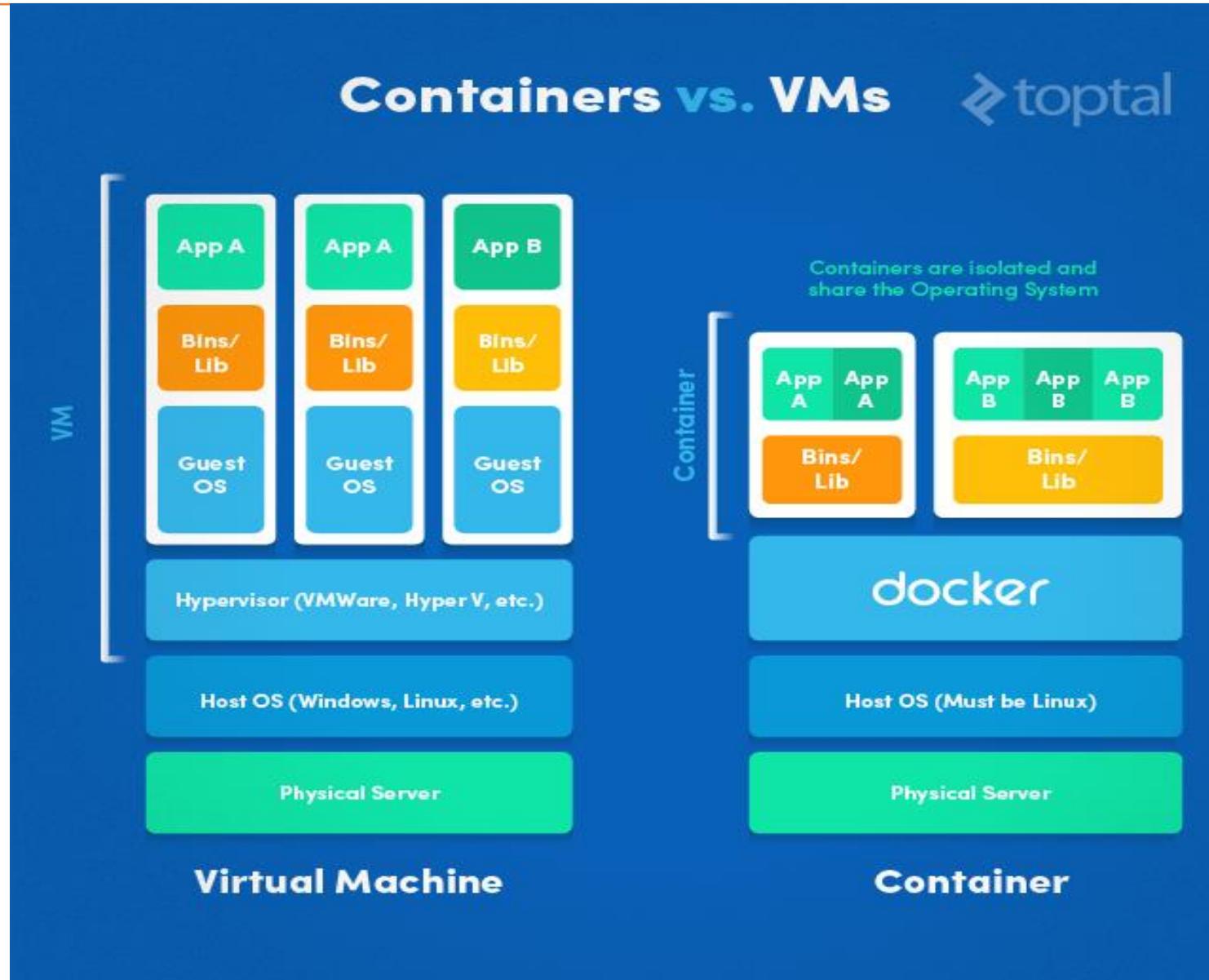




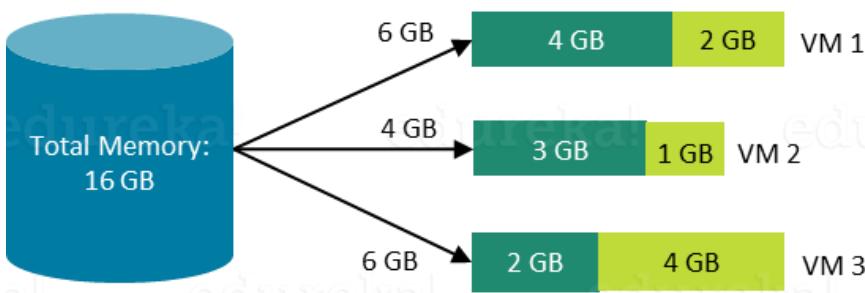
Docker

- Docker is an excellent tool for managing and deploying microservices.
- Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files.
- Combined with a provisioning tool such as Kubernetes, each microservice can then be easily deployed, scaled, and collaborated on by a developer team.
- Specifying an environment in this way also makes it easy to link microservices together to form a larger application.

Docker



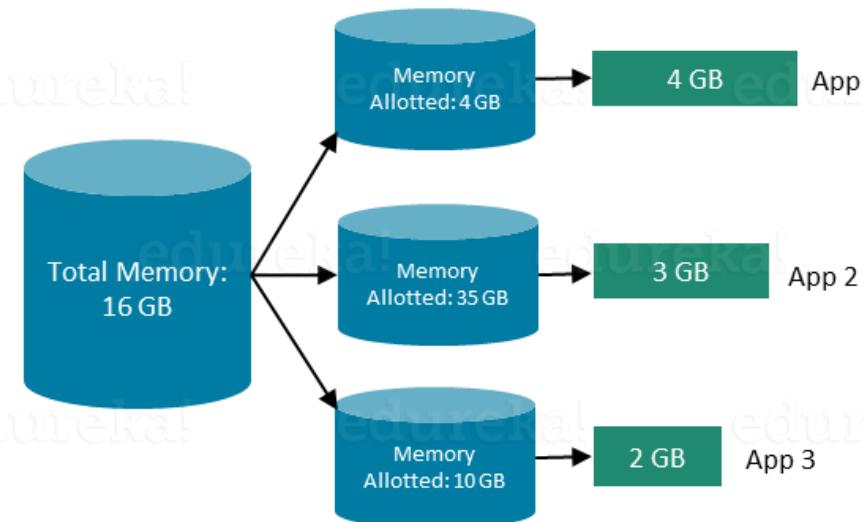
In case of Virtual Machines



- [Green Square] → Memory Used: 9 GB
- [Yellow Square] → Memory wasted: 7 GB

7 Gb of Memory is blocked and cannot be allotted to a new VM

In case of Docker



- [Green Square] → Memory Used: 9 GB

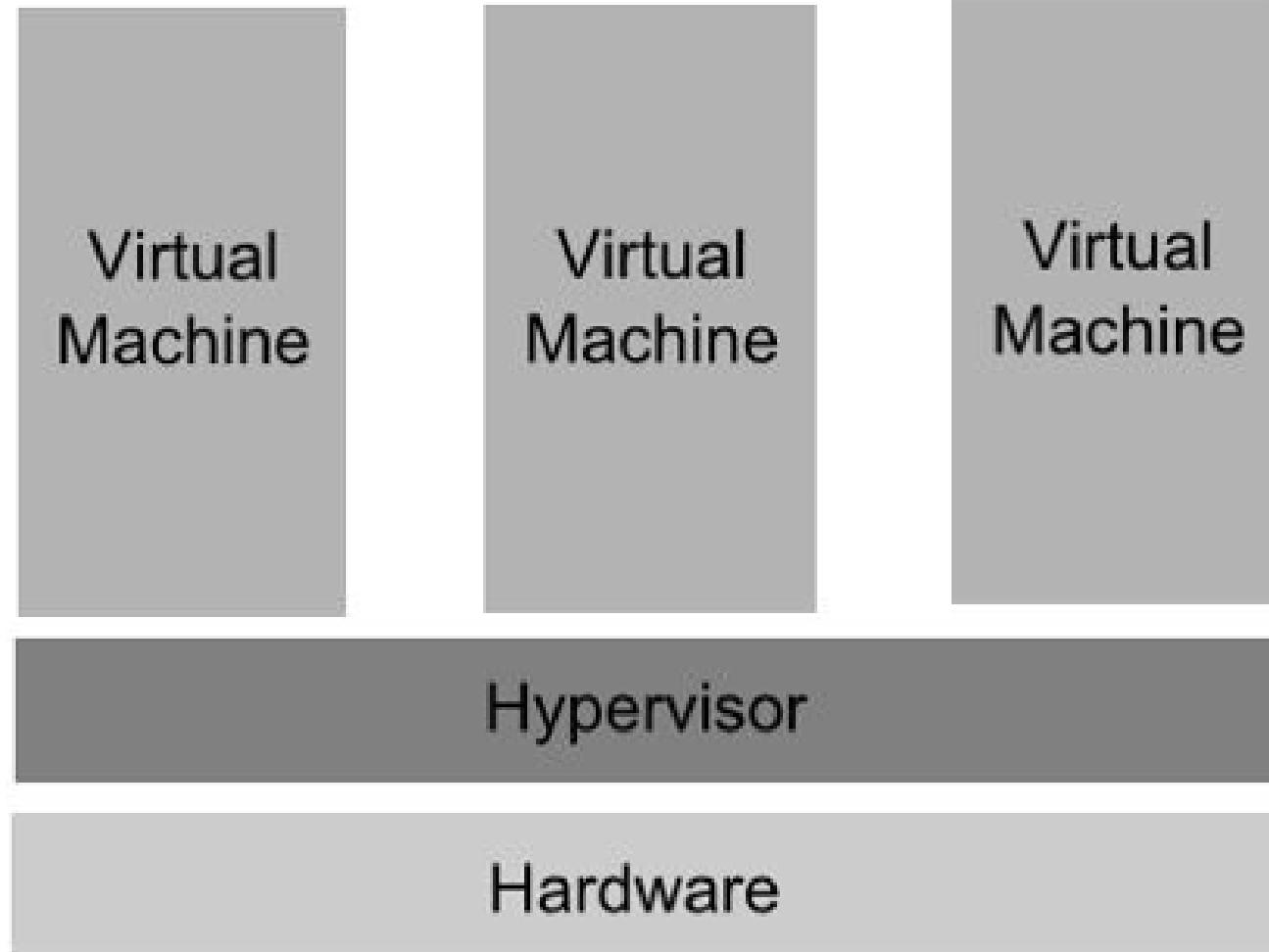
Only 9 GB memory utilized;
7 GB can be allotted to a new Container



Hypervisor

- A hypervisor is a software that makes virtualization possible.
- It is also called Virtual Machine Monitor.
- It divides the host system and allocates the resources to each divided virtual environment.

Hypervisor





Hypervisor

- Hyper-V will only run on processors which support hardware assisted virtualization.
- The x86 family of CPUs provide a range of protection levels also known as rings in which code can execute.
- Ring 0 has the highest level privilege and it is in this ring that the operating system kernel normally runs.
- Code executing in ring 0 is said to be running in system space, kernel mode or supervisor mode.
- All other code, such as applications running on the operating system, operate in less privileged rings, typically ring 3.



Hypervisor

- Under Hyper-V hypervisor virtualization a program known as a hypervisor runs directly on the hardware of the host system in ring 0.
- The task of this hypervisor is to handle tasks such CPU and memory resource allocation for the virtual machines in addition to providing interfaces for higher level administration and monitoring tools.



Hypervisor

- If the hypervisor is going to occupy ring 0 of the CPU, the kernels for any guest operating systems running on the system must run in less privileged CPU rings.
- Unfortunately, most operating system kernels are written explicitly to run in ring 0 for the simple reason that they need to perform tasks that are only available in that ring, such as the ability to execute privileged CPU instructions and directly manipulate memory.



Hypervisor

- One solution to this problem is to modify the guest operating systems, replacing any privileged operations that will only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls).
- The hypervisor in turn performs the task on behalf of the guest system.

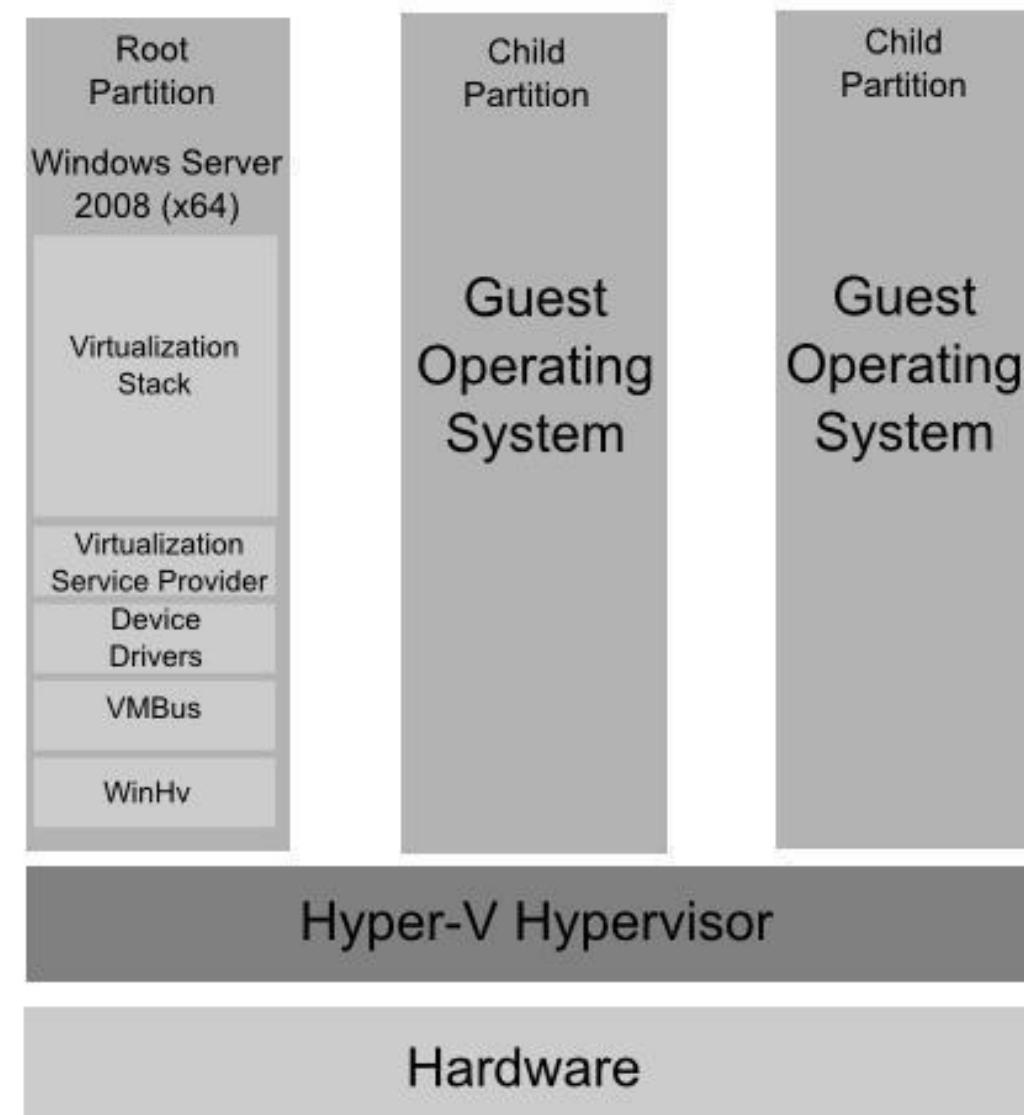


Hypervisor

- Another solution is to leverage the hardware assisted virtualization features of the latest generation of processors from both Intel and AMD.
- These technologies, known as Intel VT and AMD-V respectively, provide extensions necessary to run unmodified guest virtual machines.
- These new processors provide an additional privilege mode (referred to as ring -1) above ring 0 in which the hypervisor can operate, essentially leaving ring 0 available for unmodified guest operating systems.



Hyper-V Root and Child Partitions

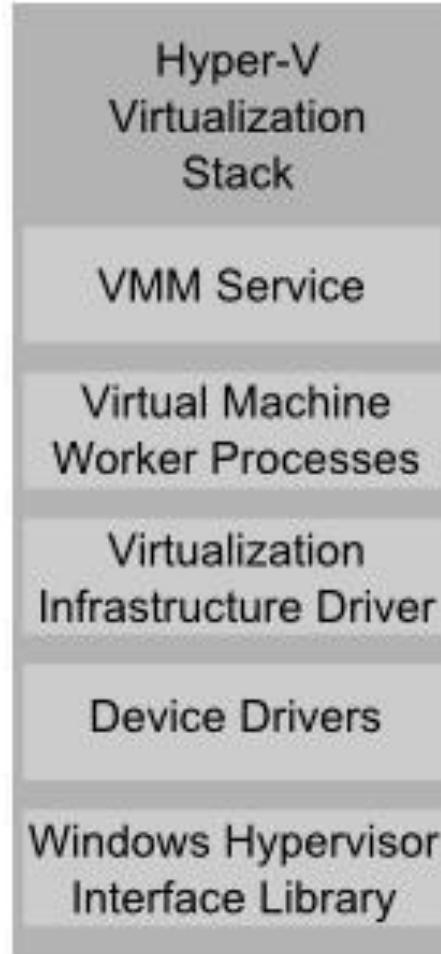




Hyper-V Root and Child Partitions

- The root partition is essentially a virtual machine which runs a copy of 64-bit Windows Server 2008 which, in turn, acts as a host for a number of special Hyper-V components.
- The root partition is responsible for providing the device drivers for the virtual machines running in the child partitions, managing the child partition lifecycles, power management and event logging.
- The root partition operating system also hosts the Virtualization Stack which is responsible for performing a wide range of virtualization functions.
- Child partitions host the virtual machines in which the guest operating systems run. Hyper-V supports both Hyper-V Aware (also referred to as enlightened) and Hyper-V Unaware guest operating systems.

The Virtualization Stack and Other Root Partition Components



The Virtualization Stack and Other Root Partition Components



Component	Description
Virtual Machine Management Service (VMM Service)	Manages the state of virtual machines running in the child partitions (active, offline, stopped etc) and controls the tasks that can be performed on a virtual machine based on current state (such as taking snapshots). Also manages the addition and removal of devices. When a virtual machine is started, the VMM Service is also responsible for creating a corresponding <i>Virtual Machine Worker Process</i> .
Virtual Machine Worker Process	Virtual Machine Worker Processes are started by the VMM Service when virtual machines are started. A Virtual Machine Worker Process (named vmwp.exe) is created for each Hyper-V virtual machine and is responsible for much of the management level interaction between the parent partition Windows Server 2008 system and the virtual machines in the child partitions. The duties of the Virtual Machine Worker Process include creating, configuring, running, pausing, resuming, saving, restoring and snapshotting the associated virtual machine. It also handles IRQs, memory and I/O port mapping through a <i>Virtual Motherboard</i> (VMB).

The Virtualization Stack and Other Root Partition Components



Component	Description
Virtual Devices	Virtual Devices are managed by the Virtual Motherboard (VMB). Virtual Motherboards are contained within the Virtual Machine Worker Processes, of which there is one for each virtual machine. Virtual Devices fall into two categories, <i>Core VDevs</i> and <i>Plug-in VDevs</i> . Core VDevs can either be <i>Emulated Devices</i> or <i>Synthetic Devices</i> .
Virtual Infrastructure Driver	Operates in kernel mode (i.e. in the privileged CPU ring) and provides partition, memory and processor management for the virtual machines running in the child partitions. The Virtual Infrastructure Driver (Vid.sys) also provides the conduit for the components higher up the Virtualization Stack to communicate with the hypervisor.

The Virtualization Stack and Other Root Partition Components



Component	Description
Windows Hypervisor Interface Library	A DLL (named WinHv.sys) located in the parent partition Windows Server 2008 instance and any guest operating systems which are <i>Hyper-V aware</i> (in other words modified specifically to operate in a Hyper-V child partition). Allows the operating system's drivers to access the hypervisor using standard Windows API calls instead of hypercalls.
VMBus	Part of Hyper-V Integration Services, the VMBus facilitates highly optimized communication between child partitions and the parent partition

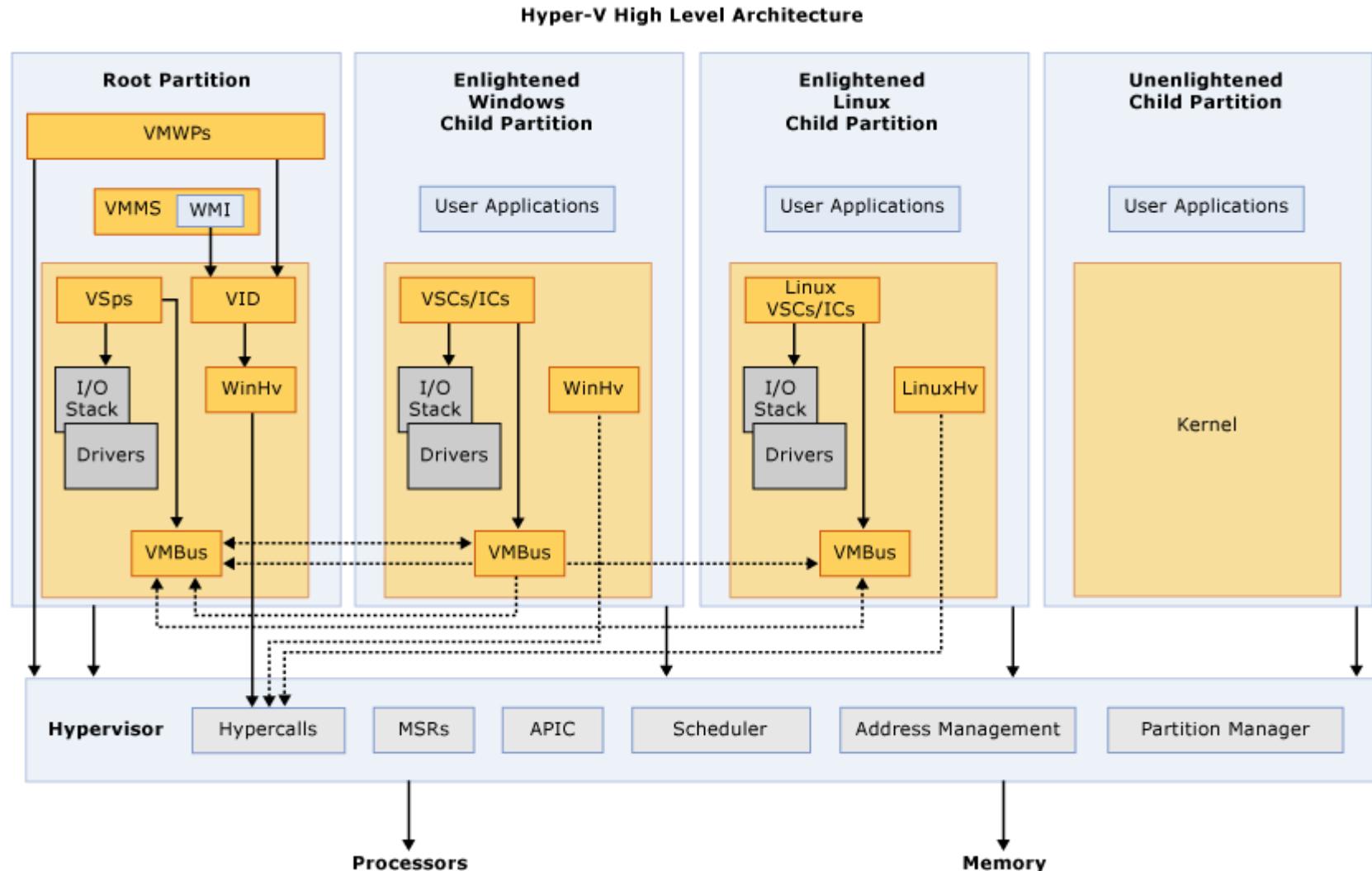
The Virtualization Stack and Other Root Partition Components



Component	Description
Virtualization Service Providers	Resides in the parent partition and provides synthetic device support via the VMBus to Virtual Service Clients (VSCs) running in child partitions.
Virtualization Service Clients	Virtualization Service Clients are synthetic device instances that reside in child partitions. They communicate with the VSPs in the parent partition over the VMBus to fulfill the child partition's device access requests.



Hyper-V Architecture





Hyper-V Architecture

- APIC – Advanced Programmable Interrupt Controller – A device which allows priority levels to be assigned to its interrupt outputs.
- Child Partition – Partition that hosts a guest operating system - All access to physical memory and devices by a child partition is provided via the Virtual Machine Bus (VMBus) or the hypervisor.
- Hypercall – Interface for communication with the hypervisor - The hypercall interface accommodates access to the optimizations provided by the hypervisor.



Hyper-V Architecture

- Hypervisor – A layer of software that sits between the hardware and one or more operating systems. Its primary job is to provide isolated execution environments called partitions. The hypervisor controls and arbitrates access to the underlying hardware.
- IC – Integration component – Component that allows child partitions to communicate with other partitions and the hypervisor.
- I/O stack – Input/output stack
- MSR – Memory Service Routine



Hyper-V Architecture

- Root Partition – Sometimes called parent partition. Manages machine-level functions such as device drivers, power management, and device hot addition/removal. The root (or parent) partition is the only partition that has direct access to physical memory and devices.
- VID – Virtualization Infrastructure Driver – Provides partition management services, virtual processor management services, and memory management services for partitions.



