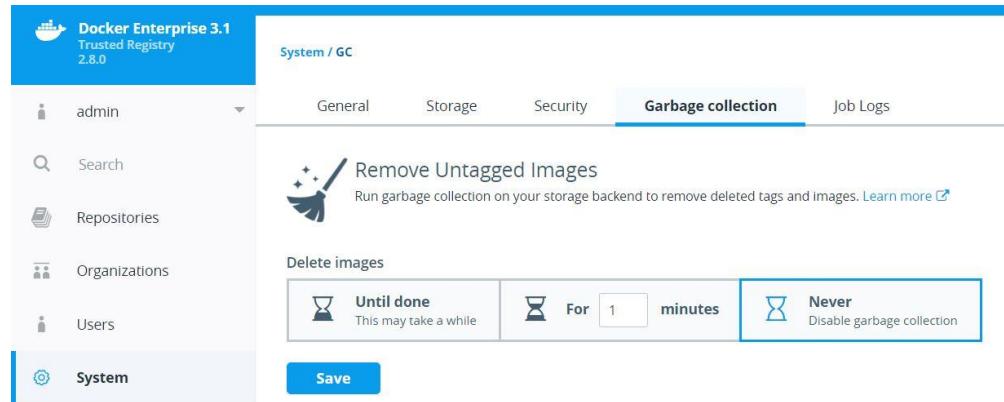
DTR Cache Architecture



DTR Garbage Collection

You can configure the Docker Trusted Registry (DTR) to automatically delete unused image layers, thus saving you disk space.

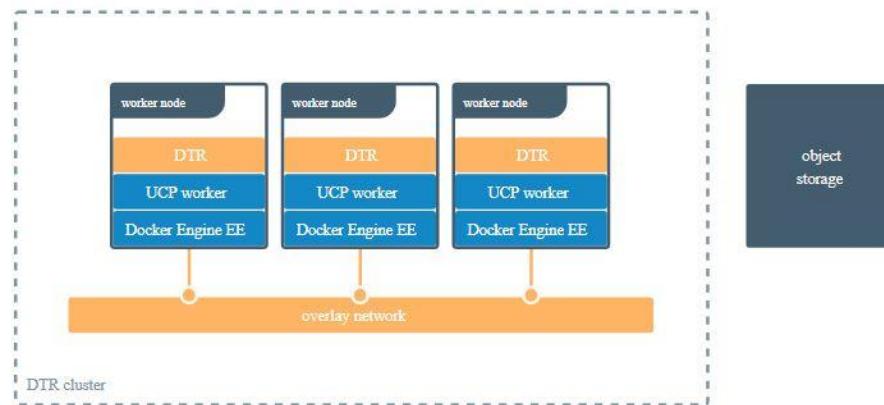
This process is also known as garbage collection.



Important Pointers 07 - DTR High Availability

Docker Trusted Registry is designed to scale horizontally as your usage increases. You can add more replicas to make DTR scale to your demand and for high availability.

If your DTR deployment has multiple replicas, for high availability, you need to ensure all replicas are using the same storage backend.



Important Pointers 08 - HA of UCP and DTR

To have high-availability on UCP and DTR, you need a minimum of:

- 3 dedicated nodes to install UCP with high availability,
- 3 dedicated nodes to install DTR with high availability,

You can monitor the status of UCP by using the web UI or the CLI. You can also use the [_ping](#) endpoint to build monitoring automation.

Important Pointers 09 - Orchestrator Types in UCP

Docker UCP supports both Swarm and Kubernetes.

When you install Docker Enterprise, new nodes are managed by Docker Swarm, but you can change the default orchestrator to Kubernetes in the administrator settings.