Hyper-V Architecture

- VMBus – Channel-based communication mechanism used for inter-partition communication and device enumeration on systems with multiple active virtualized partitions. The VMBus is installed with Hyper-V Integration Services.
- VMMS – Virtual Machine Management Service – Responsible for managing the state of all virtual machines in child partitions.
- VMWP – Virtual Machine Worker Process – A user mode component of the virtualization stack.
- The worker process provides virtual machine management services from the Windows Server 2008 instance in the parent partition to the guest operating systems in the child partitions.
- The Virtual Machine Management Service spawns a separate worker process for each running virtual machine.



Hyper-V Architecture

- VSC – Virtualization Service Client – A synthetic device instance that resides in a child partition.
- VSCs utilize hardware resources that are provided by Virtualization Service Providers (VSPs) in the parent partition.
- They communicate with the corresponding VSPs in the parent partition over the VMBus to satisfy a child partitions device I/O requests.
- VSP – Virtualization Service Provider – Resides in the root partition and provide synthetic device support to child partitions over the Virtual Machine Bus (VMBus).



Hyper-V Architecture

- WinHv – Windows Hypervisor Interface Library - WinHv is essentially a bridge between a partitioned operating system's drivers and the hypervisor which allows drivers to call the hypervisor using standard Windows calling conventions
- WMI – The Virtual Machine Management Service exposes a set of Windows Management Instrumentation (WMI)-based APIs for managing and controlling virtual machines.



hyper-v containers

- Windows Hyper-v container is a windows server container that runs in a VM.
- Every hyper-v container creates its own VM.
- This means that there is no kernel sharing between the different hyper-v containers.
- This is useful for cases where additional level of isolation is needed by customers who don't like the traditional kernel sharing done by containers.
- The same Docker image and CLI can be used to manage hyper-v containers.
- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.



hyper-v containers

- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.



hyper-v containers

There are 2 modes of hyper-v container.

1. Windows hyper-v container – Here, hyper-v container runs on top of Windows kernel. Only Windows containers can be run in this mode.
2. Linux hyper-v container – Here, hyper-v container runs on top of Linux kernel. This mode was not available earlier and it was introduced as part of Dockercon 2017. Any Linux flavor can be used as the base kernel. Docker's Linuxkit project can be used to build the Linux kernel needed for the hyper-v container. Only Linux containers can be run in this mode.

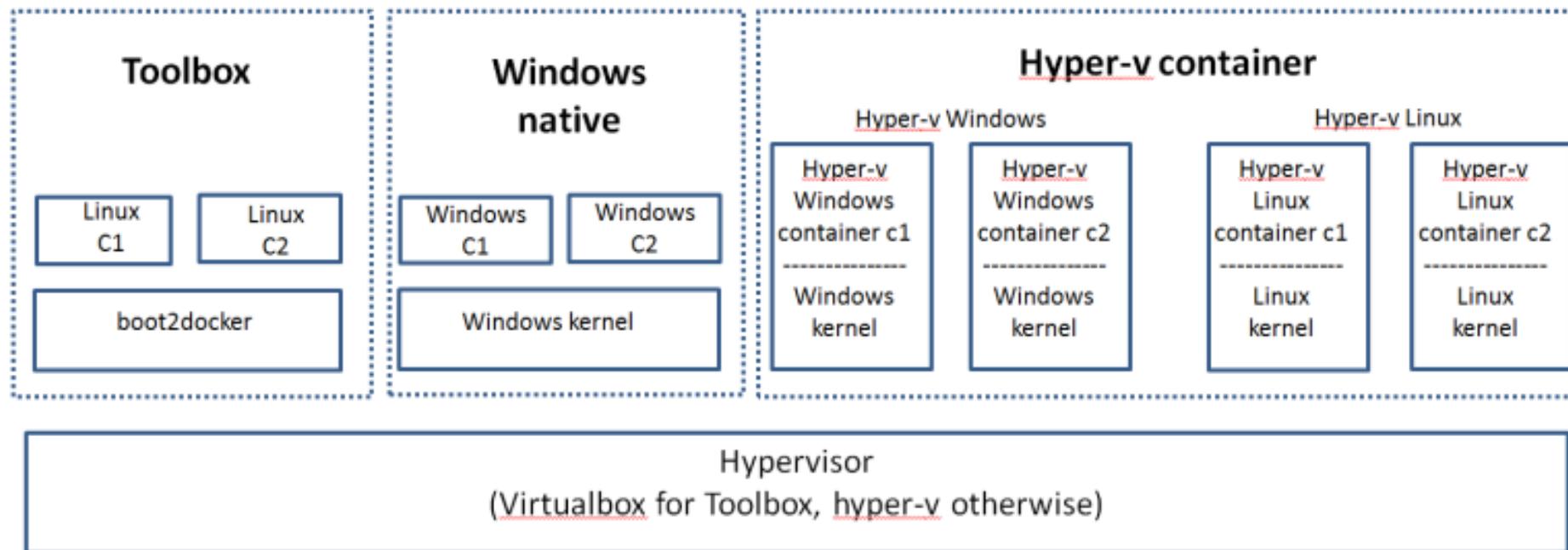


hyper-v containers

1. We cannot use Docker Toolbox and hyper-v containers at the same time. Virtualbox cannot run when “Docker for Windows” is installed.
2. <https://nickjanetakis.com/blog/should-you-use-the-docker-toolbox-or-docker-for-mac-windows>

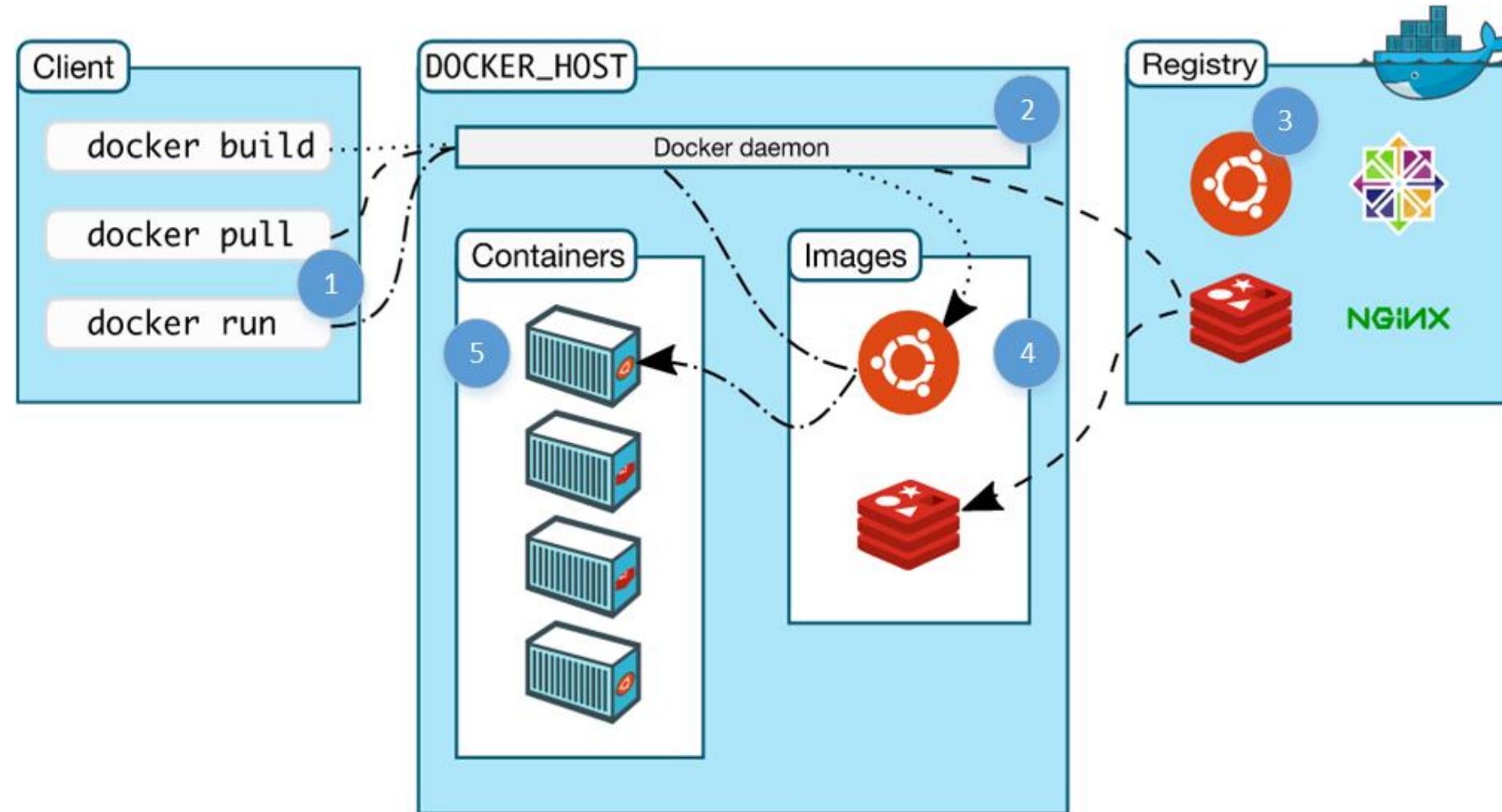


Docker



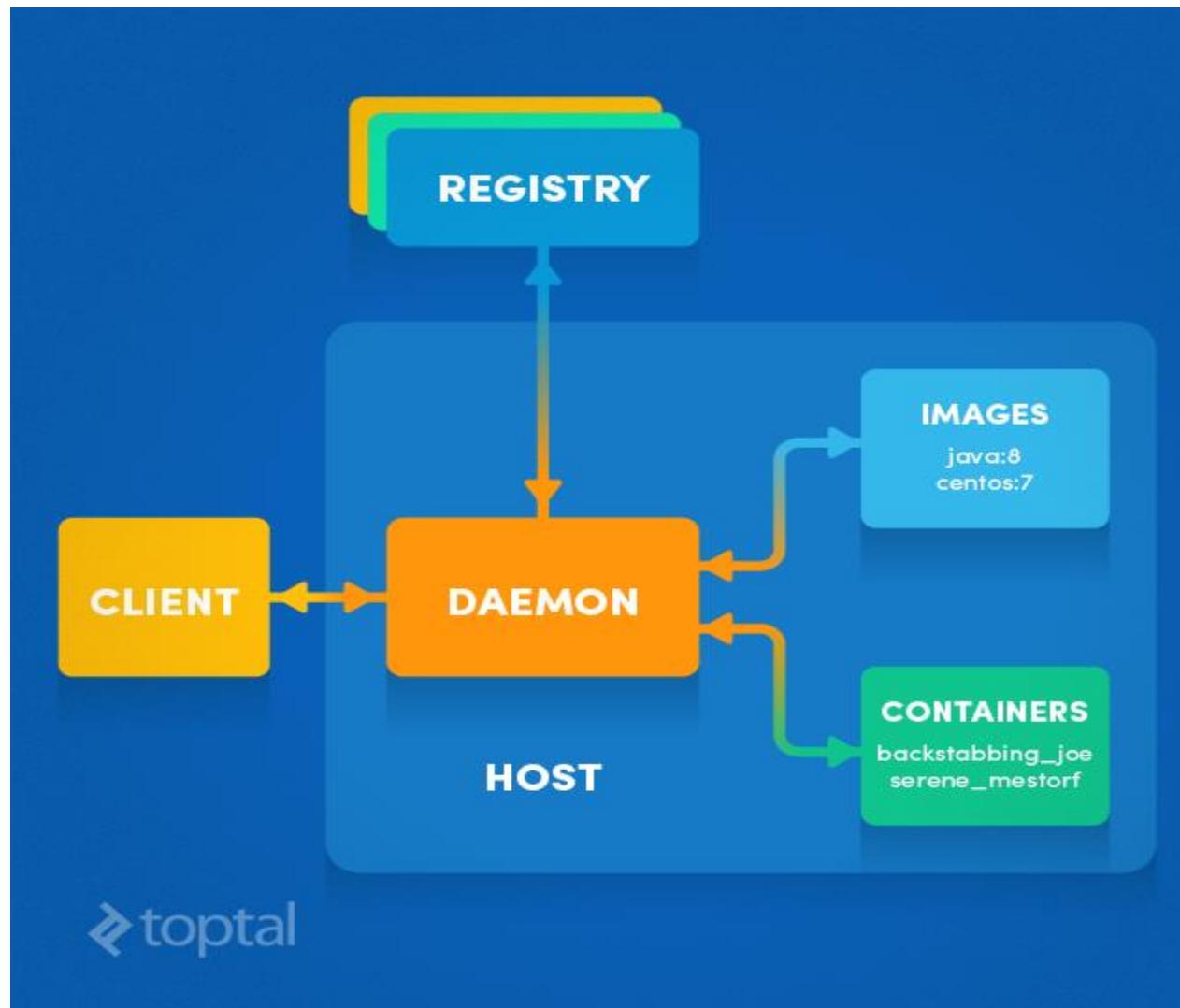


Docker Architecture

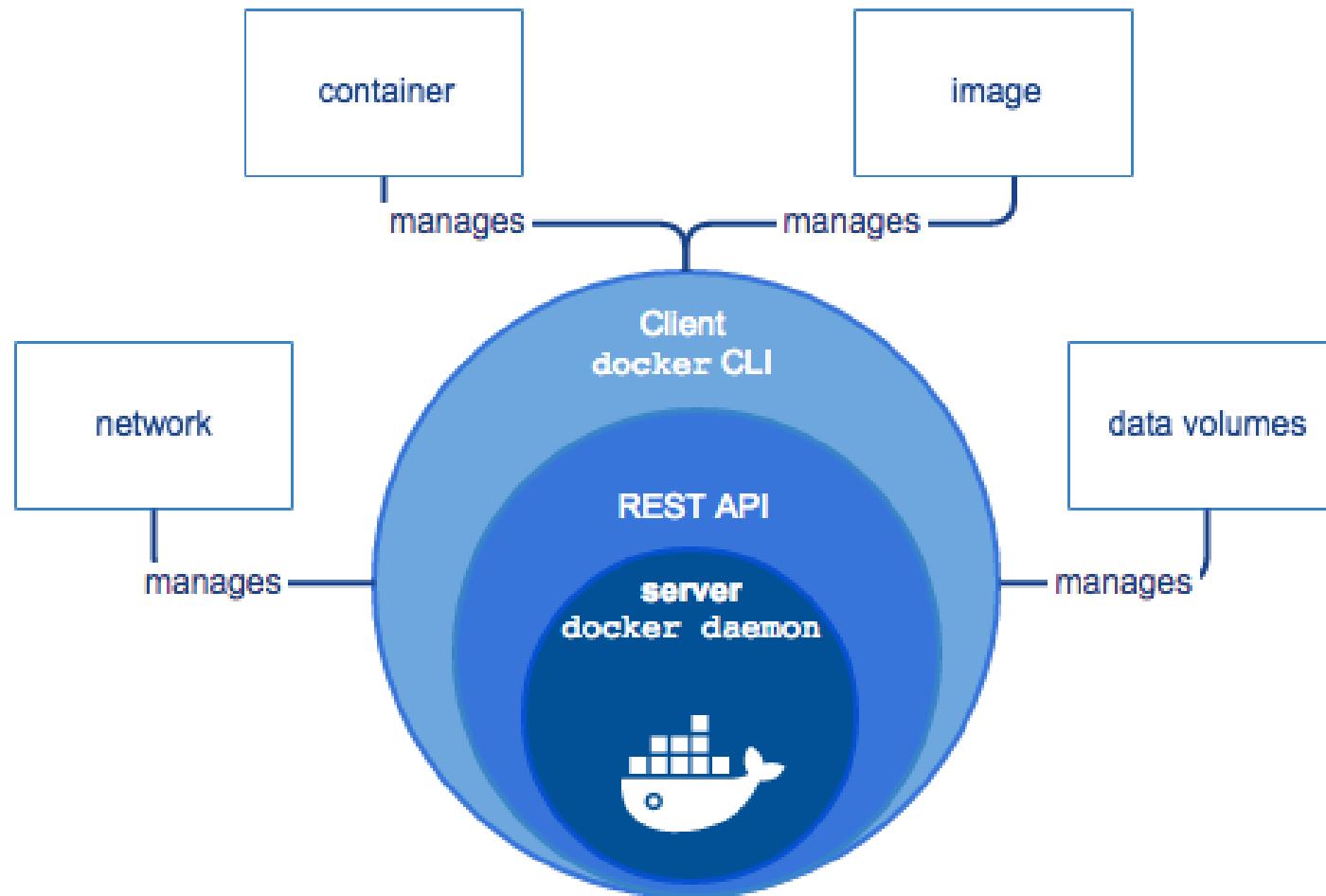




Docker Architecture



Docker Engine



Dockerfile, Docker Image And Docker Container:



- A Docker Image is created by the sequence of commands written in a file called as Dockerfile.
- When this Dockerfile is executed using a docker command it results into a Docker Image with a name.
- When this Image is executed by “docker run” command it will by itself start whatever application or service it must start on its execution.



Image Demo

- <https://learnk8s.io/spring-boot-kubernetes-guide>





Docker Network

- Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers.
- There are basically two types of networks available - the default Docker network and user-defined networks.
- By default, you get three different networks on the installation of Docker - none, bridge, and host.



Docker Network

- The none and host networks are part of the network stack in Docker.
- The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.
- This network is not commonly used as it does not scale well and has constraints in terms of network usability and service discovery.



Docker Network

- The other type of networks is user-defined networks. Administrators can configure multiple user-defined networks.
- There are three types:
- Bridge network: Similar to the default bridge network, a user-defined Bridge network differs in that there is no need for port forwarding for containers within the network to communicate with each other.
- The other difference is that it has full support for automatic network discovery.



Docker Network

- Overlay network: An Overlay network is used when you need containers on separate hosts to be able to communicate with each other, as in the case of a distributed network.
- However, a caveat is that swarm mode must be enabled for a cluster of Docker engines, known as a swarm, to be able to join the same group.
- Macvlan network: When using Bridge and Overlay networks a bridge resides between the container and the host.
- A Macvlan network removes this bridge, providing the benefit of exposing container resources to external networks without dealing with port forwarding.
- This is realized by using MAC addresses instead of IP addresses.



Storage

- You can store data within the writable layer of a container but it requires a storage driver.
- Being non-persistent, it perishes whenever the container is not running.
- It is not easy to transfer this data.
- In terms of persistent storage, Docker offers four options:
- Data Volume, Data Volume Container, Directory Mounts, Storage Plugins



Storage

- There are storage plugins from various companies to automate the storage provisioning process. For example,
- HPE 3PAR
- EMC (ScaleIO, XtremIO, VMAX, Isilon)
- NetApp
- There are also plugins that support public cloud providers like:
- Azure File Storage
- Google Compute Platform.



Registries

- A Docker registry is a storage and distribution system for named Docker images.
- The same image might have multiple different versions, identified by their tags.
- A Docker registry is organized into Docker repositories , where a repository holds all the versions of a specific image.
- The registry allows Docker users to pull images locally, as well as push new images to the registry (given adequate access permissions when applicable).



Registries

- By default, the Docker engine interacts with DockerHub , Docker's public registry instance.
- However, it is possible to run on-premise the open-source Docker registry/distribution, as well as a commercially supported version called Docker Trusted Registry .



Docker Hub

- DockerHub is a hosted registry solution by Docker Inc.
- Besides public and private repositories, it also provides automated builds, organization accounts, and integration with source control solutions like Github and Bitbucket .



Docker Hub

- A public repository is accessible by anyone running Docker, and image names include the organization/user name.
- For example, `docker pull jenkins/jenkins` will pull the Jenkins CI server image with tag latest from the Jenkins organization.
- There are hundreds of thousands public images available.
- Private repositories restrict access to the repository creator or members of its organization.



Docker Hub

- DockerHub supports official repositories, which include images verified for security and best practices.
- These do not require an organization/user name, for example docker pull nginx will pull the latest image of the Nginx load balancer.
- DockerHub can perform automated image builds if the DockerHub repository is linked to a source control repository which contains a build context (Dockerfile and all any files in the same folder).
- A commit in the source repository will trigger a build in DockerHub.



Other Public Registries

- Other companies host paid online Docker registries for public use.
- Cloud providers like AWS and Google, who also offer container-hosting services, market the high availability of their registries.



Other Public Registries

- Amazon Elastic Container Registry (ECR) integrates with AWS Identity and Access Management (IAM) service for authentication. It supports only private repositories and does not provide automated image building.
- Google Container Registry (GCR) authentication is based on Google's Cloud Storage service permissions. It supports only private repositories and provides automated image builds via integration with Google Cloud Source Repositories, GitHub, and Bitbucket.



Other Public Registries

- Azure Container Registry (ACR) supports multi-region registries and authenticates with Active Directory. It supports only private repositories and does not provide automated image building.
- CoreOS Quay supports OAuth and LDAP authentication. It offers both (paid) private and (free) public repositories, automatic security scanning and automated image builds via integration with GitLab, GitHub, and Bitbucket.
- Private Docker Registry supports OAuth, LDAP and Active Directory authentication. It offers both private and public repositories, free up to 3 repositories (private or public).



Private Registries

- Use cases for running a private registry on-premise (internal to the organization) include:
- Distributing images inside an isolated network (not sending images over the Internet)
- Creating faster CI/CD pipelines (pulling and pushing images from internal network), including faster deployments to on-premise environments
- Deploying a new image over a large cluster of machines
- Tightly controlling where images are being stored



Private Registries

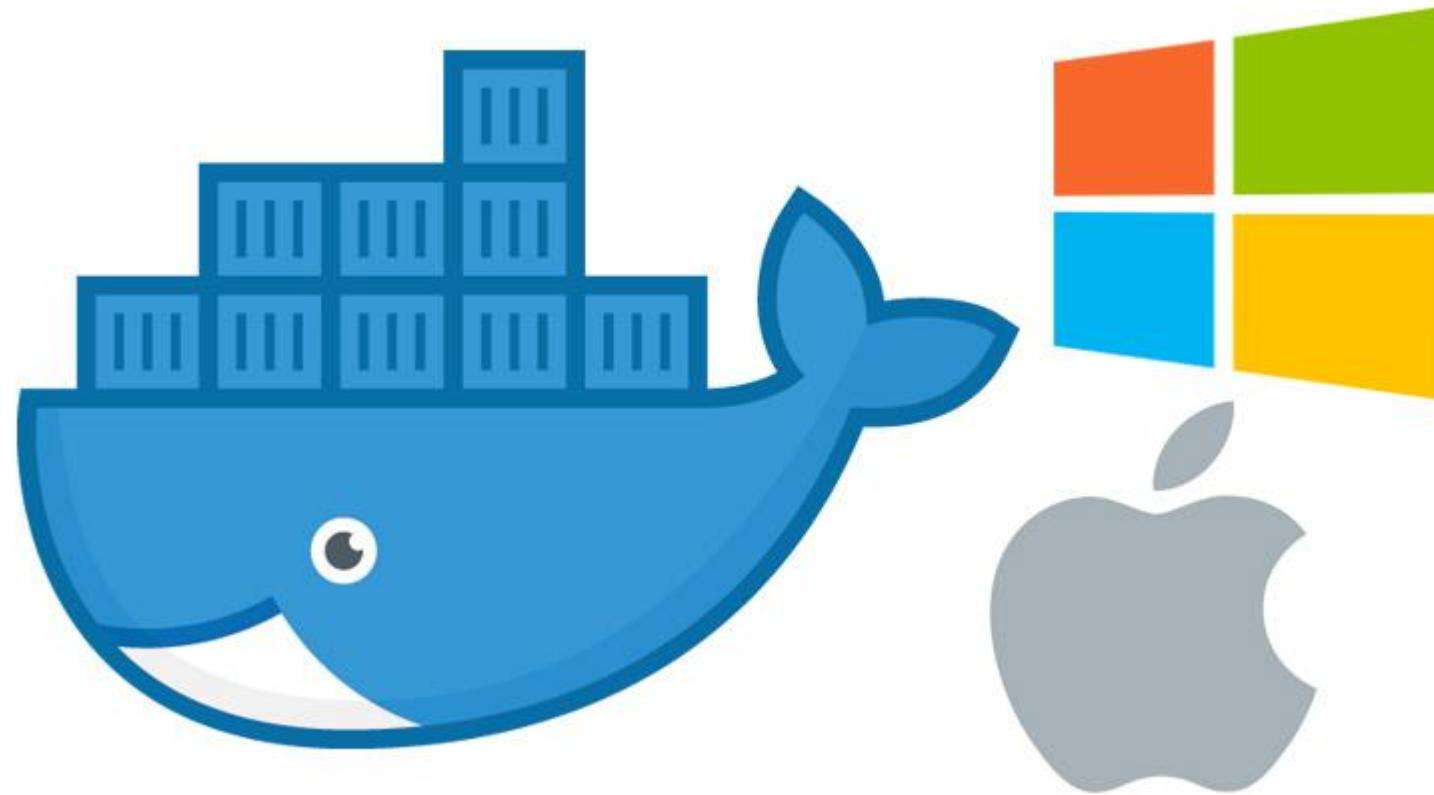
- Running a private registry system, especially when delivery to production depends on it, requires operational skills such as ensuring availability, logging and log processing, monitoring, and security.
- Strong understanding of http and overall network communications is also important.



Private Registries

- Docker Trusted Registry is Docker Inc's commercially supported version, providing high availability via replication, image auditing, signing and security scanning, integration with LDAP and Active Directory.
- Harbor is a VMWare open source offering which also provides high availability via replication, image auditing, integration with LDAP and Active Directory.
- GitLab Container Registry is tightly integrated with GitLab CI's workflow, with minimal setup.
- JFrog Artifactory for strong artifact management (not only Docker images but any artifact).

Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



- If you're on MacOS or Windows you can install Docker with:
 - Docker for Mac / Windows (now known as Docker Desktop)
 - Docker Toolbox
 - Running your own Virtual Machine and installing Docker yourself

Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



- If you're on MacOS or Windows you can install Docker with:
 - Docker for Mac / Windows (now known as Docker Desktop)
 - Docker Toolbox
 - Running your own Virtual Machine and installing Docker yourself

Docker for Mac / Docker for Windows (Docker Desktop)



- Pros
- Offers the most “native” experience, you can easily use any terminal you want since Docker is effectively running on localhost from MacOS / Windows’ POV.
- Docker is heavily developing and polishing this solution.

Docker for Mac / Docker for Windows (Docker Desktop)



- **Cons**
- On certain MacOS hardware combos the volume performance can be a little slow.
- I can legit say there are not any “wow this sucks!” cons for Windows, it’s really solid.



Docker Toolbox

- Pros
- Offers an “out of the box” Docker experience if you have no other choice.



Docker Toolbox

- Cons
- You need to either use the Docker Quickstart Terminal, or configure your own terminal to connect to the Docker Daemon running a VM.
- Not a native solution, so you'll need to access your Docker Machine's IP address if you're developing web apps. Example: 192.168.99.100 instead of localhost.



Docker Toolbox

- Cons
- Unless you jump through hoops, your code needs to live in your Windows user directory such as C:\Users\eswari\src\myapp. Otherwise Docker won't be able to find it.
- Suffers from typical VirtualBox edge case bugs and mount performance issues.



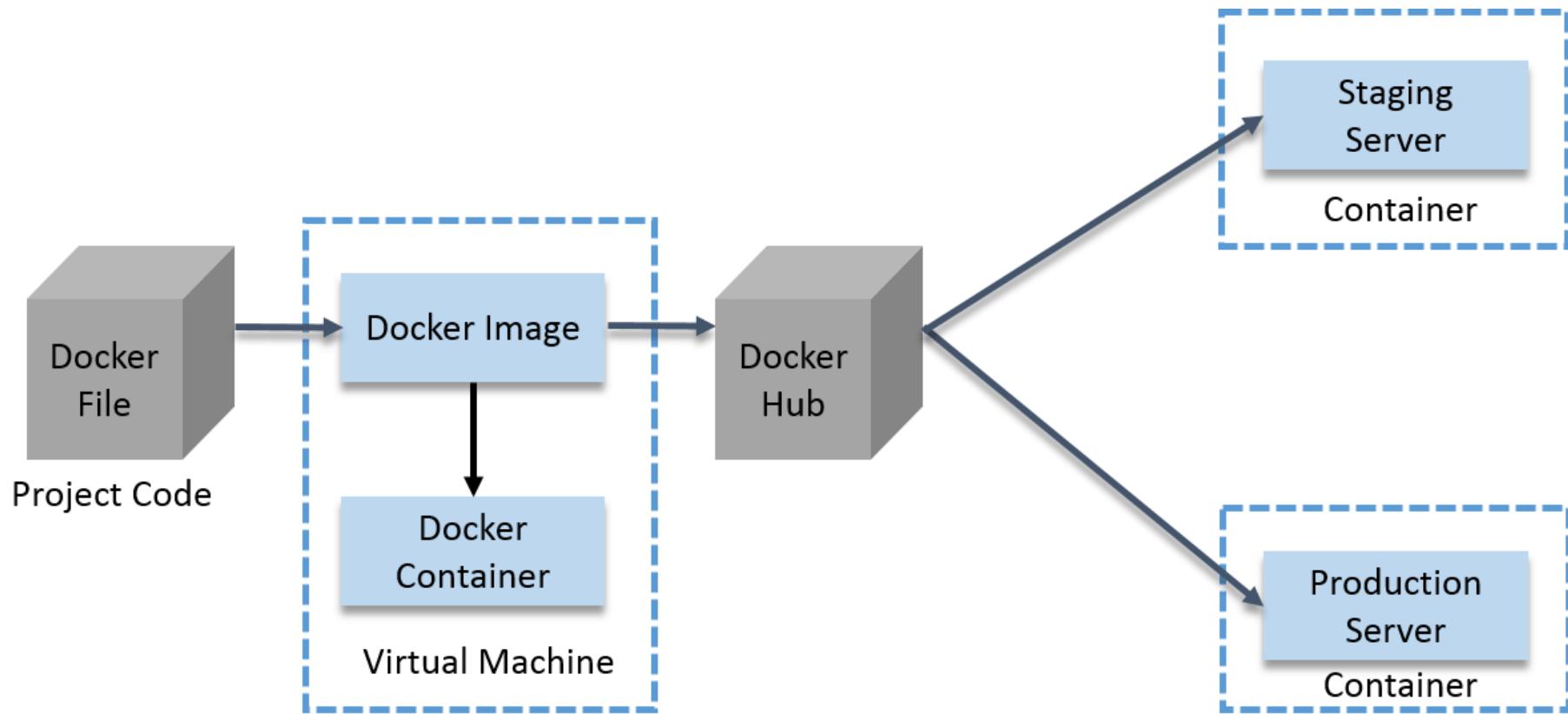
What is Docker

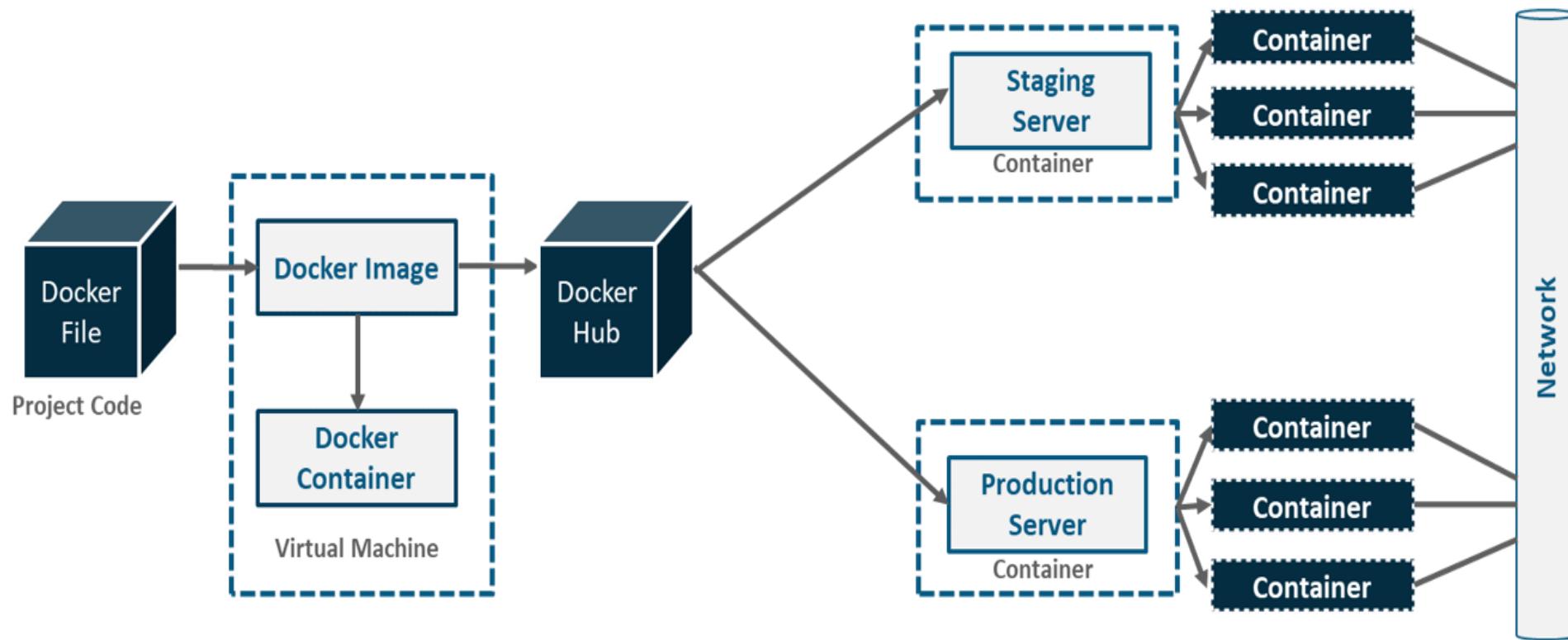
- What is Docker?
- At its heart, Docker is software which lets you create an *image* and then run instances of that image in a *container*.
- Docker maintains a vast repository of images, called the Docker Hub which you can use as starting points or as free storage for your own images.
- You can install Docker, choose an image you'd like to use, then run an instance of it in a container.



Difference between VM and Container

- Applications running in virtual machines, apart from the hypervisor, require a full instance of the operating system and any supporting libraries.
- Containers, on the other hand, share the operating system with the host.
- Hypervisor is comparable to the container engine (represented as Docker on the image) in a sense that it manages the lifecycle of the containers.
- The important difference is that the processes running inside the containers are just like the native processes on the host, and do not introduce any overheads associated with hypervisor execution.
- Additionally, applications can reuse the libraries and share the data between containers.



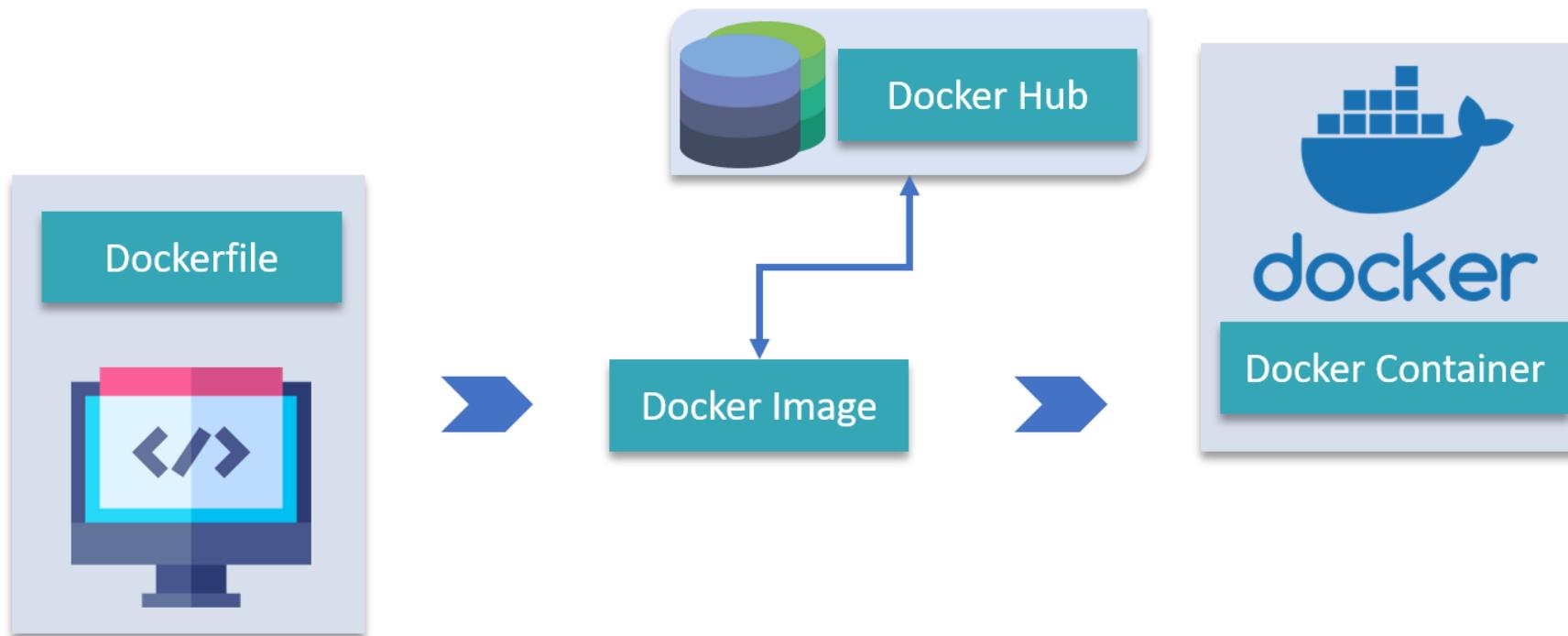


Docker Hub:



- Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

Docker Hub





Docker Architecture

- It consists of a Docker Engine which is a client-server application with three major components:
- A server which is a type of long-running program called a daemon process (the docker command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.



Docker Architecture

- By default, the main registry is the Docker Hub which hosts public and official images.
- Organizations can also host their private registries if they desire.
- Images can be downloaded from registries explicitly (`docker pull imageName`) or implicitly when starting a container. Once the image is downloaded it is cached locally.
- Containers are the instances of images - they are the living thing. There could be multiple containers running based on the same image.



Docker Architecture

- At the center, there is the Docker daemon responsible for creating, running, and monitoring containers.
- It also takes care of building and storing images.
- Finally, on the left-hand side there is a Docker client. It talks to the daemon via HTTP.
- Unix sockets are used when on the same machine, but remote management is possible via HTTP based API.

Goals of Docker Networking





Creating Test Database Server

- This is a great Docker use case.
- We might not want to run our production database in Docker (perhaps we'll just use Amazon RDS for example), but we can spin up a clean MySQL database in no time as a Docker container for development - leaving our development machine clean and keeping everything we do controlled and repeatable.



Docker Compose

- Docker Compose is basically used to run multiple Docker Containers as a single server. Let me give you an example:
- Suppose if I have an application which requires WordPress, Maria DB and PHP MyAdmin. I can create one file which would start both the containers as a service without the need to start each one separately. It is really useful especially if you have a microservice architecture.



Docker Container

Column	Description
Container ID	The unique ID of the container. It is a SHA-256.
Image	The name of the container image from which this container is instantiated.
Status	The status of the container (created, restarting, running, removing, paused, exited, or dead).
Ports	The list of container ports that have been mapped to the host.
Names	The name assigned to this container (multiple names are possible).



Docker Client and Docker Engine

- **Docker Client** : This is the utility we use when we run any docker commands e.g. docker run (docker container run) , docker images , docker ps etc. It allows us to run these commands which a human can easily understand.
- **Docker Daemon/Engine**: This is the part which does rest of the magic and knows how to talk to the kernel, makes the system calls to create, operate and manage containers, which we as users of docker dont have to worry about.



Creating Test Database Server

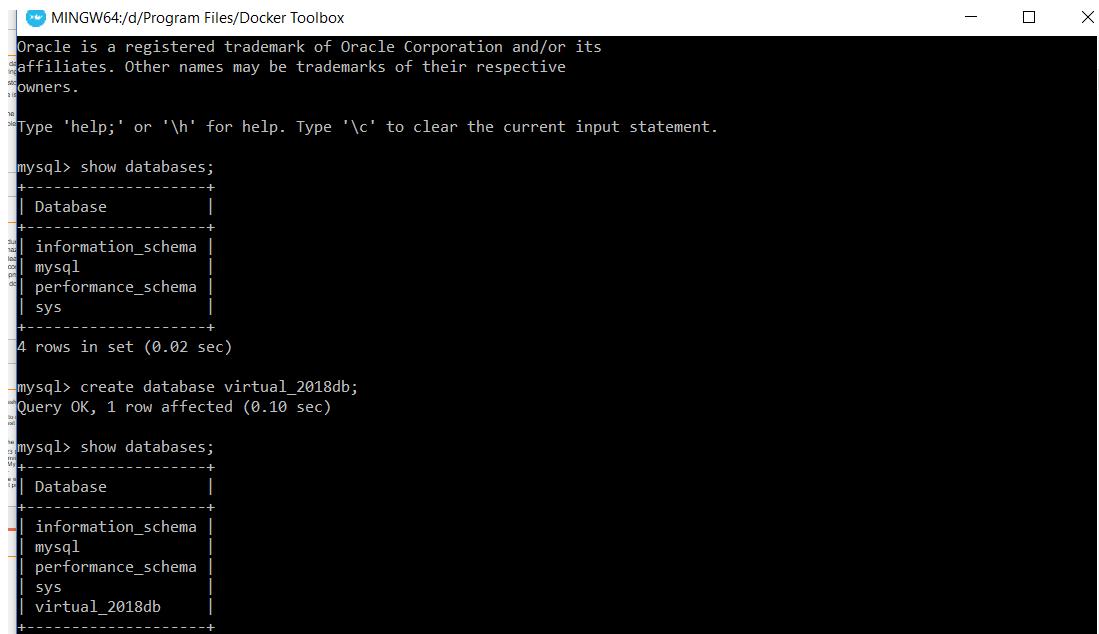
- docker run --name olddb -e MYSQL_ROOT_PASSWORD=vignesh -e MYSQL_DATABASE=virtusa_2018db -e MYSQL_USER=root -p 3306:3306 mysql/mysql-server:5.5
- docker run tells the engine we want to run an image (the image comes at the end, mysql:latest)
- --name db names this container db.
- -d detach - i.e. run the container in the background.
- -e MYSQL_ROOT_PASSWORD=123 the -e is the flag that tells docker we want to provide an environment variable. The variable following it is what the MySQL image checks for setting the default root password.
- -p 3306:3306 tells the engine that we want to map the port 3306 from inside the container to out port 3306.

```
MINGW64/d/Program Files/Docker Toolbox
File Home
← → ↑ ↓ MINGW64/d/Program Files/Docker Toolbox
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
[Entrypoint] Not creating mysql user. MYSQL_USER and MYSQL_PASSWORD must be specified to create a mysql user.
[Entrypoint] ignoring /docker-entrypoint-initdb.d/*
[Entrypoint] Server shut down
[Entrypoint] MySQL init process done. Ready for start up.
[Entrypoint] Starting MySQL 5.5.62-1.1.8
heroku-app181120 19:49:03 [Note] mysqld (mysqld 5.5.62) starting as process 1 ...
jruby-9.1.1
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
kibana-6.3$ docker kill $(docker ps -q)
304981f38447
logstash-6
material-fr
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
metricbeat$ docker kill $(docker ps -a -q)
Error response from daemon: Cannot kill container: 304981f384473c7caf3235c2fa998da2f0240efff01a1ed0d00381c25180b1dd is not running
Microservice
nssm-2.24
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker rm $(docker ps -a -q)
304981f38447
OpenSSL-1.0.2j-fips
Oracle
Pega Robo
PostgreSQL
Program Files
Program Files (x86)
Python
Ruby25-x64
scope
SQLEXPRESS_x86_ENU
SQLServer2017Media
temp
winlogbeat-6.3.0-windows-x86_64
wwwroot
auditbeat-6.3.0-windows-x86_64.zip

3 items
Type here to search
File Explorer
OneDrive
Microsoft Edge
Google Chrome
PowerShell
Task View
Settings
File Explorer
Search resources
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
01:20
ENG
21/11/2018
39
```

Creating Test Database Server

- Docker ps
- docker exec -it olddb /bin/bash
- mysql –u root –p



```
MINGW64:/d/Program Files/Docker Toolbox
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.02 sec)

mysql> create database virtual_2018db;
Query OK, 1 row affected (0.10 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| virtual_2018db |
+-----+
```



Springboot Docker

- <plugin>
- <groupId>com.spotify</groupId>
- <artifactId>docker-maven-plugin</artifactId>
- <version>VERSION GOES HERE</version>
- <configuration>
- <imageName>example</imageName>
- <baseImage>java</baseImage>
- <entryPoint>["java", "-jar", "\${project.build.finalName}.jar"]</entryPoint>
- <!-- copy the service's jar file from target into the root directory of the image -->
- <resources>
- <resource>
- <targetPath>/</targetPath>
- <directory>\${project.build.directory}</directory>
- <include>\${project.build.finalName}.jar</include>
- </resource>
- </resources>
- </configuration>
- </plugin>



Best practices for writing Dockerfiles

- Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image.
- A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.
- A Docker image consists of read-only layers each of which represents a Dockerfile instruction.
- The layers are stacked and each one is a delta of the changes from the previous layer.



Best practices for writing Dockerfiles

- FROM ubuntu:18.04
- COPY ./app
- RUN make /app
- CMD python /app/app.py
- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client's current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.



Best practices for writing Dockerfiles

- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/



Springboot Docker

- Spring boot Project Build
- Goal package docker:build
- mvn clean install dockerfile:build
- Docker folder created and docker image name example deployed in docker machine
- Check docker images
- Docker ps



Springboot Docker

- Docker run -t --name sampleapp --link v1db -p 8080:8080 example:latest
- Example:latest --- image name
- --name -- container reference
- --link -- db ref in container

```
docker run -h 192.168.99.100 -p 7070:7070 -t my-repo/example --name my-repo-image:latest
```



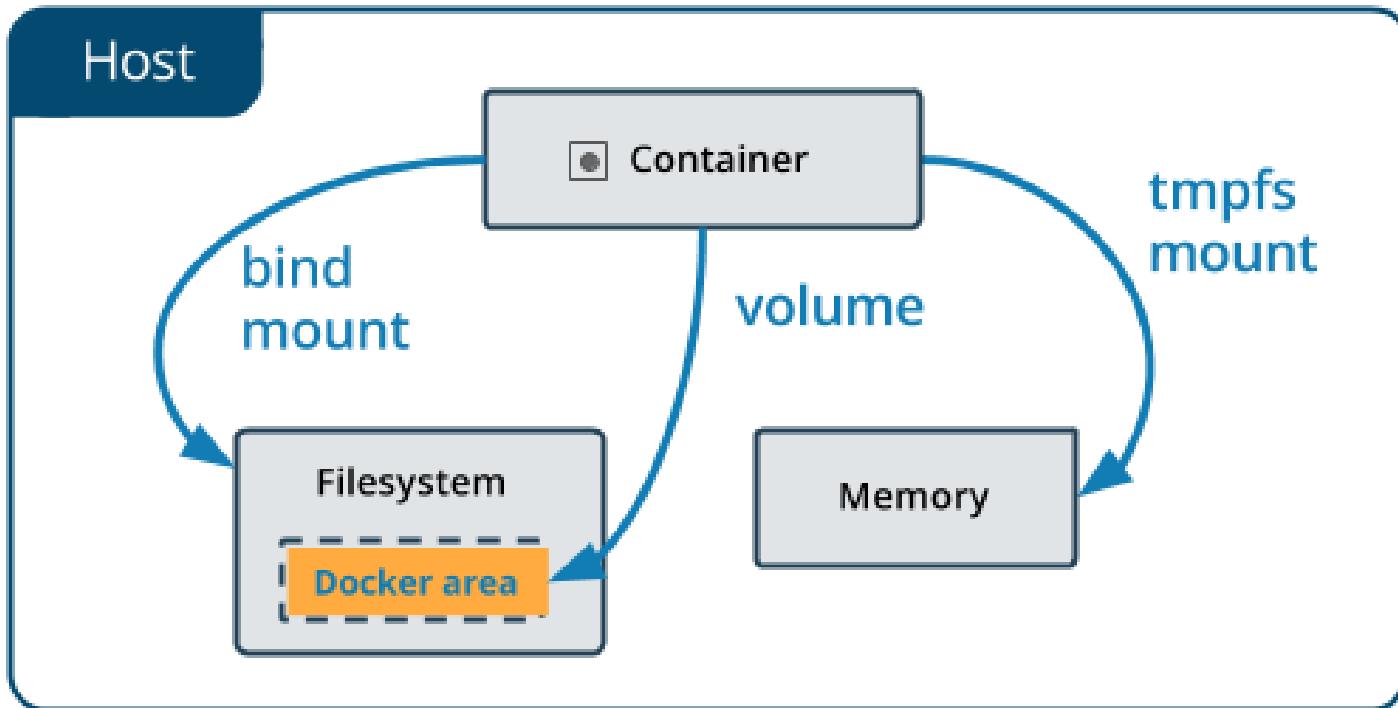
Docker Volume

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- While bind mounts are dependent on the directory structure of the host machine, volumes are completely managed by Docker.



Docker Volume

- Volumes have several advantages over bind mounts:
 - Volumes are easier to back up or migrate than bind mounts.
 - You can manage volumes using Docker CLI commands or the Docker API.
 - Volumes work on both Linux and Windows containers.
 - Volumes can be more safely shared among multiple containers.
 - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
 - New volumes can have their content pre-populated by a container





Manage Data in Docker

- There are 2 ways in which you can manage data in Docker:
 - Data volumes
 - Data volume containers



Manage Data in Docker

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created.
- By default, it is not deleted when the container is stopped.
- It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated.
- Data volumes can be shared across containers too. They could be mounted in read-only mode too.