To change the orchestrator type for a node from Swarm to Kubernetes:

```
docker node update --label-add com.docker.ucp.orchestrator.kubernetes=true <node-id>
```

Scheduler

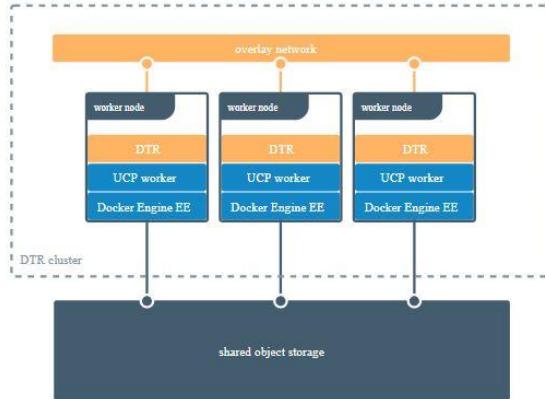
SET ORCHESTRATOR TYPE FOR NEW NODES

Swarm Kubernetes

Important Pointers 10 - Storage Driver in DTR

By default, Docker Trusted Registry stores images on the filesystem of the node where it is running, but you should configure it to use a centralized storage backend.

You can configure DTR to use an external storage backend, for improved performance or high availability.



Important Pointers 11 - Supported Storage Systems

Some of the supported storage systems in DTR are:

Local:

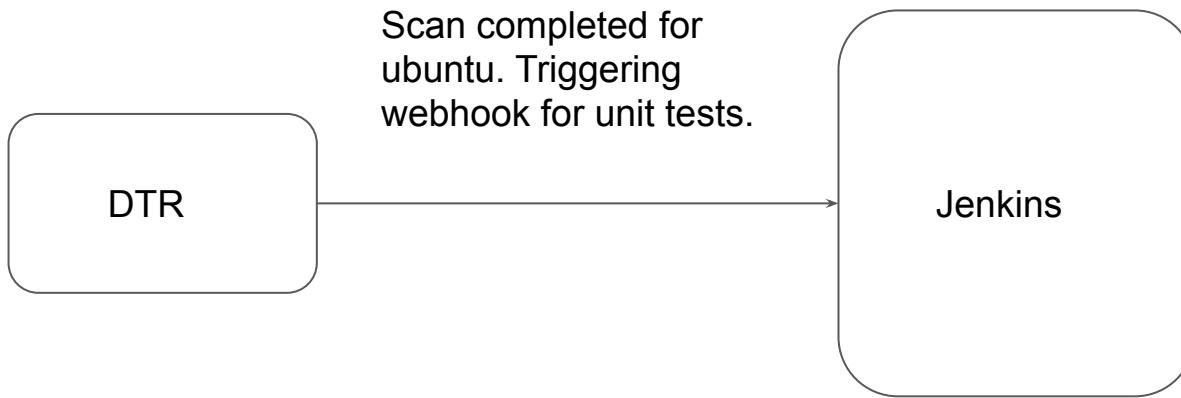
- NFS
- Bind Mount
- Volume

Cloud Storage Provider:

- AWS S3
- Azure
- Google Cloud

Important Pointers 12 - DTR Webhooks

You can configure DTR to automatically post event notifications to a webhook URL of your choosing



Important Pointer - Part 4

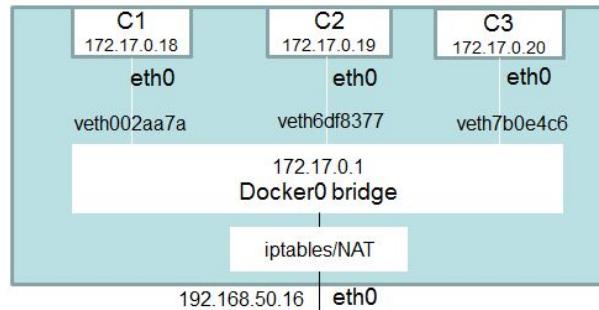
Networking

Important Pointers - Part 01

You should be aware of the primary networking drivers

1. Bridge Network Driver:

- Bridge is the default network driver for Docker.