Manage Data in Docker

- Mounting a Data volume
- docker container run -it -v/udata --tty ubuntu /bin/bash
- cd udata
- touch file1.txt
- exit
- docker container restart 2eec01eb7368
- docker attach 2eec01eb7368exit
- docker container rm 2eec01eb7368
- docker volume ls



Manage Data in Docker

- After removing container also the data volume is still present on the host.
- This is a dangling or ghost volume and could remain there on your machine consuming space.
- Do remember to clean up if you want.
- Alternatively, there is also a -v option while removing the container

```
c:\root@46a4edc2cf30:/ Error response from daemon: You cannot remove a running container e26b6d0c5de3bc889381c20b2d169ce8a61bf701dfe46aea127e0e03d314fb0e. Stop the container before attempting removal or force remove  
C:\WINDOWS\system32>docker container stop e26b6d0c5de3  
e26b6d0c5de3  
C:\WINDOWS\system32>docker container rm e26b6d0c5de3  
e26b6d0c5de3  
C:\WINDOWS\system32>docker container run -it --tty ubuntu /bin/bash  
Unable to find image 'ubuntu:latest' locally  
latest: Pulling from library/ubuntu  
a4a2a29f9ba4: Pull complete  
127c9761dcba: Pull complete  
d13bf203e905: Pull complete  
4039240d2e0b: Pull complete  
Digest: sha256:35c4a2c15539c6c1e4e5fa4e554dac323ad0107d8eb5c582d6ff386b383b7dce  
Status: Downloaded newer image for ubuntu:latest  
root@46a4edc2cf30:/# ls  
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var  
root@46a4edc2cf30:/#
```



```
root@2eec01eb7368:/udata
laughingeinstein

C:\WINDOWS\system32>docker container rm 46a4edc2cf30
46a4edc2cf30

C:\WINDOWS\system32>docker container run -it -v/udata --tty ubuntu /bin/bash
root@2eec01eb7368:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  udata  usr  var
root@2eec01eb7368:/# cd udata
root@2eec01eb7368:/udata#
```

```
c:\> Administrator: Command Prompt
    "LowerDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1-init/diff:/var/lib/docker/overlay2/3eab3aebc8c580252e330fe3efa59b8d92c1339cb4dbf46e8fc374dfef63f569/diff:/var/lib/docker/overlay2/3872c83edf6b5c30265e21691152c9126a2ffee424afc81202f02f07b1819210/diff:/var/lib/docker/overlay2/25d3f8faffb92bef36d1b620a6ababbd344c1cb58f41fb565620be058a0dcf8e/diff:/var/lib/docker/overlay2/e4ebde681507d7b624d11dae676fbca8cb273d91e18e55a3d511a5b4fd0cd9ac/diff",
        "MergedDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/merged",
        "UpperDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/diff",
        "WorkDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/work"
    },
    "Name": "overlay2"
},
"Mounts": [
    {
        "Type": "volume",
        "Name": "7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3",
        "Source": "/var/lib/docker/volumes/7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3/_data",
        "Destination": "/udata",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
"Config": {
    "Hostname": "2eec01eb7368",
    "Domainname": "",
    "User": "",
    "AttachStdin": true,
    "AttachStdout": true,
    "AttachStderr": true,
    "Tty": true,
    "OpenStdin": true,
    "StdinOnce": true,
    "Env": [

```



Type here to search



22:08
29/06/2020 18

```
c:\ root@2eec01eb7368: /udata
94d81de361e1      mysql          "docker-entrypoint.s..."   3 hours ago      Up 2 hours  330
6/tcp, 33060/tcp   virtusa-mysql
36acbeba8c36     docker/getting-started
                  laughing_einstein

C:\WINDOWS\system32>docker container restart 2eec01eb7368
2eec01eb7368

C:\WINDOWS\system32>docker attach 2eec01eb7368
root@2eec01eb7368:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  udata  usr  var
root@2eec01eb7368:/# cd udata
root@2eec01eb7368:/udata# ls
file1.txt
root@2eec01eb7368:/udata#
```



Manage Data in Docker

- The sequence of steps are as follows:
- We restarted the container (**container1**)
- We attached to the running container (**container1**)
- We did a `ls` and found that our volume `/data` is still there.
- We did a `cd` into the `/data` volume and did a `ls` there. And our file is still present.



Mounting a Host Directory as a Data volume

The screenshot shows the Docker Settings window. The title bar includes the Docker logo, user profile (eswaribala), and standard window controls. The main area has a header "Resources File sharing". A descriptive text states: "These directories (and their subdirectories) can be bind mounted into Docker containers. You can check the [documentation](#) for more details." Below this, two host directory entries are listed: "E:\DockerFileShare" and "C:\path\to\exported\directory". Each entry has a minus sign (-) to its right and a plus sign (+) to its left. On the left sidebar, under the "Resources" tab, there is a list of sections: General, ADVANCED (FILE SHARING is selected), PROXIES, NETWORK, Docker Engine, Command Line, and Kubernetes. The "Resources" tab is highlighted with a blue background.

```
docker container run -it -v e:/DockerFileShare/project/web01:/mnt/test ubuntu /bin/bash
```



Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
 - You first create a Data volume container
 - Create another container and mount the volume from the container created in Step 1.



Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
 - You first create a Data volume container
 - Create another container and mount the volume from the container created in Step 1.



Data volume containers

```
c:\ Administrator: Command Prompt - docker run -it -v /data --name container1 busybox
```

```
local          4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local          7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local          030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local          5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local          a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dc2
```

```
C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aa1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```



Data volume containers

```
c:\ Administrator: Command Prompt - docker run -it -v /data --name container1 busybox
```

```
local          4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local          7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local          030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local          5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local          a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dc2
```

```
C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aa1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```



Docker network

```
Administrator: Command Prompt
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data # exit

C:\WINDOWS\system32>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
5cc9ae61ffda   bridge    bridge      local
3857a50b891b   host      host       local
e43a4f132e84   none     null       local

C:\WINDOWS\system32>
```



Docker network

- **docker network inspect bridge**
- Docker automatically creates a subnet and gateway for the bridge network, and docker run automatically adds containers to it. I
- If you have containers running on your network, docker network inspect displays networking information for your containers.



Docker network

- Any containers on the same network may communicate with one another via IP addresses.
- Docker does not support automatic service discovery on bridge.
- You must connect containers with the --link option in your docker run command.



Docker network

- The Docker bridge supports port mappings and docker run --link allowing communications between containers on the docker0 network.
- However, these error-prone techniques require unnecessary complexity.
- It's better to define your own networks instead.



Docker network

- None
- This offers a container-specific network stack that lacks a network interface.
- This container only has a local loopback interface (i.e., no external network interface).



Docker network

- Host
- This enables a container to attach to your host's network (meaning the configuration inside the container matches the configuration outside the container).



Defining your own networks

- You can create multiple networks with Docker and add containers to one or more networks.
- Containers can communicate within networks but not across networks.
- A container with attachments to multiple networks can connect with all of the containers on all of those networks.
- This lets you build a “hub” of sorts to connect to multiple networks and separate concerns.



Defining your own networks

- Bridge networks (similar to the default docker0 network) offer the easiest solution to creating your own Docker network.
- While similar, you do not simply clone the default0 network, so you get some new features and lose some old ones.
- Follow along below to create your own `my_isolated_bridge_network` and run your Postgres container `my_mysql_db` on that network:



Defining your own networks

- docker network create --driver bridge my_isolated_bridge_network
- docker network inspect my_isolated_bridge_network
- docker network ls
- docker run --net=my_isolated_bridge_network --name=my_psql_db postgres
- docker run --network=my_isolated_bridge_network --name=my_app hello-world



Defining your own networks

```
C:\ Administrator: Command Prompt
{
    "EnableIPv6": false,
    "IPAM": {
        "Driver": "default",
        "Options": {},
        "Config": [
            {
                "Subnet": "172.18.0.0/16",
                "Gateway": "172.18.0.1"
            }
        ],
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "fce4adc125341807e7b334cb2908436a2a5b9596aa6b08749203f9c6e18d6dbb": {
                "Name": "competent_darwin",
                "EndpointID": "ca6a9d07770e8226967cf6c601d327cfe19b7b15396a09229f0e540f83f066e1",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
}

C:\WINDOWS\system32>
```

The screenshot shows a Windows Command Prompt window titled 'Administrator: Command Prompt'. The command entered is a JSON configuration object for a Docker network. The configuration includes settings for IPAM (IP Address Management), such as a subnet of '172.18.0.0/16' and a gateway of '172.18.0.1'. It also defines a container named 'competent_darwin' with specific endpoint details like MAC address and IPv4 address. The command prompt is located at the bottom of the screen, along with the Windows taskbar which includes icons for File Explorer, Edge browser, and other system tools.



Creating an overlay network

- If you want native multi-host networking, you need to create an overlay network.
- These networks require a valid key-value store service, such as Consul, Etcd, or ZooKeeper.
- You must install and configure your key-value store service before creating your network.
- Your Docker hosts (you can use multiple hosts with overlay networks) must communicate with the service you choose.
- Each host needs to run Docker.
- You can provision the hosts with Docker Machine.



Creating an overlay network

- Create the overlay network in a similar manner to the bridge network (network name `my_multi_host_network`):
- `docker network create --driver overlay my_multi_host_network`
- Launch containers on each host; make sure you specify the network name:
- `docker run -itd -net=my_multi_host_network my_python_app`

Getting Started with Artifactory as a Docker Registry

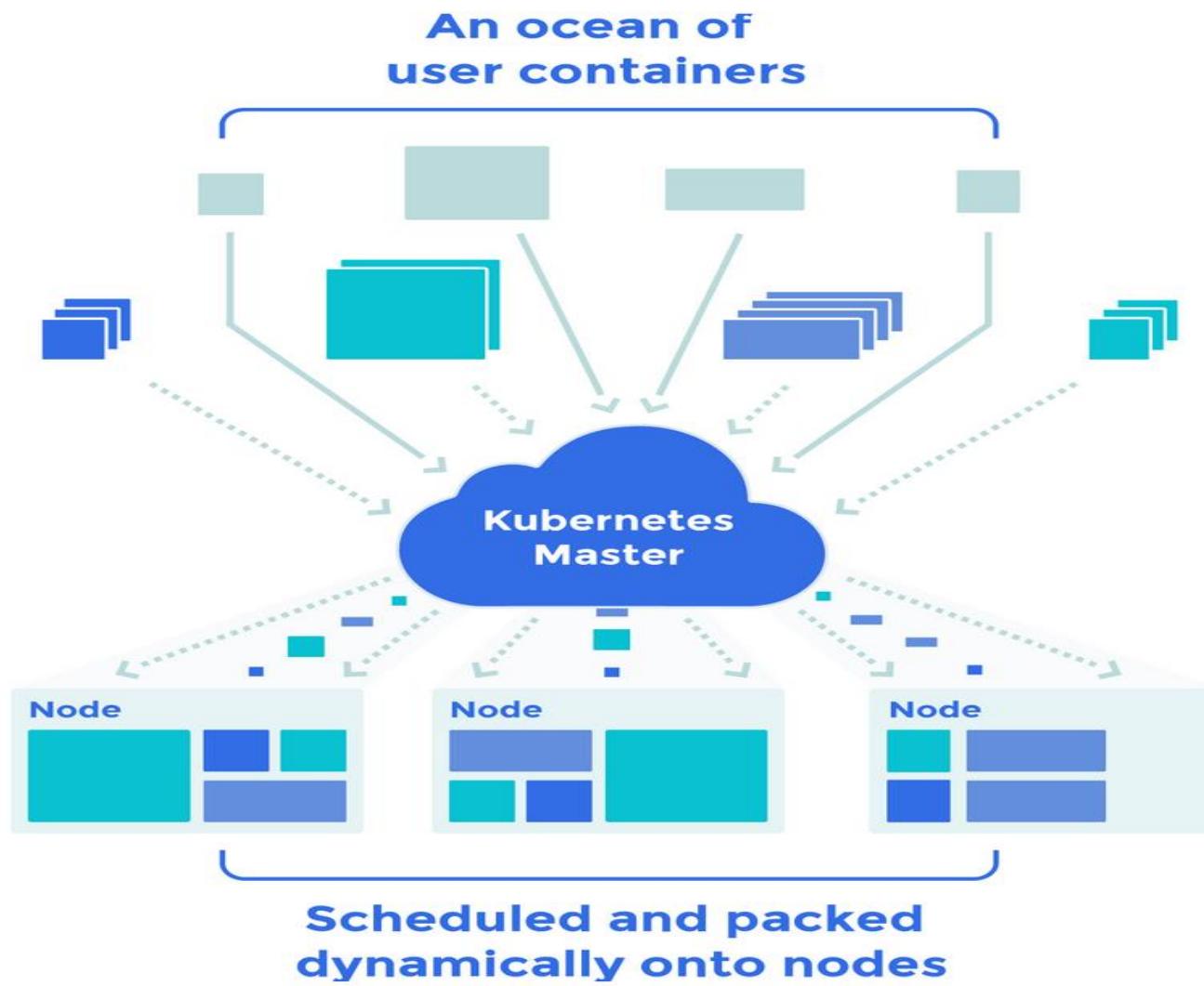


- Login to your repository use the following command with your Artifactory Cloud credentials.
- `docker login ${server-name}-${repo-name}.jfrog.io`
- Pull an image using the following command.
- `docker pull ${server-name}-${repo-name}.jfrog.io/<image name>`
- Push an image by first tagging it and then using the push command.
- `docker tag <image name> ${server-name}-${repo-name}.jfrog.io/<image name>`
- `docker push ${server-name}-${repo-name}.jfrog.io/<image name>`

Getting Started with Artifactory as a Docker Registry



- In this example, the Artifactory Cloud server is named **acme**.
- Start by creating a virtual Docker repository called dockerv2-virtual.
- Pull the hello-world image
- docker pull hello-world
- Login to repository dockerv2-virtual
- docker login acme-dockerv2-virtual.jfrog.io
- Tag the hello-world image
- docker tag hello-world acme-dockerv2-virtual.jfrog.io/hello-world
- Push the tagged hello-world image to dockerv2-virtual
- docker push acme-dockerv2-virtual.jfrog.io/hello-world





Docker Swarm

- Docker Swarm is Docker's native feature to support clustering of Docker machines.
- This enables multiple machines running Docker Engine to participate in a cluster, called Swarm.
- The Docker engines contributing to a Swarm are said to be running in Swarm mode.
- Machines enter into the Swarm mode by either initializing a new swarm or by joining an existing swarm.
- To the end user the swarm would seem like a single machine.
- A Docker engine participating in a swarm is called a node



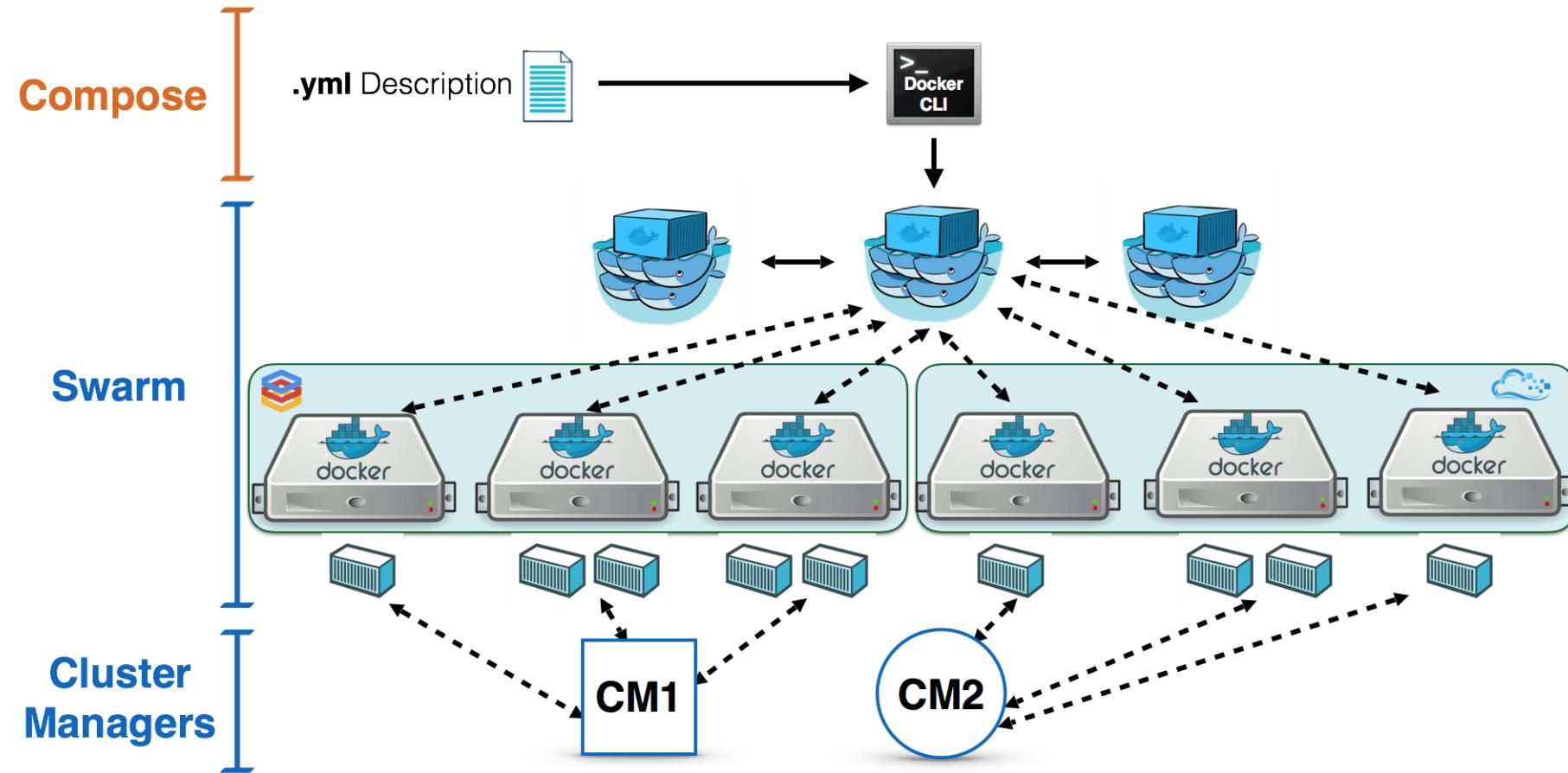
Docker Swarm

- A node can either be a manager node or a worker node.
- The manager node performs cluster management and orchestration while the worker nodes perform tasks allocated by the manager.
- A manager node itself, unless configured otherwise, is also a worker node.
- The central entity in the Docker Swarm infrastructure is called a service.
- A Docker swarm executes services. The user submits a service to the manager node to deploy and execute.
- A service is made up of many tasks. A task is the most basic work unit in a Swarm.
- A task is allocated to each worker node by the manager node.

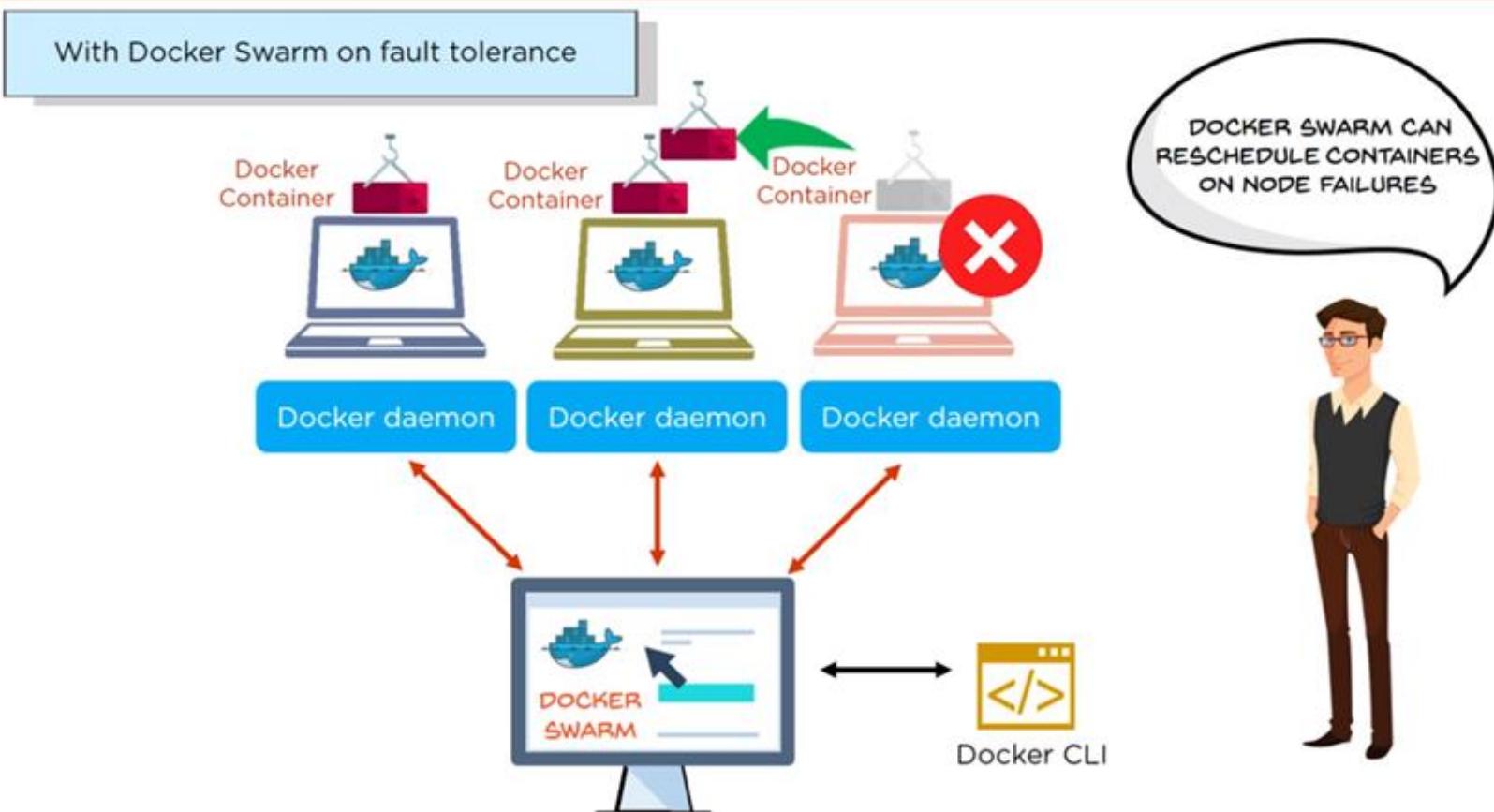
Docker Swarm



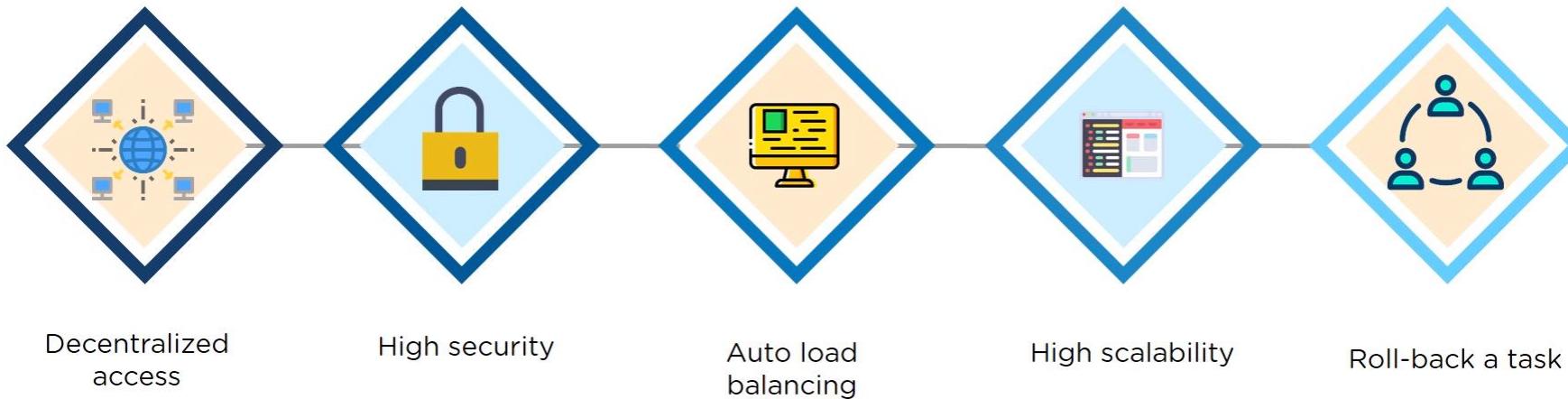
- The Docker ecosystem consists of tools from development to production deployment frameworks.
- In that list, docker swarm fits into cluster management.
- A mix of docker-compose, swarm, overlay network and a good service discovery tool such as etcd or consul can be used for managing a cluster of Docker containers.
- Docker swarm is still maturing in terms of functionalities when compared to other open-source container cluster management tools.
- Considering the vast docker contributors, it won't be so long for docker swarm to have all the best functionalities other tools possess.
- Docker has documented a good production plan for using docker swarm in production.



What is Docker Swarm?



Features of Docker Swarm



Docker Swarm

- In Swarm, containers are launched using services
- A service is a group of containers of the same image
- Services enables to scale your application
- Before you can deploy a service in Docker Swarm, you must have at least one node deployed
- There are two types of nodes in Docker Swarm

Manager node



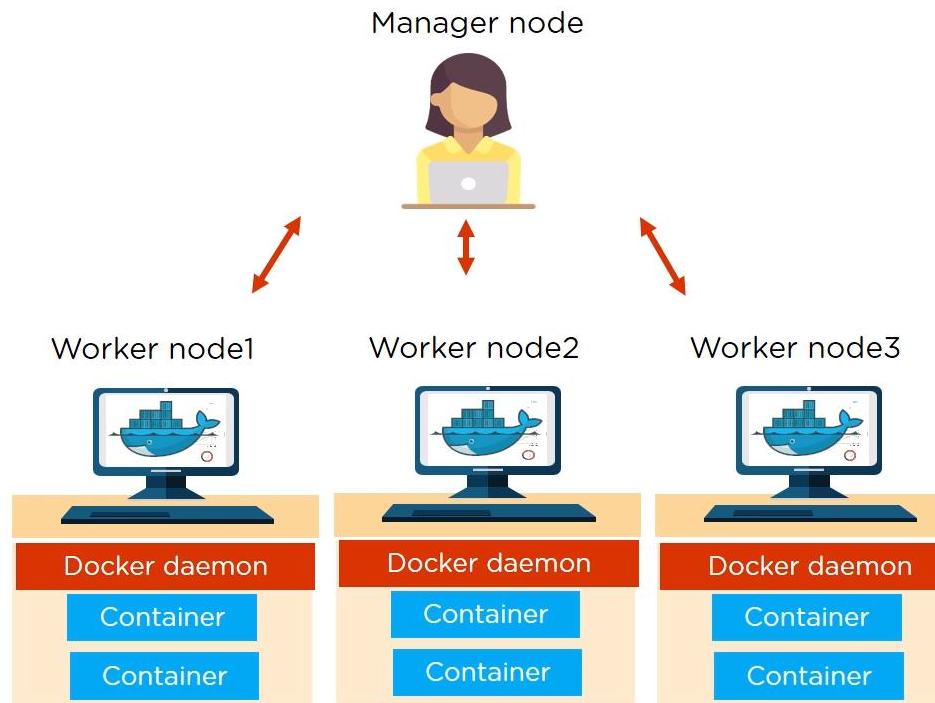
Worker node



Manager node
maintains cluster
management tasks

Worker nodes receive
and execute tasks from
manager node

Architecture of Docker Swarm



Docker Swarm Steps

```
C:\Users\Balasubramaniam>docker swarm init
Swarm initialized: current node (79assnkjjut36pv3blv7tb1lz) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-47gh210k487vjtj5diybgij1l3mmkxt6qxbz6m725yod5z8poq7-3idk04u6gonz4elxrf0vde644 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

C:\Users\Balasubramaniam>
```

```
C:\Users\Balasubramaniam> docker node ls
ID                  HOSTNAME        STATUS        AVAILABILITY   MANAGER STATUS      ENGINE VERSION
aw1gnnexmdexbwncvlj72c723 *  docker-desktop  Ready       Active        Leader           19.03.13

C:\Users\Balasubramaniam>
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service create --name eurekaswarm -p 8761:8761 eureka-app:latest
image eureka-app:latest could not be accessed on a registry to record
its digest. Each node will access eureka-app:latest independently,
possibly leading to different nodes running different
versions of the image.

0uw58z9g5vbjl13zzzjop10k9
overall progress: 1 out of 1 tasks
1/1: running  [=====>]
verify: Service converged

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
ID           NAME      MODE      REPLICAS      IMAGE      PORTS
0uw58z9g5vbj  eurekaswarm  replicated  1/1          eureka-app:latest  *:8761->8761/tcp

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service scale eurekaswarm=3
eurekaswarm scaled to 3
overall progress: 3 out of 3 tasks
1/3: running  [=====>]
2/3: running  [=====>]
3/3: running  [=====>]
verify: Service converged

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
ID           NAME      MODE      REPLICAS      IMAGE      PORTS
0uw58z9g5vbj  eurekaswarm  replicated  3/3          eureka-app:latest  *:8761->8761/tcp

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
```



What is Kubernetes?

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.
- It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Google open-sourced the Kubernetes project in 2014.
- Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

What Is Kubernetes? An Introduction To Container Orchestration Tool



- **What Is Kubernetes?**
- Kubernetes is an open-source container management (orchestration) tool. Its container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Download from
- <https://github.com/kubernetes/minikube>
- Under curl - windows
- <https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl>

What is Kubernetes



What Is Kubernetes? An Introduction To Container Orchestration Tool



- **Why Use Kubernetes?**
- Companies out there maybe using Docker or Rocket or maybe simply Linux containers for containerizing their applications. But, whatever it is, they use it on a massive scale. They don't stop at using 1 or 2 containers in Prod. But rather, **10's or 100's** of containers for load balancing the traffic and ensuring high availability.
- .



Why use Kubernetes?

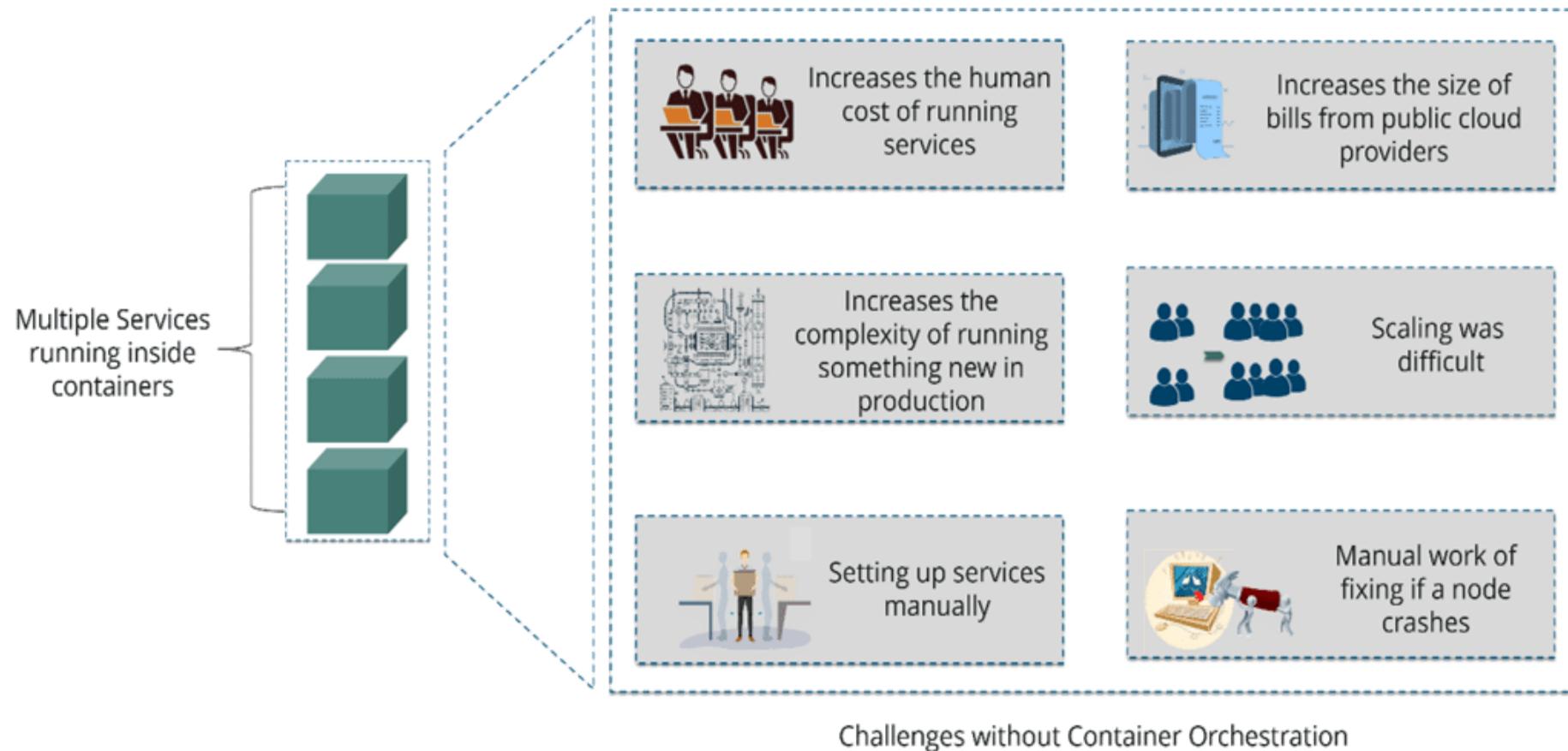
- Kubernetes can run on-premises bare metal, OpenStack, public clouds Google, Azure, AWS, etc.
- Helps you to avoid vendor lock issues as it can use any vendor-specific APIs or services except where Kubernetes provides an abstraction, e.g., load balancer and storage.
- Containerization using kubernetes allows package software to serve these goals. It will enable applications that need to be released and updated without any downtime.
- Kubernetes allows you to assure those containerized applications run where and when you want and helps you to find resources and tools which you want to work.



What task are performed by Kubernetes?

- Kubernetes is the Linux kernel which is used for distributed systems.
- It helps you to abstract the underlying hardware of the nodes(servers) and offers a consistent interface for applications that consume the shared pool of resources.

What is the need for Container Orchestration?



Kubernetes vs Docker Swarm

Features	Kubernetes	Docker Swarm
Installation & Cluster Config	Setup is very complicated, but once installed cluster is robust.	Installation is very simple, but the cluster is not robust.
GUI	GUI is the Kubernetes Dashboard.	There is no GUI.
Scalability	Highly scalable and scales fast.	Highly scalable and scales 5x faster than Kubernetes.
Auto-scaling	Kubernetes can do auto-scaling.	Docker swarm cannot do auto-scaling.
Load Balancing	Manual intervention needed for load balancing traffic between different containers and pods.	Docker swarm does auto load balancing of traffic between containers in the cluster.
Rolling Updates & Rollbacks	Can deploy rolling updates and does automatic rollbacks.	Can deploy rolling updates, but not automatic rollback.
DATA Volumes	Can share storage volumes only with the other containers in the same pod.	Can share storage volumes with any other container.
Logging & Monitoring	In-built tools for logging and monitoring.	3rd party tools like ELK stack should be used for logging and monitoring.

What are the features of Kubernetes?

01

Automated Scheduling

Kubernetes provides advanced scheduler to launch container on cluster nodes

02

Self Healing Capabilities

Rescheduling, replacing and restarting the containers which are died.

03

Automated rollouts and rollback

Kubernetes supports rollouts and rollbacks for the desired state of the containerized application

04

Horizontal Scaling and Load Balancing

Kubernetes can scale up and scale down the application as per the requirements



Features of Kubernetes

- Automated Scheduling
- Self-Healing Capabilities
- Automated rollouts & rollback
- Horizontal Scaling & Load Balancing
- Offers environment consistency for development, testing, and production
- Infrastructure is loosely coupled to each component can act as a separate unit
- Provides a higher density of resource utilization
- Offers enterprise-ready features
- Application-centric management
- Auto-scalable infrastructure
- You can create predictable infrastructure



Kubernetes Basics

- Cluster:
- It is a collection of hosts(servers) that helps you to aggregate their available resources. That includes ram, CPU, ram, disk, and their devices into a usable pool.
- Master:
- The master is a collection of components which make up the control panel of Kubernetes. These components are used for all cluster decisions. It includes both scheduling and responding to cluster events.



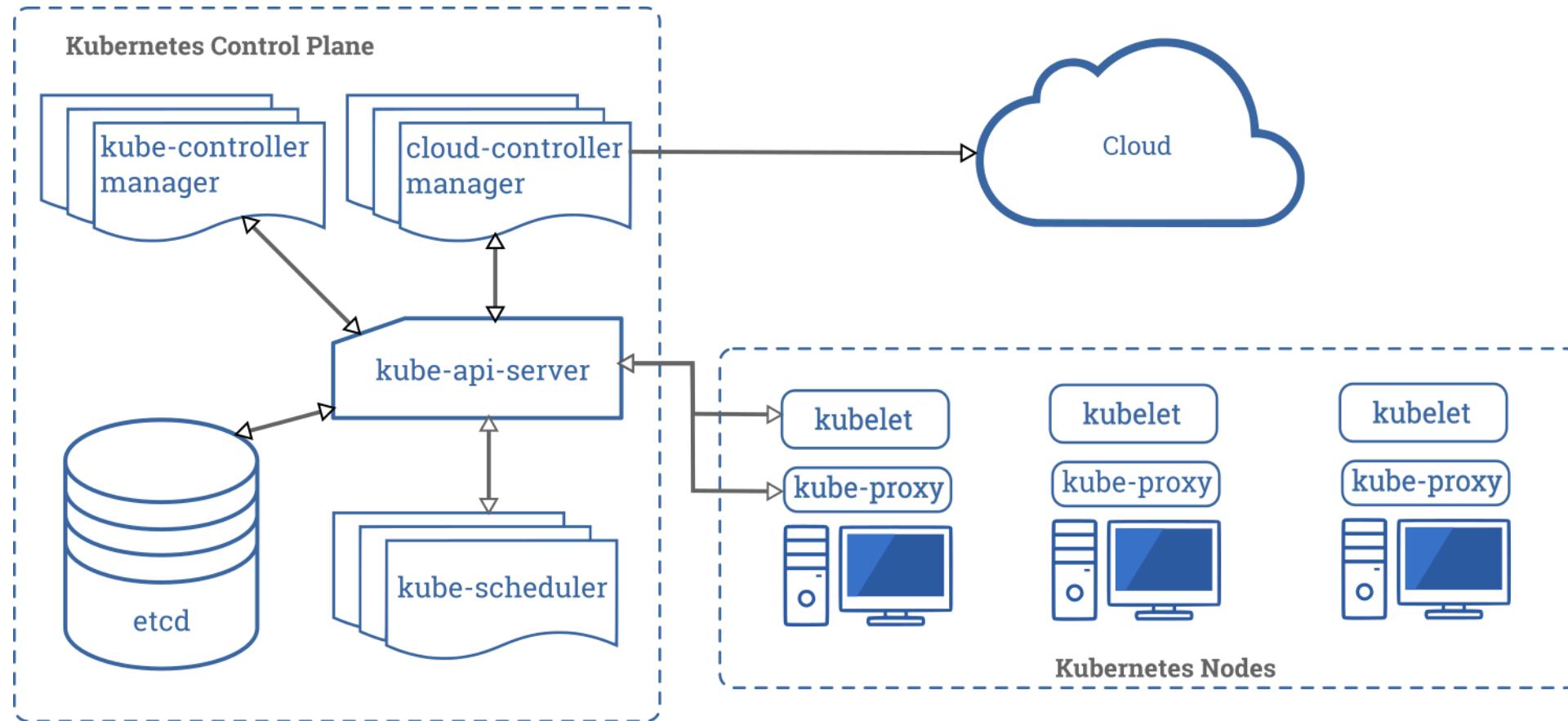
Kubernetes Basics

- Node:
- It is a single host which is capable of running on a physical or virtual machine. A node should run both kube-proxy, minikube, and kubelet which are considered as a part of the cluster.
- Namespace:
- It is a logical cluster or environment. It is a widely used method which is used for scoping access or dividing a cluster.

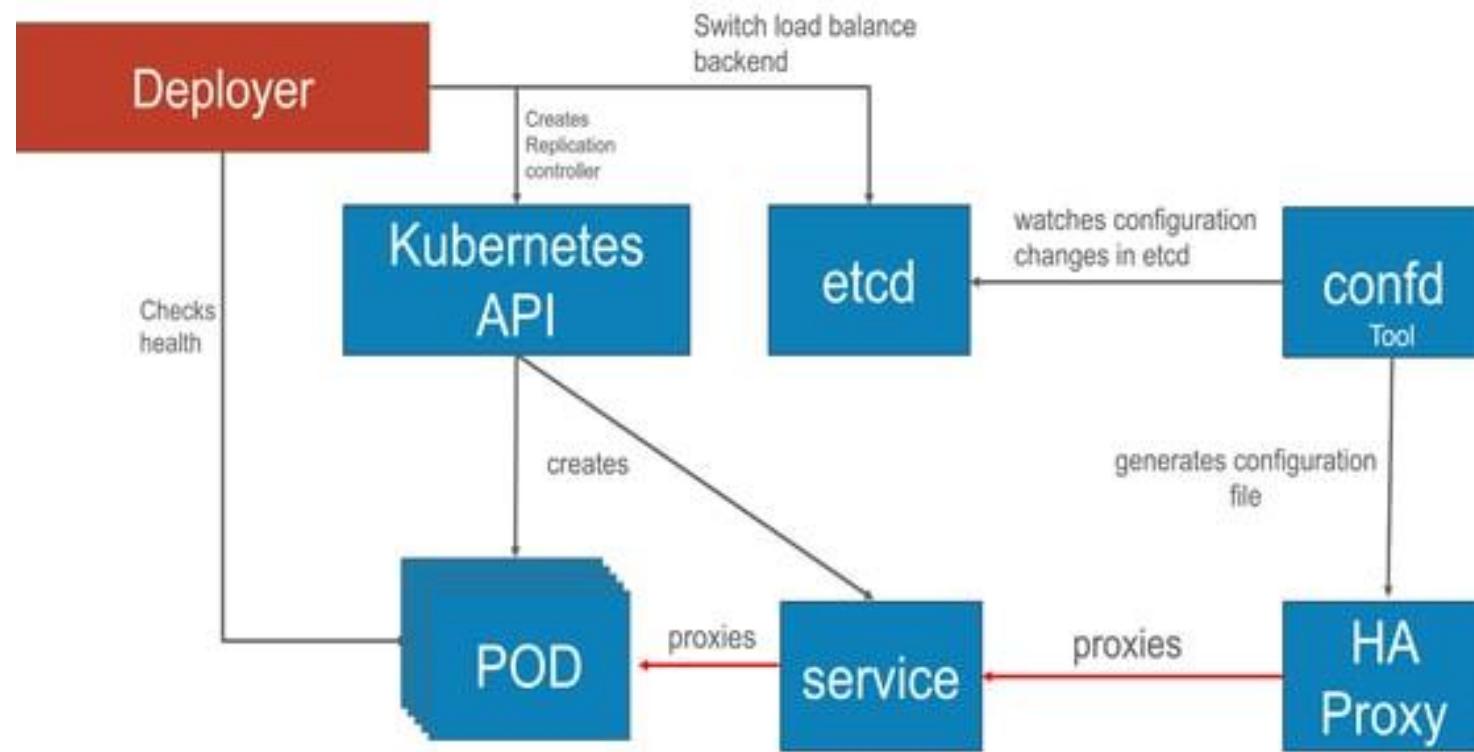


Kubernetes Architecture and Components

- Kubernetes has a decentralized architecture that does not handle tasks sequentially.
- It functions based on a declarative model and implements the concept of a ‘desired state.’



Kubernetes architecture



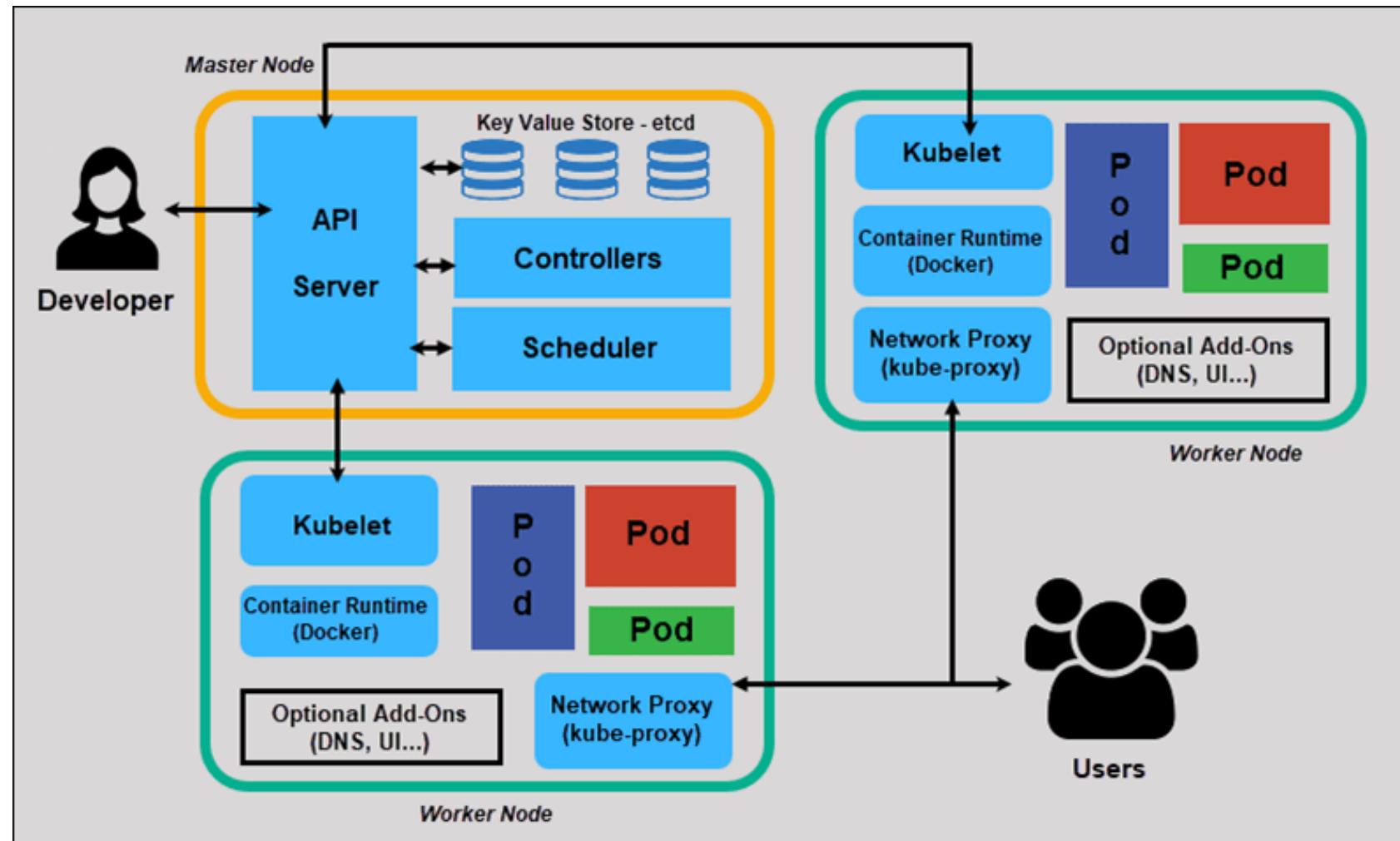


Kubernetes Architecture and Components

- These steps illustrate the basic Kubernetes process:
- An administrator creates and places the desired state of an application into a manifest file.
- The file is provided to the Kubernetes API Server using a CLI or UI.
- Kubernetes' default command-line tool is called kubectl.
- Kubernetes stores the file (an application's desired state) in a database called the Key-Value Store (etcd).
- Kubernetes then implements the desired state on all the relevant applications within the cluster.
- Kubernetes continuously monitors the elements of the cluster to make sure the current state of the application does not vary from the desired state.