Important Pointers - Part 02

2. Host Network

- This driver removes the network isolation between the docker host and the docker containers to use the host's networking directly.

4. User Defined Bridge and Legacy Link Approach

Remember that --link is a legacy approach and should not be used.

Important Pointers - Part 03

Overlay Network

- Default in Swarm.
- Allows containers across host to communicate with each other.
- Communication can be encrypted with `--opt encrypted` option.
- Do not confuse, `-o` is same as `--opt`

You don't need to create the overlay network on the other nodes, because it will be automatically created when one of those nodes starts running a service task which requires it.

Important Pointers - Part 04

6. Know difference between publish list (-p) and publish all (-P)

- Publish List (-p) will publish list of ports that you define. [-p 80:80]
- Publish All (-P) will assign random ports for all exposed ports of the container.
- -P will map the container port to random port above 32768

7. None Network

- If you want to completely disable the networking stack on a container, you can use the none network.
- This mode will not configure any IP for the container and doesn't have any access to the external network as well as for other containers.

Important Pointers - Part 05

8. Configuring docker for external DNS

- Docker Container's DNS configuration is taken from the host's /etc/resolv.conf
- DNS settings for containers be customized via daemon.json file.

```
{  
  "dns": ["8.8.8.8", "172.31.0.2"]  
}
```

Important Pointers 06 - Inspecting Network Details

To Fetch information about a specific network, you can run the following command:

```
docker network inspect <network-name>
```

Join us in our Adventure

Be Awesome



kplabs.in/twitter



kplabs.in/linkedin

instructors@kplabs.in

Important Pointer - Part 5

Security

Important Pointers 01 - UCP Client Bundles

- Client bundles are group of certificates and files downloaded from UCP.
- Depending on the permission associated with the user, you can now execute docker swarm commands from your remote machine that take effect on the remote cluster.

Important Pointers 02 - Docker Content Trust

Used for interacting with only trusted images (signed images)

Enabled with `export DOCKER_CONTENT_TRUST=1`

Example Dockerfile:

```
FROM myubuntu:latest
RUN apt-get install net-tools
CMD["bash"]
```

Important Pointers 02 - Docker Content Trust

Engine Signature Verification prevents the following:

- docker container run of an unsigned or altered image.
- docker pull of an unsigned or altered image.
- docker build where the FROM image is not signed or is not scratch.

Important Pointers 03 - Docker Secrets

To create a new secret, `docker secret create` command can be used.

This is a cluster management command, and must be executed on a swarm manager node.

Remember that you cannot update or rename a secret.

--secret-add and --secret-rm flags for docker service update

Important Pointers 04 - Secrets Mount Path

When you grant a newly-created or running service access to a secret, the decrypted secret is mounted into the container in following path:

/run/secrets/<secret_name>

We can also specify a custom location for the secret.

```
docker service create --name redis --secret source=mysecret,target=/root/secretdata
```

Important Pointers 05 - Swarm Auto-Lock

Know about the how you can lock the swarm cluster

If your Swarm is compromised and if data is stored in plain-text, an attack can get all the sensitive information.

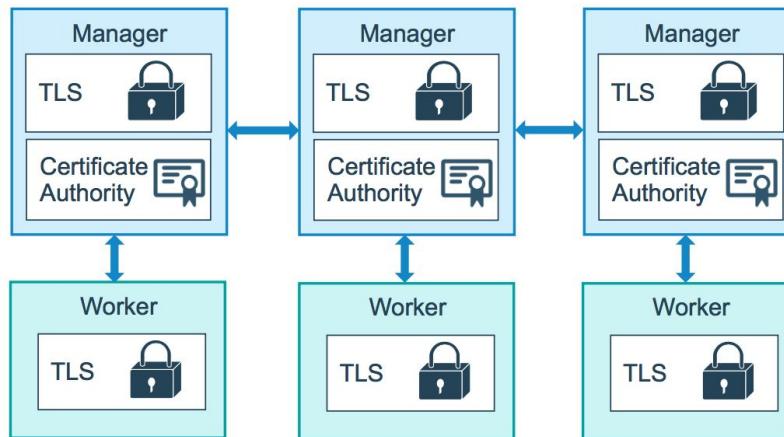
Docker Lock allows us to have control over the keys.