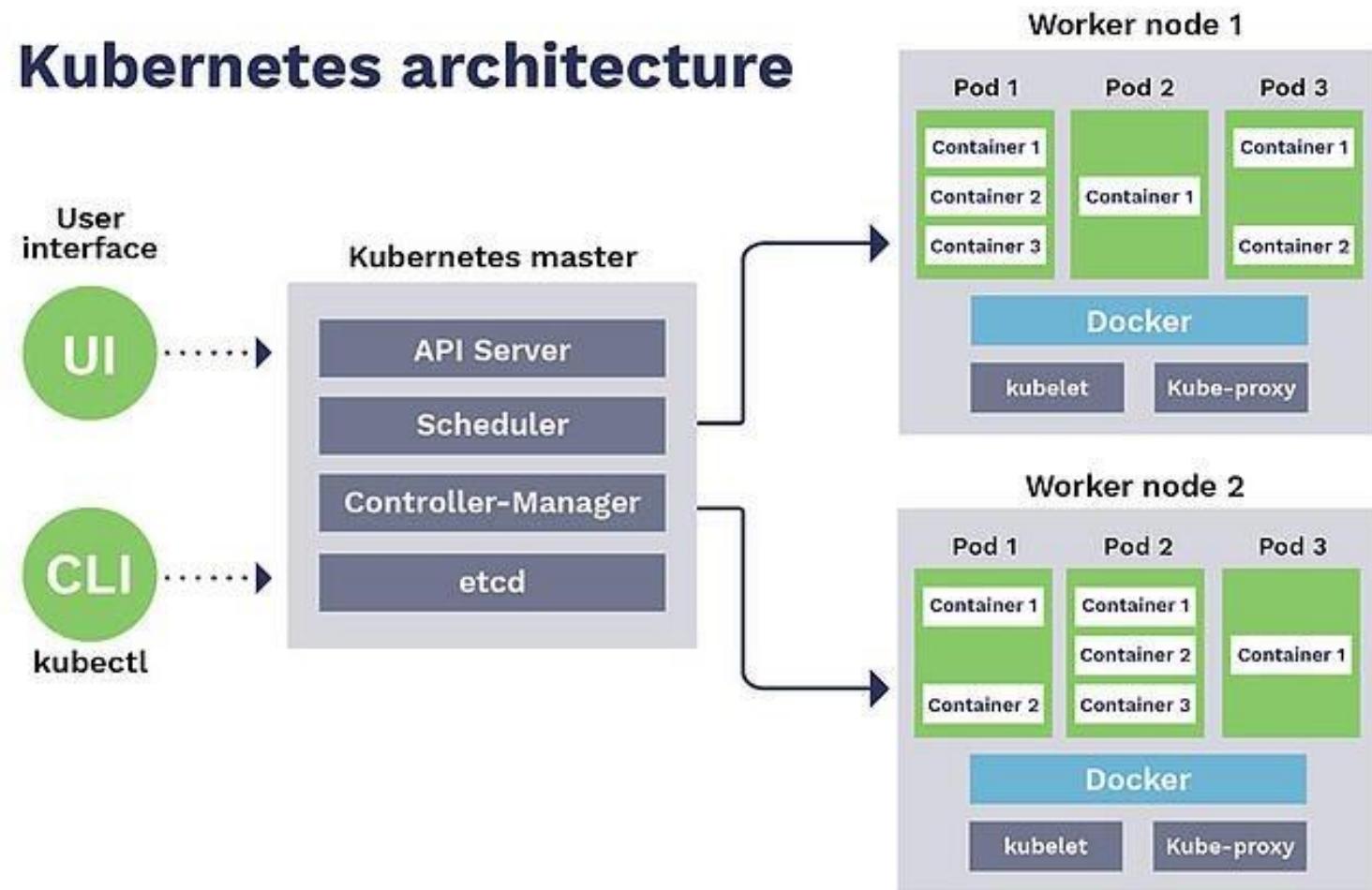
Kubernetes Architecture and Components



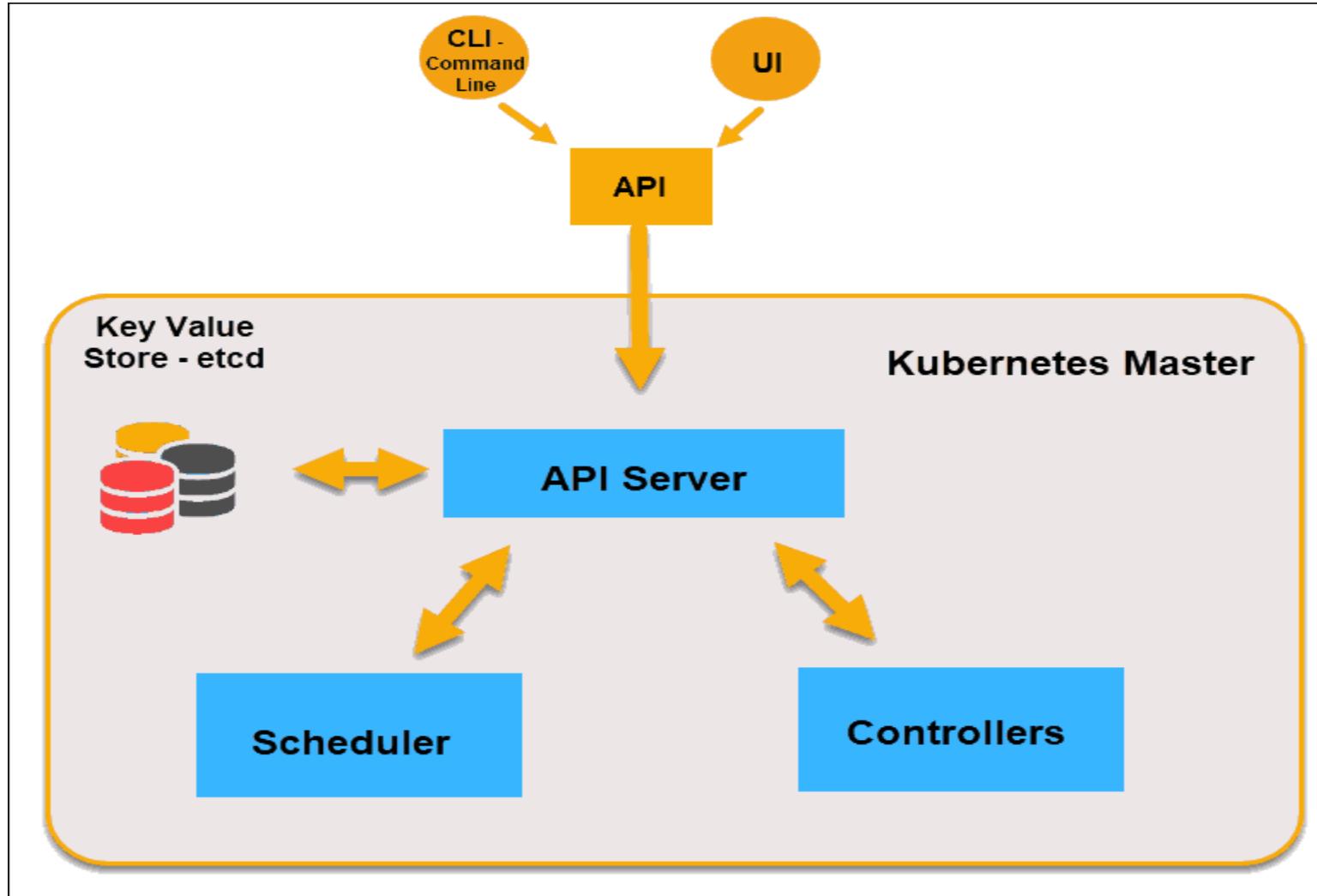


Kubernetes Architecture and Components

Kubernetes architecture



What is Master Node in Kubernetes Architecture?





API Server

- The API Server is the front-end of the control plane and the only component in the control plane that we interact with directly.
- Internal system components, as well as external user components, all communicate via the same API.



Key-Value Store (etcd)

- The Key-Value Store, also called etcd, is a database Kubernetes uses to back-up all cluster data.
- It stores the entire configuration and state of the cluster.
- The Master node queries etcd to retrieve parameters for the state of the nodes, pods, and containers.



Controller

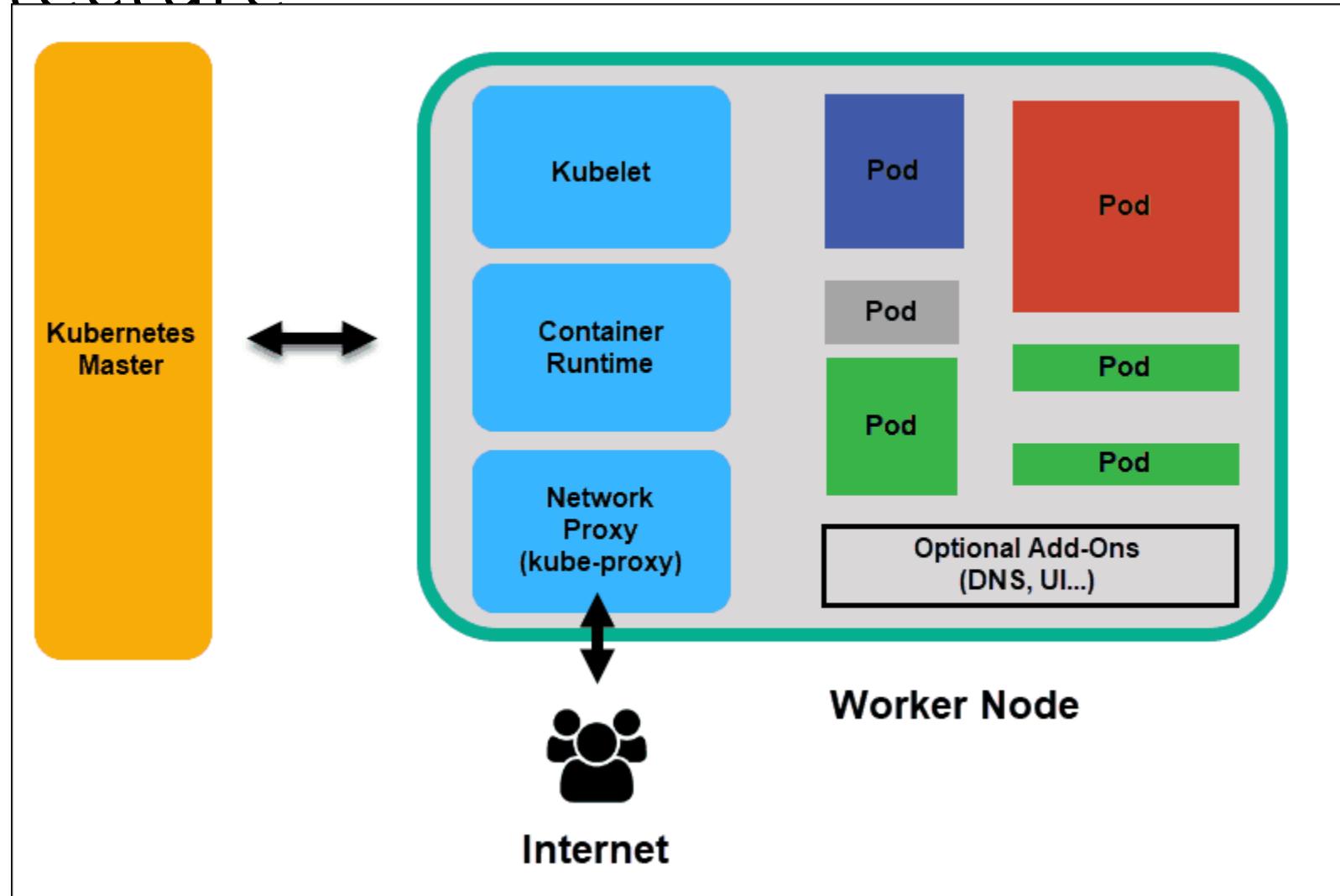
- The role of the Controller is to obtain the desired state from the API Server.
- It checks the current state of the nodes it is tasked to control, and determines if there are any differences, and resolves them, if any.

Scheduler



- A **Scheduler** watches for new requests coming from the API Server and assigns them to healthy nodes.
- It ranks the quality of the nodes and deploys pods to the best-suited node.
- If there are no suitable nodes, the pods are put in a pending state until such a node appears.

What is Worker Node in Kubernetes Architecture





Kubelet

- The kubelet runs on every node in the cluster.
- It is the principal Kubernetes agent.
- By installing kubelet, the node's CPU, RAM, and storage become part of the broader cluster.
- It watches for tasks sent from the API Server, executes the task, and reports back to the Master.
- It also monitors pods and reports back to the control panel if a pod is not fully functional.
- Based on that information, the Master can then decide how to allocate tasks and resources to reach the desired state.



Container Runtime

- The container runtime pulls images from a container image registry and starts and stops containers.
- A 3rd party software or plugin, such as Docker, usually performs this function.



Kube-proxy

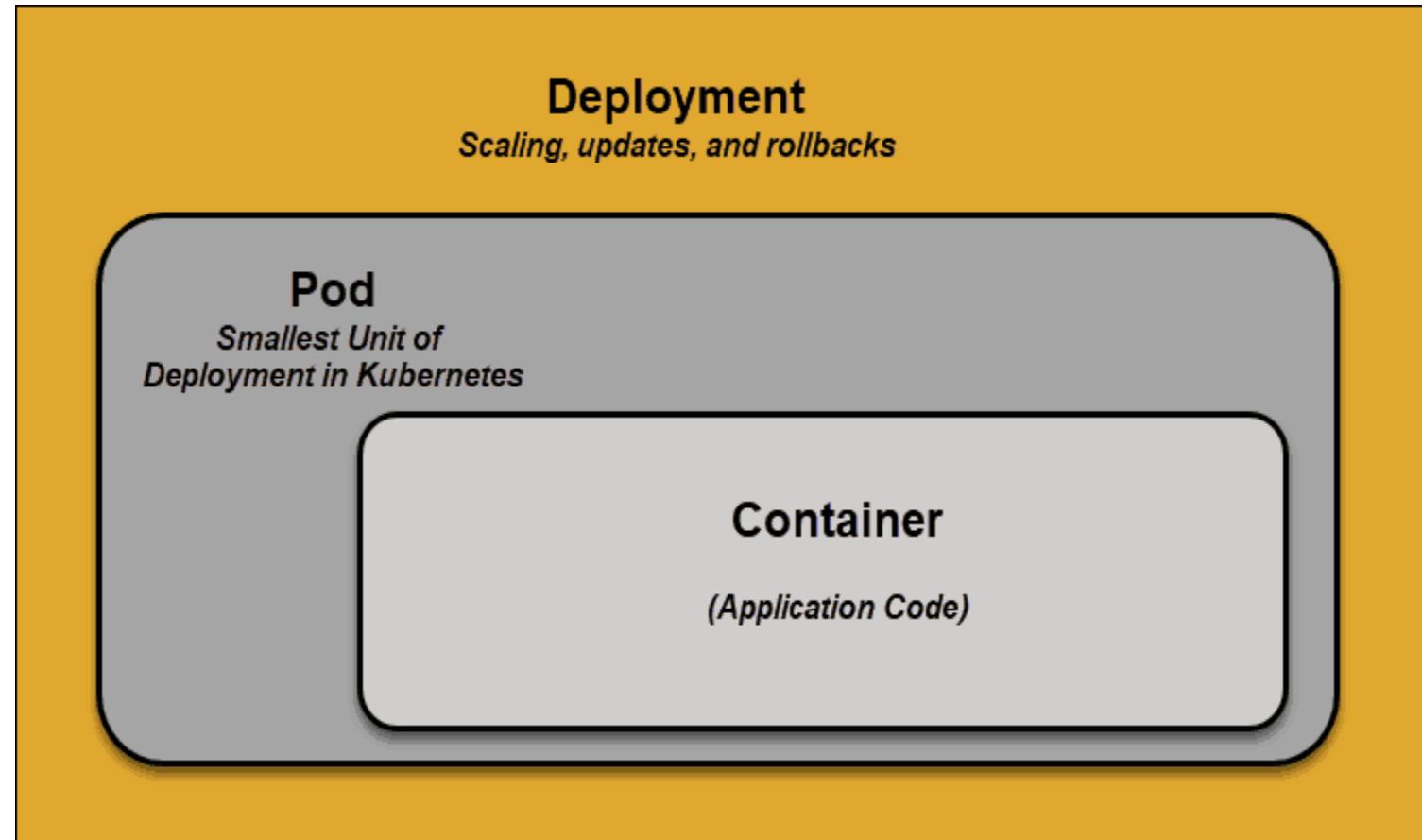
- The kube-proxy makes sure that each node gets its IP address, implements local iptables and rules to handle routing and traffic load-balancing.

Pod



- A pod is the smallest element of scheduling in Kubernetes. Without it, a container cannot be part of a cluster. If you need to scale your app, you can only do so by adding or removing pods.
- The pod serves as a ‘wrapper’ for a single container with the application code. Based on the availability of resources, the Master schedules the pod on a specific node and coordinates with the container runtime to launch the container

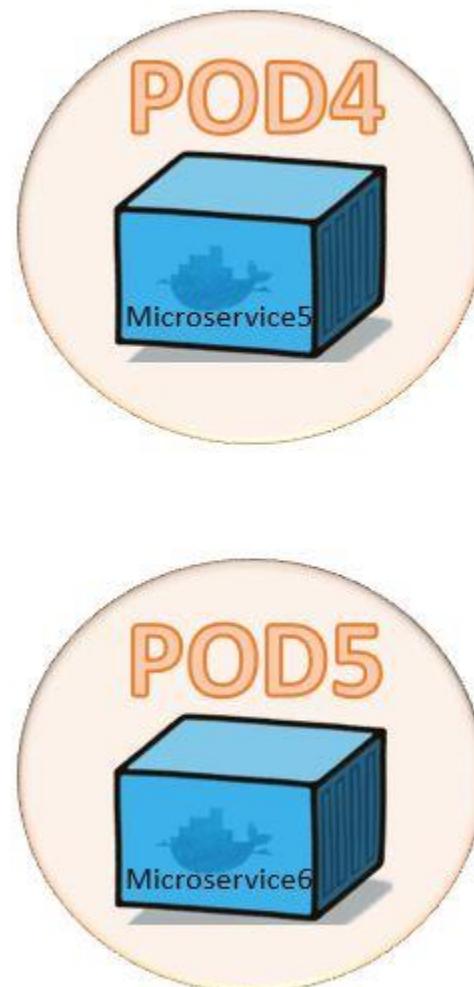
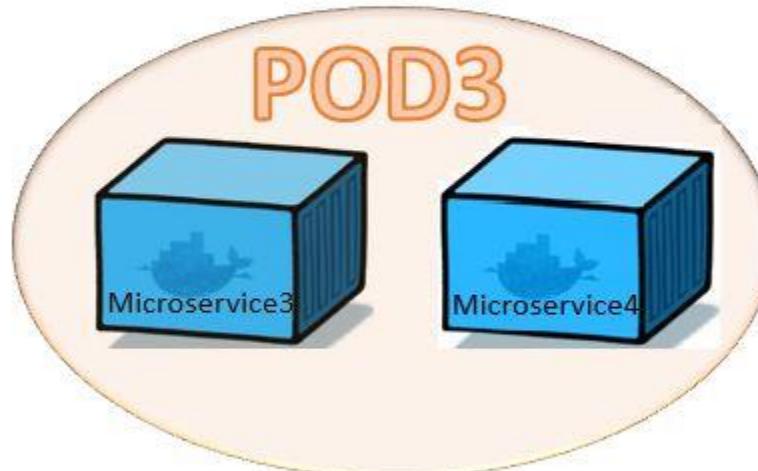
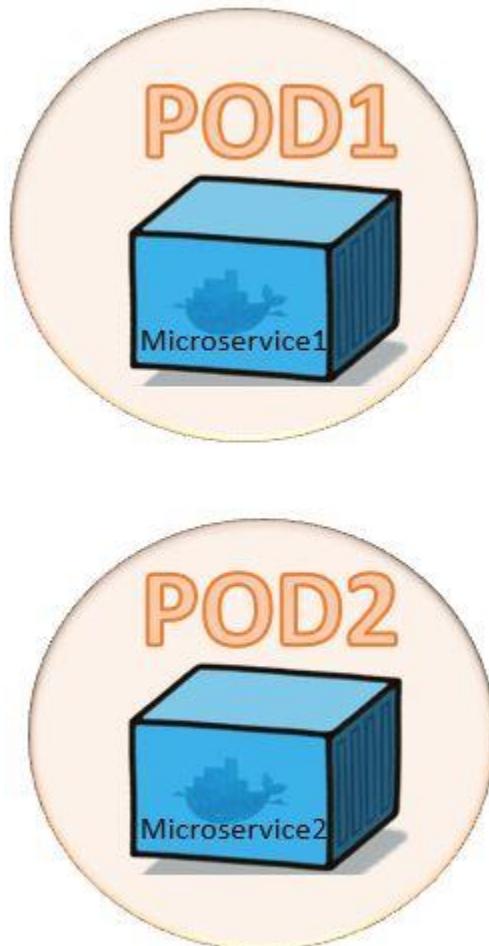
Pod



Pod



- In instances where pods unexpectedly fail to perform their tasks, Kubernetes does not attempt to fix them.
- Instead, it creates and starts a new pod in its place.
- This new pod is a replica, except for the DNS and IP address.
- This feature has had a profound impact on how developers design applications.
- Due to the flexible nature of Kubernetes architecture, applications no longer need to be tied to a particular instance of a pod.
- Instead, applications need to be designed so that an entirely new pod, created anywhere within the cluster, can seamlessly take its place. To assist with this process, Kubernetes uses services





Single Pod with multiple containers

- apiVersion: v1
- kind: Pod
- metadata:
- name: mc1
- spec:
- volumes:
- - name: html
- emptyDir: {}
- containers:
- - name: 1st
- image: nginx



Single Pod with multiple containers

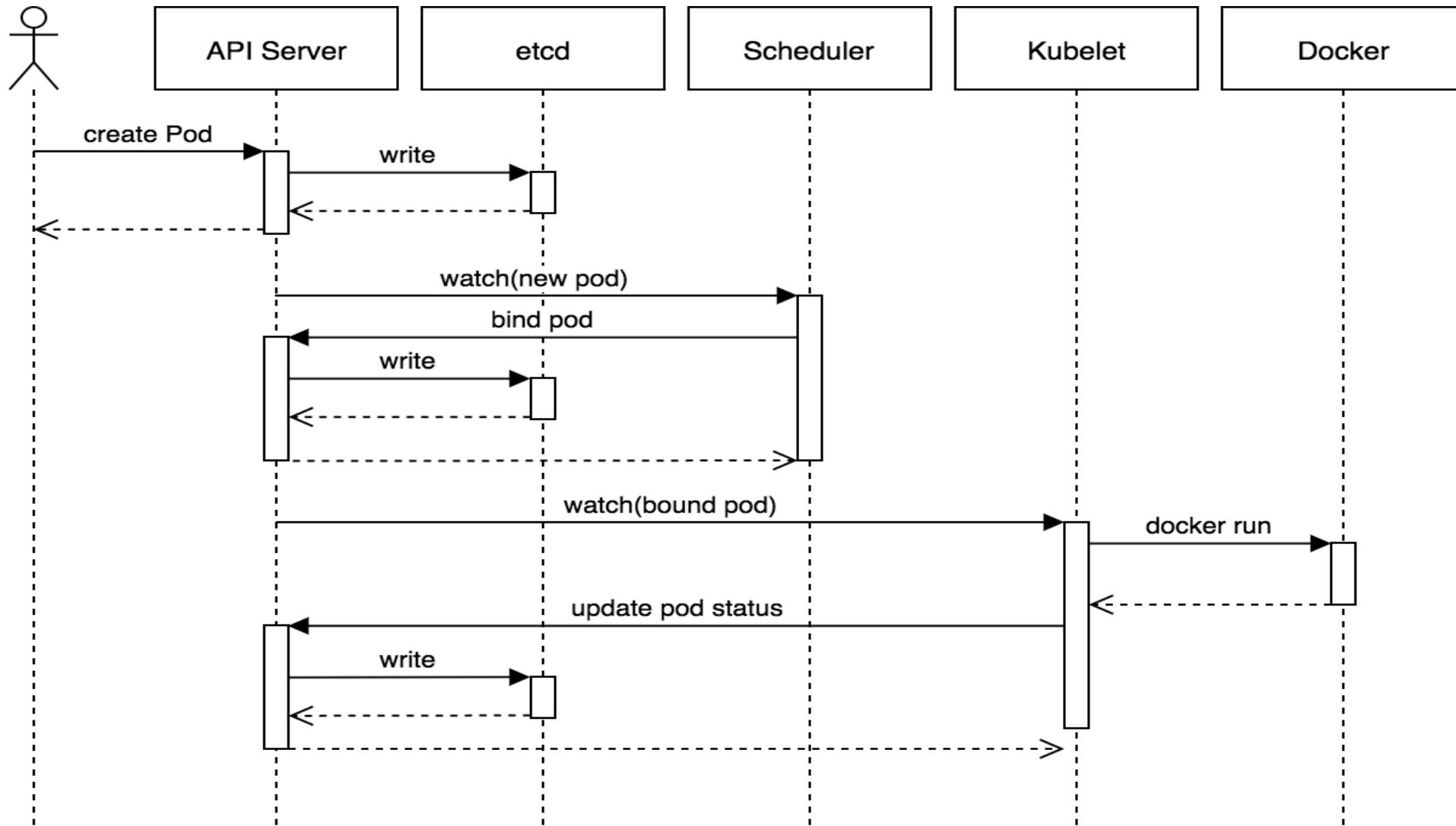
- volumeMounts:
- - name: html
- mountPath: /usr/share/nginx/html
- - name: 2nd
- image: debian
- volumeMounts:
- - name: html
- mountPath: /html
- command: ["/bin/sh", "-c"]
- args:
- - while true; do
- date >> /html/index.html;
- sleep 1;
- done



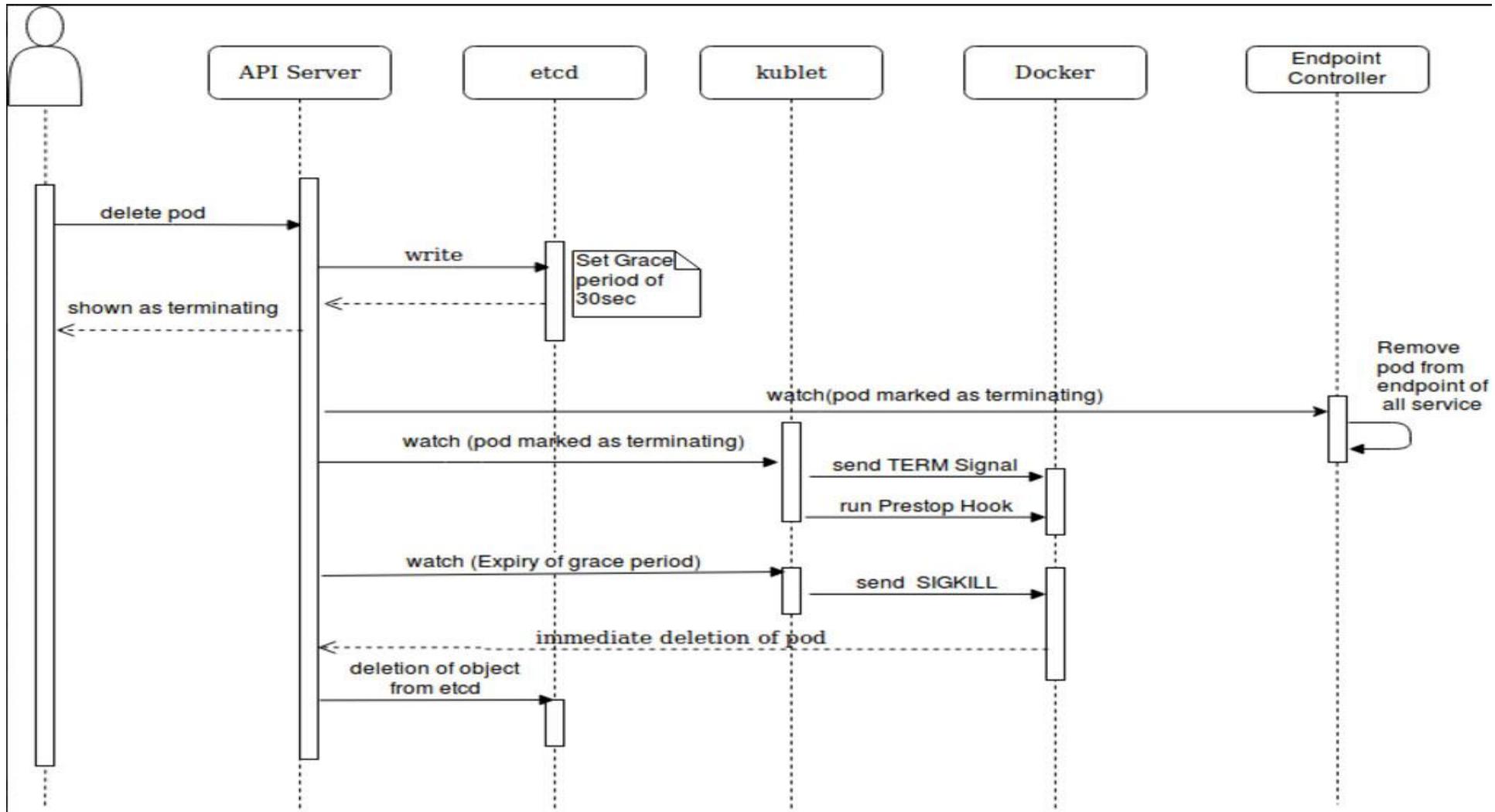
States of a Pod

- Through its lifecycle, a Pod can attain following states:
- Pending: The pod is accepted by the Kubernetes system but its container(s) is/are not created yet.
- Running: The pod is scheduled on a node and all its containers are created and at-least one container is in Running state.
- Succeeded: All container(s) in the Pod have exited with status 0 and will not be restarted.
- Failed: All container(s) of the Pod have exited and at least one container has returned a non-zero status.
- CrashLoopBackoff: The container fails to start and is tried again and again.

Birth of a Pod



Pod Termination





Kubernetes Services

- Pods are not constant. One of the best features Kubernetes offers is that non-functioning pods get replaced by new ones automatically.
- However, these new pods have a different set of IPs. It can lead to processing issues, and IP churn as the IPs no longer match. If left unattended, this property would make pods highly unreliable.
- Services are introduced to provide reliable networking by bringing stable IP addresses and DNS names to the unstable world of pods.
- By controlling traffic coming and going to the pod, a Kubernetes service provides a stable networking endpoint – a fixed IP, DNS, and port.
- Through a service, any pod can be added or removed without the fear that basic network information would change in any way.



How Do Kubernetes Services Work?

- Pods are associated with services through key-value pairs called labels and selectors.
- A service automatically discovers a new pod with labels that match the selector.
- This process seamlessly adds new pods to the service, and at the same time, removes terminated pods from the cluster.
- For example, if the desired state includes three replicas of a pod and a node running one replica fails, the current state is reduced to two pods.
- Kubernetes observes that the desired state is three pods.
- It then schedules one new replica to take the place of the failed pod and assigns it to another node in the cluster.



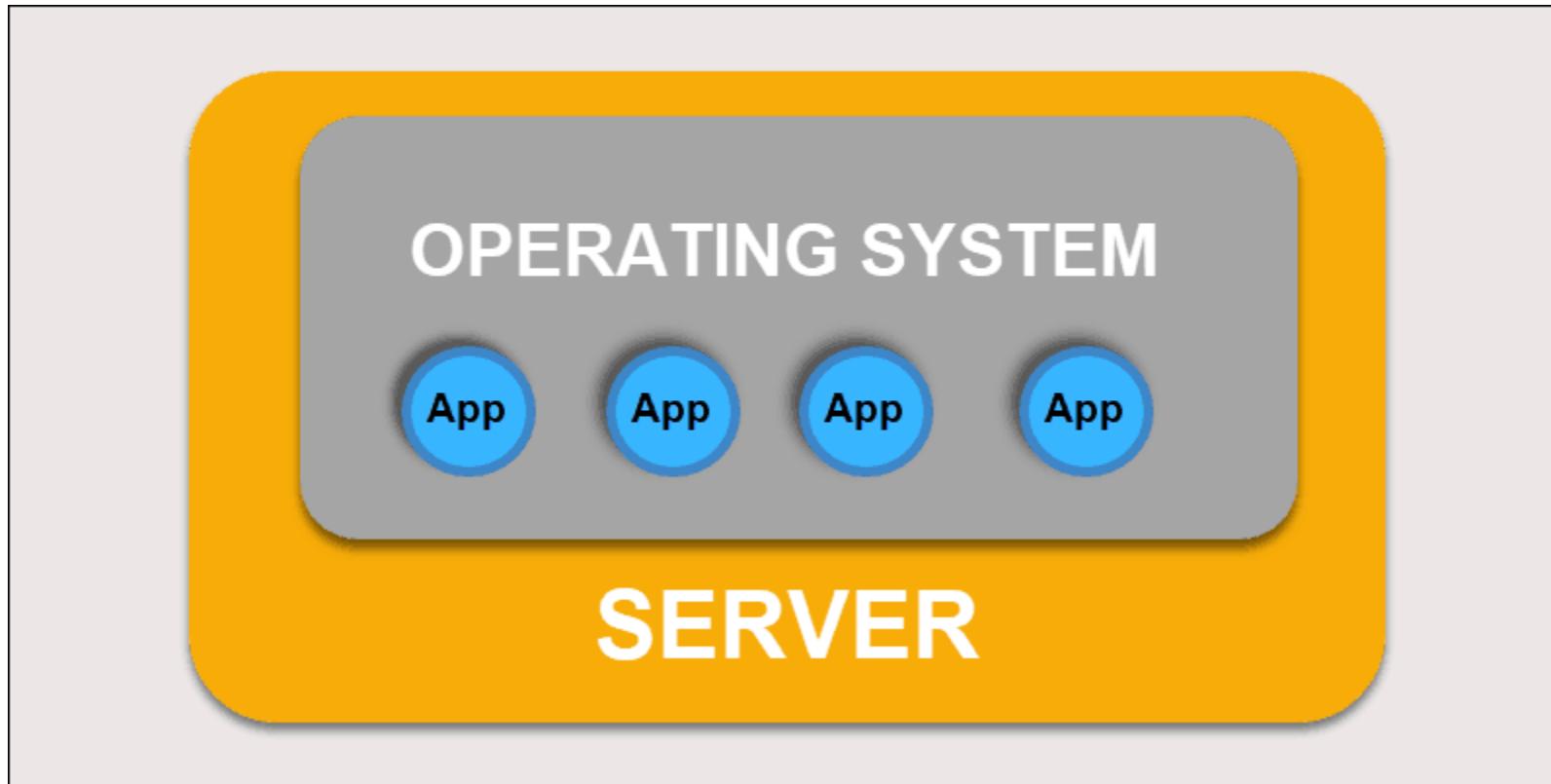
How Do Kubernetes Services Work?

- The same would apply when updating or scaling the application by adding or removing pods.
- Once we update the desired state, Kubernetes notices the discrepancy and adds or removes pods to match the manifest file.
- The Kubernetes control panel records, implements, and runs background reconciliation loops that continuously check to see if the environment matches user-defined requirements.



What is Container Deployment?

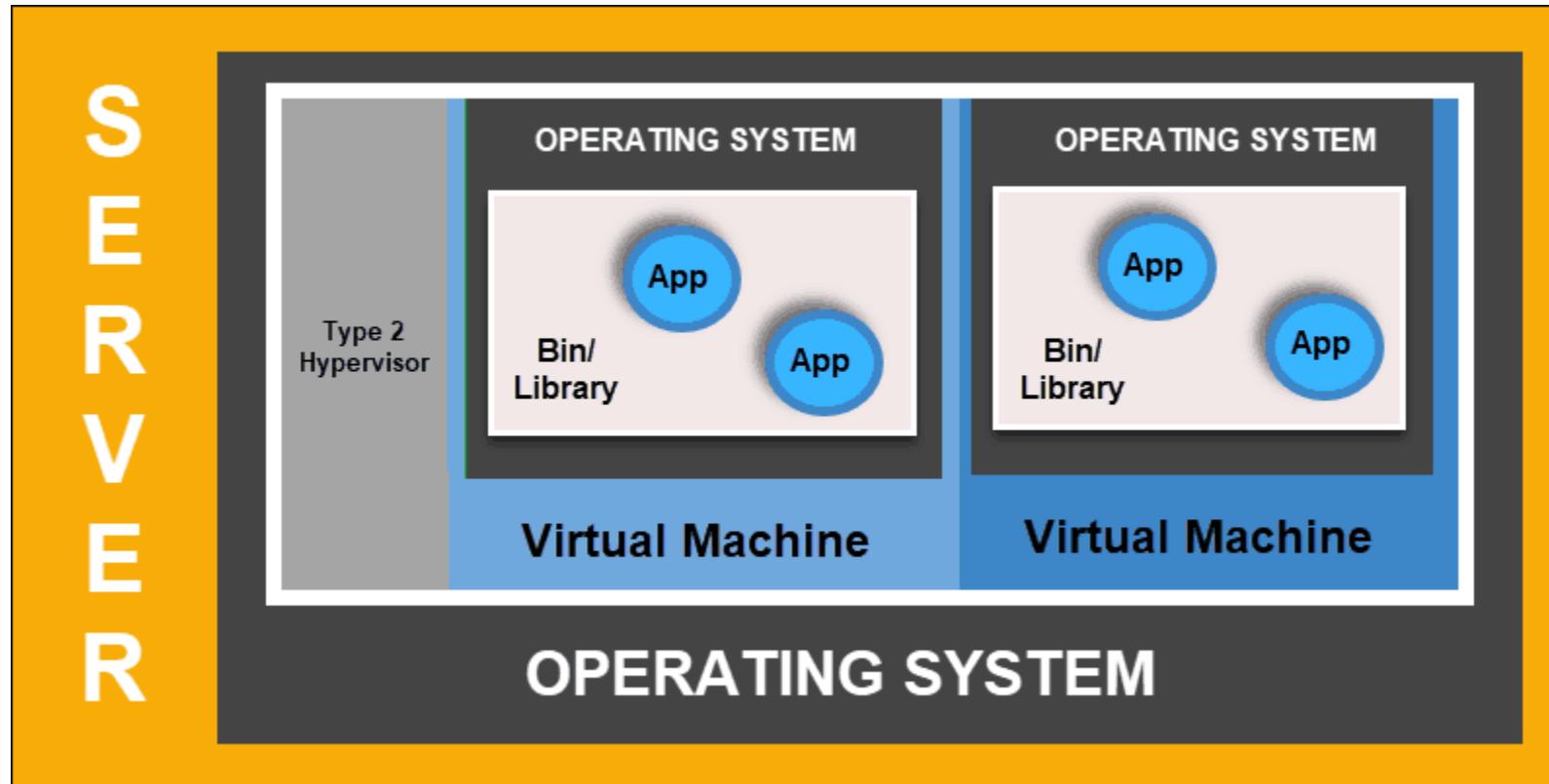
Traditional Deployment





What is Container Deployment?

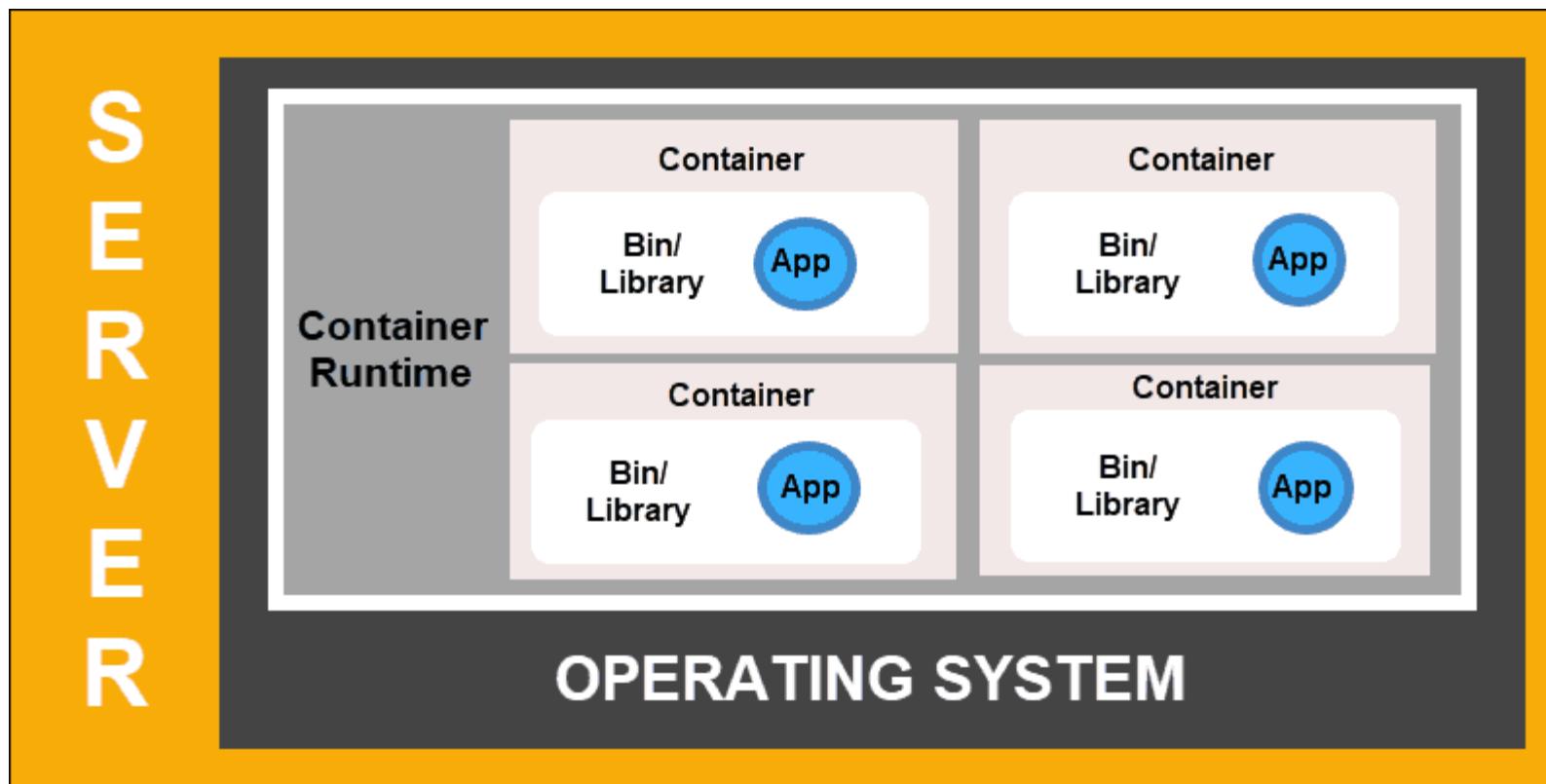
virtualized Deployment





What is Container Deployment?

ContainerDeployment





Docker and Kubernetes - Python

```
C:\ Administrator: Command Prompt
=> => transferring context: 597B
=> [2/5] RUN apk add --no-cache python3 &&      python3 -m ensurepip &&      rm -r /usr/lib/python*/ensurepip &&      pip3 instal 0.0s
=> [3/5] COPY . /app                           21.7s
=> [4/5] WORKDIR /app                         0.1s
=> [5/5] ERROR [5/5] RUN pip3 install -r requirements.txt    0.1s
-----                                         1.6s
> [5/5] RUN pip3 install -r requirements.txt:
#10 1.185 ERROR: Could not open requirements file: [Errno 2] No such file or directory: 'requirements.txt'
-----
executor failed running [/bin/sh -c pip3 install -r requirements.txt]: exit code: 1

G:\Local disk\ Docker\python> docker build -f dockerfile -t k8flask .
[+] Building 13.5s (10/10) FINISHED
=> [internal] load build definition from dockerfile          0.0s
=> => transferring dockerfile: 32B                          0.0s
=> [internal] load .dockerignore                            0.0s
=> => transferring context: 2B                            0.0s
=> [internal] load metadata for docker.io/library/alpine:latest 1.6s
=> [internal] load build context                          0.0s
=> => transferring context: 116B                         0.0s
=> [1/5] FROM docker.io/library/alpine@sha256:69e70a79f2d41ab5d637de98c1e0b055206ba40a8145e7bddb55ccc04e13cf8f 0.0s
=> CACHED [2/5] RUN apk add --no-cache python3 &&      python3 -m ensurepip &&      rm -r /usr/lib/python*/ensurepip &&      pip3 0.0s
=> [3/5] COPY . /app                                     0.0s
=> [4/5] WORKDIR /app                                    0.0s
=> [5/5] RUN pip3 install -r requirements.txt           10.5s
=> exporting to image                                    1.0s
=> => exporting layers                                  1.0s
=> => writing image sha256:15ef33837836486ccc54f2ef7bc0b69b76b19b84fe3f72a1fab5a1a54f268ec9 0.0s
=> => naming to docker.io/library/k8flask              0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

G:\Local disk\ Docker\python>
```



11:17 21/04/2021 ENG 19

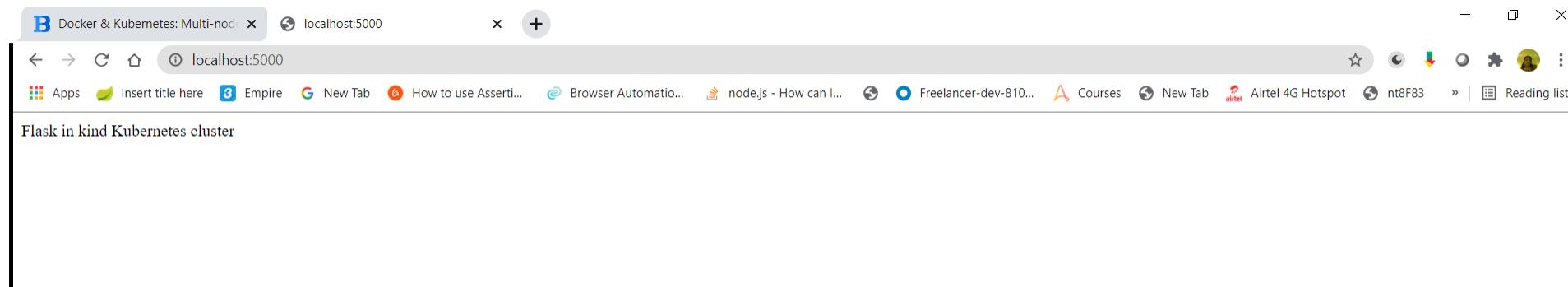


Docker and Kubernetes - Python

- docker run -p 5000:8087 k8flask:latest

```
c:\ Administrator: Command Prompt - docker run -p 5000:8087 k8flask:latest
G:\Local disk\ Docker\python>docker run -p 5000:8087 k8-flask:latest
Unable to find image 'k8-flask:latest' locally

G:\Local disk\ Docker\python>docker run -p 5000:8087 k8flask:latest
 * Running on http://0.0.0.0:8087/ (Press CTRL+C to quit)
172.17.0.1 - - [21/Apr/2021 05:49:46] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [21/Apr/2021 05:49:47] "GET /favicon.ico HTTP/1.1" 404 -
```





Docker and Kubernetes - Python

- docker tag k8flask:latest
eswaribala/ecommercecgirepo:p1
- Docker push eswaribala/ecommercecgirepo:p1

```
C:\> Administrator: Command Prompt
G:\Local disk\ Docker\python>docker run -p 5000:8087 k8-flask:latest
Unable to find image 'k8-flask:latest' locally

G:\Local disk\ Docker\python>docker run -p 5000:8087 k8flask:latest
 * Running on http://0.0.0.0:8087/ (Press CTRL+C to quit)
172.17.0.1 - - [21/Apr/2021 05:49:46] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [21/Apr/2021 05:49:47] "GET /favicon.ico HTTP/1.1" 404 -

G:\Local disk\ Docker\python>docker tag k8flask:latest eswaribala/ecommercecgirepo:p1

G:\Local disk\ Docker\python>docker push eswaribala/ecommercecgirepo:p1
The push refers to repository [docker.io/eswaribala/ecommercecgirepo]
d713d0f0484b: Pushed
5f70bf18a086: Mounted from wurstmeister/zookeeper
1e9319f86cdd: Pushed
fc2a5bdb17c: Pushed
b2d5eeeaba3a: Mounted from library/alpine
p1: digest: sha256:5ed95475167f7cf404102e4958c338c407efc5e5ed7512855a7c3de7e91ceec size: 1364

G:\Local disk\ Docker\python>
```



Deployment file

Each configuration file has **3 parts**

- 1) metadata
- 2) specification
- 3) status

Automatically generated and added by Kubernetes!



Deployment file

Each configuration file has 3 parts

Deployment

```
! nginx-deployment.yaml ✘  
  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: nginx-deployment  
5  +  labels:-  
6  +  spec:  
7    replicas: 2  
8    selector:-  
9    +  template:-  
10   +  22
```

1) metadata

2) specification

Service

```
! nginx-service.yaml ✘  
  
1  apiVersion: v1  
2  kind: Service  
3  metadata:  
4    name: nginx-service  
5  +  spec:  
6    selector:-  
7    +  ports:-  
8    +  12
```

Deployment file

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  >  labels:- 
6  spec:
7    replicas: 2
8    selector:- 
9  >  template:
10   metadata:
11     labels:
12       app: nginx
13
14   spec:
15     containers:
16       - name: nginx
17         image: nginx:1.16
18         ports:
19           - containerPort: 8080
```

Template

- has its own "metadata" and "spec" section
- applies to Pod
- blueprint for a Pod

port?

имя?

image?



Deployment file

Deployment

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10   matchLabels:
11     app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16   > spec:-
```

Labels & Selectors

Service

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5  spec:
6    selector:
7      app: nginx
8  > ports:-
```

174

Deployment file

Deployment

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4      name: nginx-deployment
5      labels:
6          app: nginx
7  spec:
8      replicas: 2
9      selector:
10         matchLabels:
11             app: nginx
12     template:
13         metadata:
14             labels:
15                 app: nginx
16 >     spec: -
```

Connecting Deployment to Pods

- Pods get the label through the template blueprint
- This label is matched by the selector

```
selector:
matchLabels:
    app: nginx
```



Deployment file

Connecting Services to Deployments

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10   matchLabels:
11     app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16   >     spec: -
```

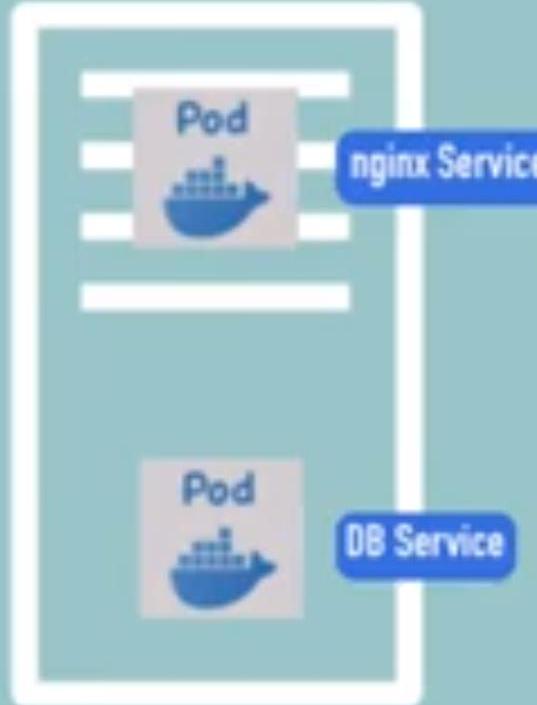
Service

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5  spec:
6    selector:
7      app: nginx
8    > ports:-
```

Deployment file

Suggested: Yaml Tutorial | Learn YAML in 18 mins [\(i\)](#)

Ports in Service and Pod



The diagram illustrates the relationship between a Pod and its associated services. A central Pod icon contains two containers: "nginx Service" and "DB Service". Arrows point from each container to its corresponding definition in a deployment file.

nginx Service

```
ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

DB Service

```
port: 80
```

Pod



Docker and Kubernetes - Python

```
G:\Local disk\ Docker\python>kubectl apply -f flask-deployment.yaml
service/flask-app-service created
deployment.apps/flask-deployment created

G:\Local disk\ Docker\python>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
flask-deployment-74c6747b67-b6882  1/1     Running    0          7s
mysql-545ccdb445-6jnt8            0/1     CrashLoopBackOff 29         17h
mysql-545ccdb445-k4glg            0/1     CrashLoopBackOff 26         17h
mysql-545ccdb445-v5v2g            1/1     Running    30         17h

G:\Local disk\ Docker\python>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
flask-deployment-74c6747b67-b6882  1/1     Running    0          12s
mysql-545ccdb445-6jnt8            0/1     CrashLoopBackOff 29         17h
mysql-545ccdb445-k4glg            0/1     CrashLoopBackOff 26         17h
mysql-545ccdb445-v5v2g            1/1     Running    30         17h