- docker swarm update --autolock=true

Important Pointers 06 - Mutual TLS Authentication

When you create a swarm by running docker swarm init, Docker designates itself as a manager node.

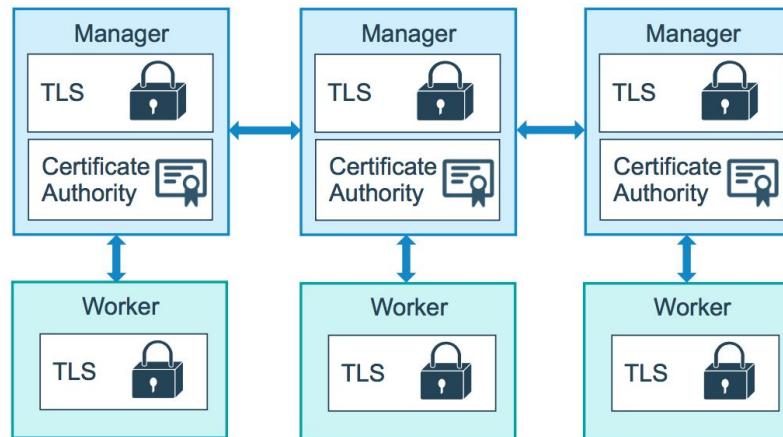
By default, the manager node generates a new root Certificate Authority (CA) along with a key pair, which are used to secure communications



Important Pointers 07 - Certificate for Each Node

Each time a new node joins the swarm, the manager issues a certificate to the node.

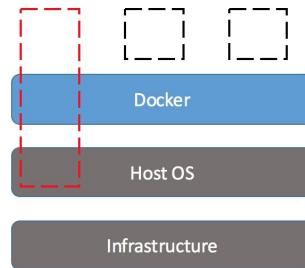
By default, each node in the swarm renews its certificate every three months.



Important Pointers 08 - Privileged container

By default, Docker containers are “unprivileged”

When the operator executes `docker run --privileged`, Docker will enable access to all devices on the host to allow the container nearly all the same access to the host as processes running outside containers on the host.



Important Pointers 09 - Roles in UCP

Built-in role	Description
None	Users have no access to Swarm or Kubernetes resources. Maps to No Access role in UCP 2.1.x.
View Only	Users can view resources but can't create them.
Restricted Control	<p>Users can view and edit resources but can't run a service or container in a way that affects the node where it's running.</p> <p>Users cannot mount a node directory, exec into containers, or run containers in privileged mode or with additional kernel capabilities</p>
Scheduler	Users can view nodes (worker and manager) and schedule (not view) workloads on these nodes. By default, all users are granted the Scheduler role against the /Shared collection
Full Control	Users can view and edit all granted resources. They can create containers without any restriction, but can't see the containers of other users.

Important Pointer - Part 6

Storage and Volumes

Important Pointers - Part 01

1. Mounting Volumes Inside Container

- docker container run -dt --name webserver -v myvolume:/etc busybox sh
- docker container run -dt --name webserver --mount source=myvolume,target=/etc busybox sh

You can even mount same volume to multiple containers.

Also know what bind mounts are all about

How you can map a directory from host to container.

Important Pointers - Part 02

2. Remove volume when container exits

If you have not named the volume, then when you specify --rm option, the volumes associated will also be deleted when container is deleted.

3. Device Mapper

- loop-lvm mode is for testing purpose only.
- direct-lvm mode can be used in production.

Join us in our Adventure

Be Awesome



kplabs.in/twitter



kplabs.in/linkedin

instructors@kplabs.in