G:\Local disk\ Docker\python>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
flask-deployment-74c6747b67-b6882  1/1     Running    0          13s
mysql-545ccdb445-6jnt8            0/1     CrashLoopBackOff 29         17h
mysql-545ccdb445-k4glg            0/1     CrashLoopBackOff 26         17h
mysql-545ccdb445-v5v2g            1/1     Running    30         17h

G:\Local disk\ Docker\python>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
flask-deployment-74c6747b67-b6882  1/1     Running    0          14s
mysql-545ccdb445-6jnt8            0/1     CrashLoopBackOff 29         17h
mysql-545ccdb445-k4glg            0/1     CrashLoopBackOff 26         17h
mysql-545ccdb445-v5v2g            1/1     Running    30         17h

G:\Local disk\ Docker\python>kubectl get nodes -o wide
```





Docker and Kubernetes - Python

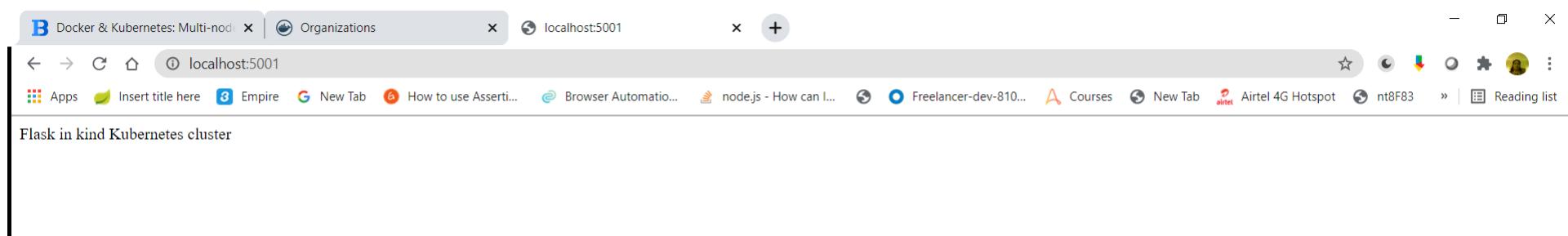
```
c:\ Administrator: Command Prompt - kubectl port-forward flask-deployment-74c6747b67-b6882 5001:8087

G:\Local disk\ Docker\python>kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
flask-app-service  NodePort  10.104.20.51 <none>        80:30087/TCP  5m11s
kubernetes      ClusterIP 10.96.0.1    <none>        443/TCP      2d2h
mysql            ClusterIP  None         <none>        3306/TCP     17h

G:\Local disk\ Docker\python>kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
flask-deployment-74c6747b67-b6882  1/1     Running   0          5m21s
mysql-545ccdb445-6jnt8          0/1     Error     30         17h
mysql-545ccdb445-k4g1g          0/1     CrashLoopBackOff 27         17h
mysql-545ccdb445-v5v2g          1/1     Running   30         17h

G:\Local disk\ Docker\python>kubectl logs flask-deployment-74c6747b67-b6882
 * Running on http://0.0.0.0:8087/ (Press CTRL+C to quit)

G:\Local disk\ Docker\python>kubectl port-forward flask-deployment-74c6747b67-b6882 5001:8087
Forwarding from 127.0.0.1:5001 -> 8087
Forwarding from [::1]:5001 -> 8087
Handling connection for 5001
Handling connection for 5001
```





Expose Service using External IP

- `kubectl expose deployment cgi-flask-deployment --type=LoadBalancer --name=flask-app-service-v1`
- Access service using service without port forwarding
- `http://localhost:8087`



Expose Service using External IP

```
C:\WINDOWS\system32>kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
cgi-flask-deployment-797b87f5cc-dp9qp   1/1     Running   1          3h23m

C:\WINDOWS\system32>kubectl get svc
NAME            TYPE      CLUSTER-IP        EXTERNAL-IP      PORT(S)        AGE
flask-app-service   NodePort   10.108.117.215   <none>           80:30087/TCP   3h23m
kubernetes       ClusterIP  10.96.0.1        <none>           443/TCP        2d12h

C:\WINDOWS\system32>kubectl get deployments
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
cgi-flask-deployment   1/1     1           1          3h23m

C:\WINDOWS\system32>kubectl expose deployment cgi-flask-deployment --type=LoadBalancer --name=flask-app-service
Error from server (AlreadyExists): services "flask-app-service" already exists

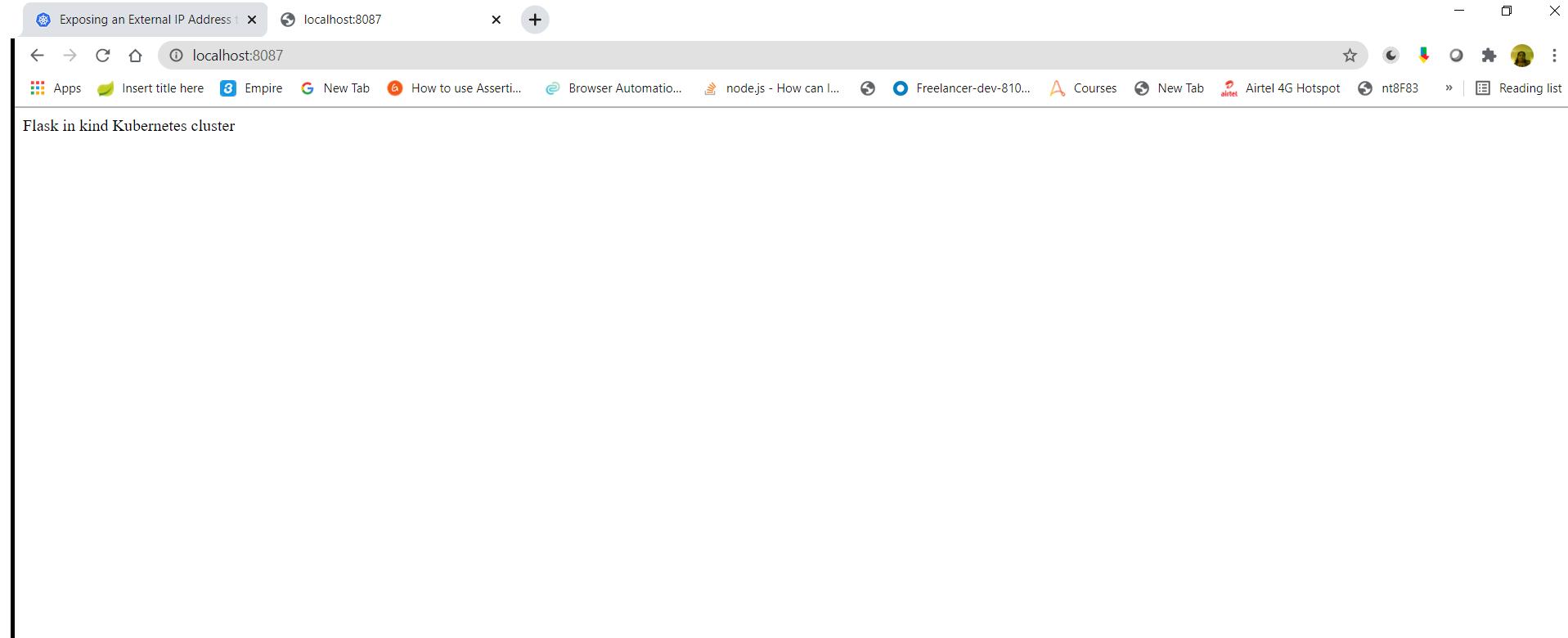
C:\WINDOWS\system32>kubectl expose deployment cgi-flask-deployment --type=LoadBalancer --name=flask-app-service-v1
service/flask-app-service-v1 exposed

C:\WINDOWS\system32>kubectl get svc
NAME            TYPE      CLUSTER-IP        EXTERNAL-IP      PORT(S)        AGE
flask-app-service   NodePort   10.108.117.215   <none>           80:30087/TCP   3h27m
flask-app-service-v1  LoadBalancer  10.107.199.122  localhost       8087:31205/TCP   4s
kubernetes       ClusterIP  10.96.0.1        <none>           443/TCP        2d12h

C:\WINDOWS\system32>
```



Expose Service using External IP





What is an Ingress?

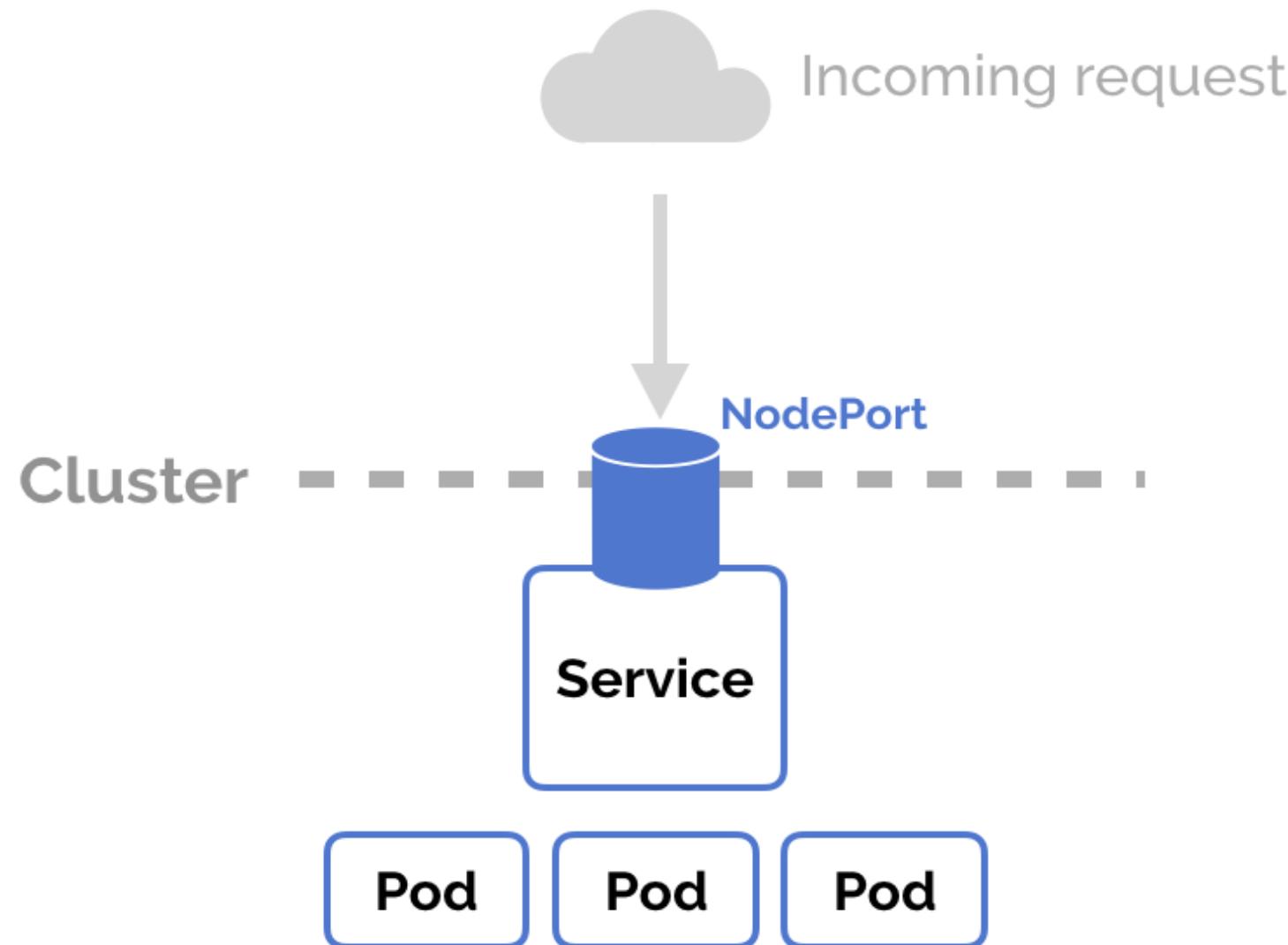
- In Kubernetes, an Ingress is an object that allows access to Kubernetes services from outside the Kubernetes cluster.
- We configure access by creating a collection of rules that define which inbound connections reach which services.



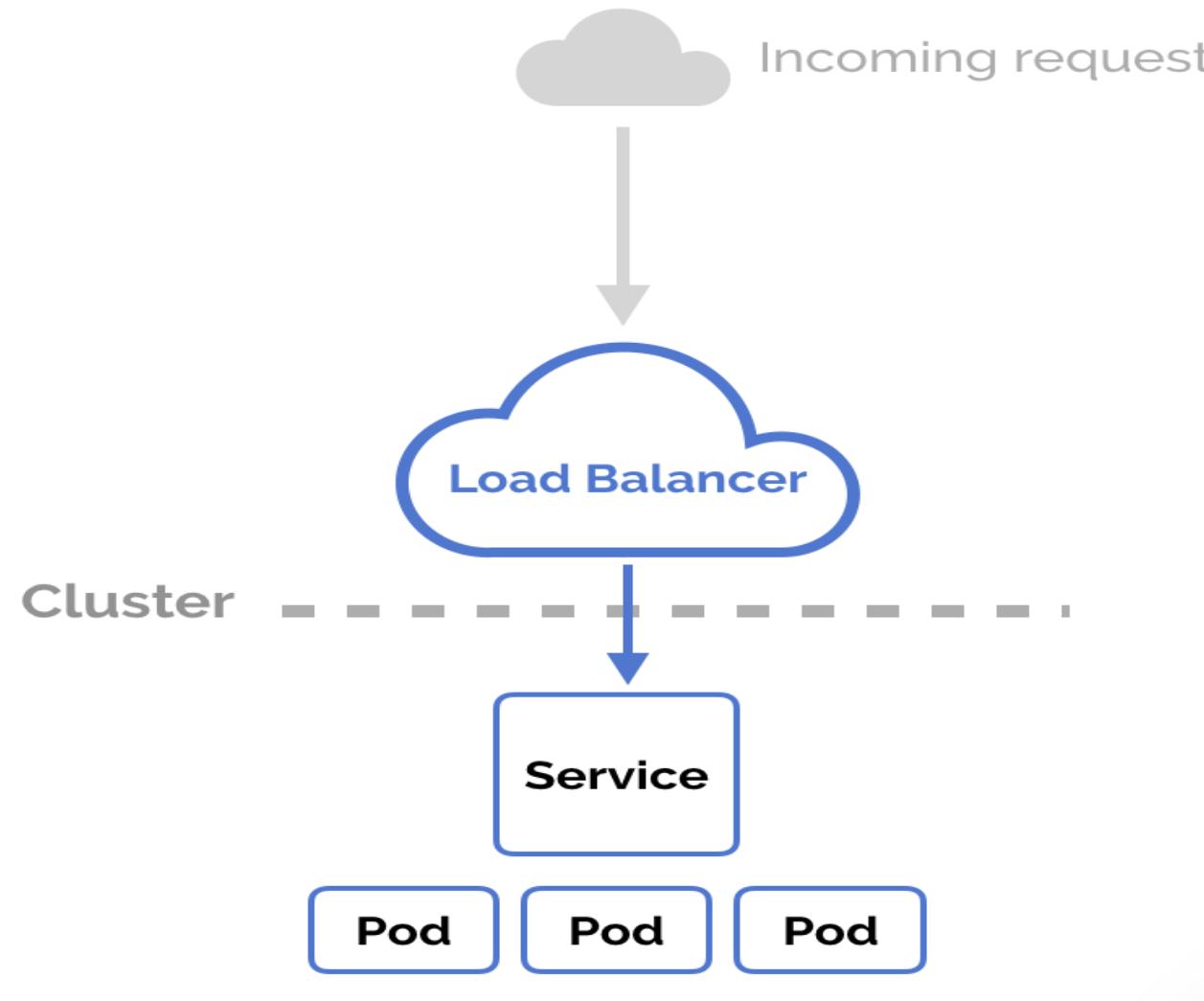
What is an Ingress?

- For example, you might want to send requests to `example.com/api/v1/` to an `api-v1` service, and requests to `example.com/api/v2/` to the `api-v2` service.
- With an Ingress, we can easily set this up without creating a bunch of LoadBalancers or exposing each service on the Node.

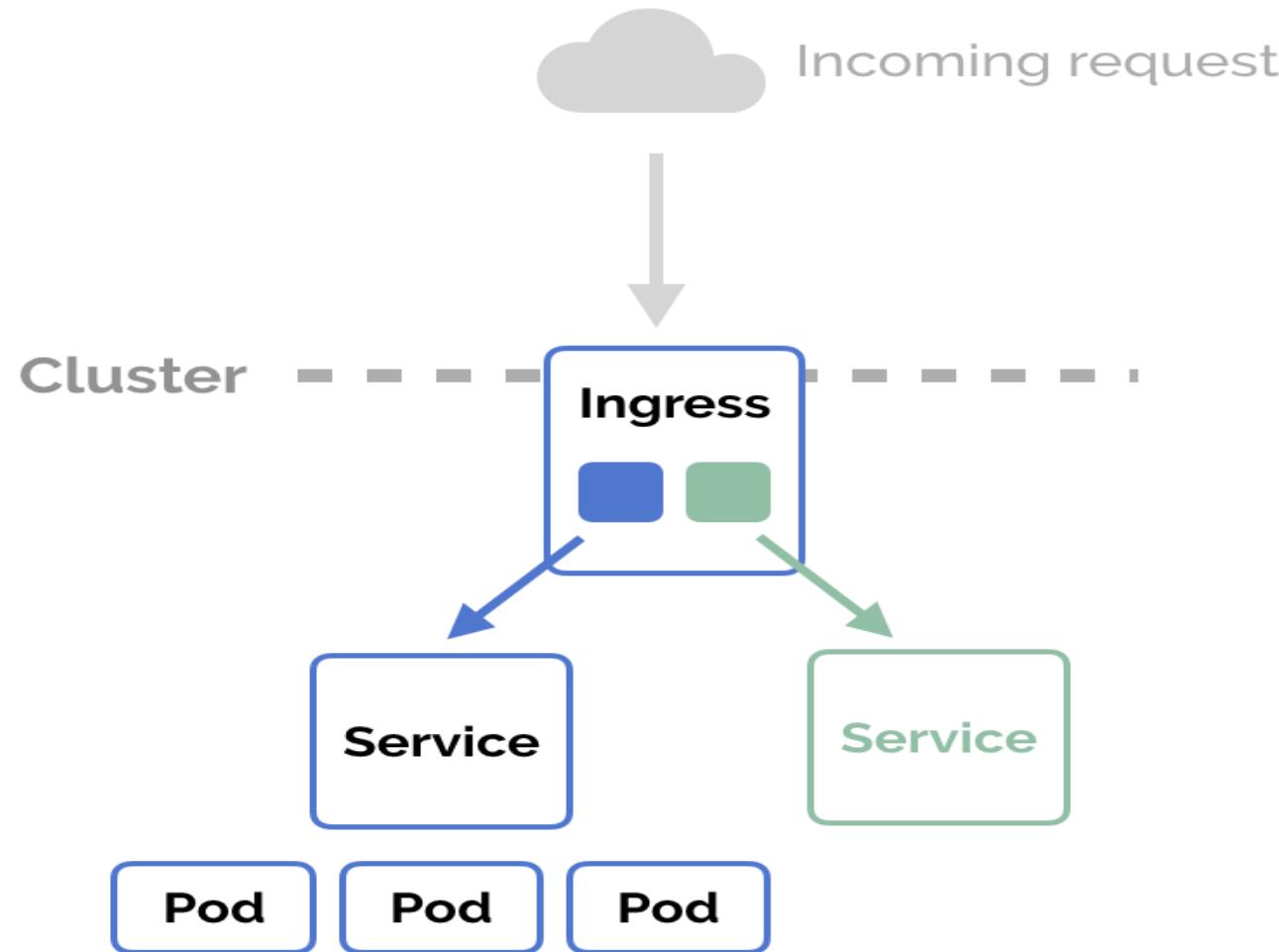
Kubernetes Ingress vs LoadBalancer vs NodePort



Kubernetes Ingress vs LoadBalancer vs NodePort⁺



Kubernetes Ingress vs LoadBalancer vs NodePort





Loadbalance and Routing - Ingress

- ingress installation
- kubectl apply -f
<https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.41.2/deploy/static/provider/cloud/deploy.yaml>
- verify installation
- kubectl get pods -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx --watch



Loadbalance and Routing - Ingress

- kubectl apply -f apple.yaml
- kubectl apply -f banana.yaml
- kubectl apply -f ingress.yaml
- curl -kL <http://localhost/apple>
- curl -kL <http://localhost/banana>



What is helm

- In simple terms, Helm is a package manager for Kubernetes.
- Helm is the K8s equivalent of yum or apt.
- Helm deploys charts, which you can think of as a packaged application.
- It is a collection of all your versioned, pre-configured application resources which can be deployed as one unit.
- You can then deploy another version of the chart with a different set of configuration.



What is helm

- Helm helps in three key ways:
- Improves productivity
- Reduces the complexity of deployments of microservices
- Enables the adaptation of cloud native applications



What are Helm charts?

- Helm Charts are simply Kubernetes YAML manifests combined into a single package that can be advertised to your Kubernetes clusters.
- Once packaged, installing a Helm Chart into your cluster is as easy as running a single `helm install`, which really simplifies the deployment of containerized applications.



Helm Install

- choco install kubernetes-helm
- Or
- <https://github.com/helm/helm/releases>
- Unzip
- Initialize helm using the **helm init** command.
- Finally, add the repo of the helm chart using the below-given command.
- helm repo add stable <https://kubernetes-charts.storage.googleapis.com/>
- helm repo list



Deploy an Apache webserver using Helm

- Location to search
- helm repo add bitnami <https://charts.bitnami.com/bitnami>
- Download apache
- helm install my-apache bitnami/apache --version 8.0.2



Deploy an Apache webserver using Helm

Microservices

```
Administrator: Command Prompt
my-apache-5fc88f7566-d9hx2      0/1  ContainerCreating  0    42s
nginx-ingress-ingress-controller-7cc994599f-q2q8v  1/1  Running       1    9h
pod-env-var                      1/1  Running       0   12m

G:\Local disk\ Docker\ configmaps> kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
cgi-flask-deployment-797b87f5cc-dp9qp     1/1    Running   2          14h
configmap-pod-1                     0/1    Completed  0          49m
hello-kubernetes-first-6bcf4dd4f5-187qg   1/1    Running   1          8h
hello-kubernetes-first-6bcf4dd4f5-lprlp    1/1    Running   1          8h
hello-kubernetes-first-6bcf4dd4f5-rs6gt    1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-4475m   1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-h2qp6    1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-zwxnj   1/1    Running   1          8h
my-apache-5fc88f7566-d9hx2      0/1    Running   0          43s
nginx-ingress-ingress-controller-7cc994599f-q2q8v  1/1    Running   1          9h
pod-env-var                      1/1    Running   0          12m

G:\Local disk\ Docker\ configmaps> kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
cgi-flask-deployment-797b87f5cc-dp9qp     1/1    Running   2          14h
configmap-pod-1                     0/1    Completed  0          49m
hello-kubernetes-first-6bcf4dd4f5-187qg   1/1    Running   1          8h
hello-kubernetes-first-6bcf4dd4f5-lprlp    1/1    Running   1          8h
hello-kubernetes-first-6bcf4dd4f5-rs6gt    1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-4475m   1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-h2qp6    1/1    Running   1          8h
hello-kubernetes-second-5b4f7977f9-zwxnj   1/1    Running   1          8h
my-apache-5fc88f7566-d9hx2      0/1    Running   0          45s
nginx-ingress-ingress-controller-7cc994599f-q2q8v  1/1    Running   1          9h
pod-env-var                      1/1    Running   0          12m

G:\Local disk\ Docker\ configmaps>
```





Deploy an Apache webserver using Helm

- helm list
- helm upgrade my-apache bitnami/apache --version 8.0.3
- Rollback
- helm rollback my-apache 1
- helm delete my-apache



How to Access Production-Ready Helm Charts

- We can search public hubs for Charts that enable you to quickly deploy your desired application with a customizable configuration.
- A Helm Chart doesn't just contain a static set of definitions.
- Helm comes with capabilities to hook into any lifecycle state of a Kubernetes deployment.
- This means during the installation or upgrade of an application, various actions can be executed like creating a database update before updating the actual database.



How to Access Production-Ready Helm Charts

- Helm might be heavy for a simple container like a single node web server, but it's very useful for more complex applications.
- For example it works great for a distributed system like Kafka or Cassandra that usually runs on multiple distributed nodes on different datacenters.



Production-ready WordPress application

- Containers that serve WordPress,
- Instances of MariaDB for persistence and
- Prometheus sidecar containers for each WordPress container to expose health metrics.



How to Deploy WordPress and MariaDB

- helm repo add bitnami <https://charts.bitnami.com/bitnami>
- helm install my-wordpress bitnami/wordpress --version 10.1.4



Loadbalance and Routing - Ingress

- 1. create first-app.yaml
- 2. create second-app.yaml
- 3. deploy them
- 4. helm repo add ingress-nginx <https://kubernetes.github.io/ingress-nginx>
- 5.helm repo update
- 6.helm install nginx-ingress ingress-nginx/ingress-nginx --set controller.publishService.enabled=true
- 7.watch the load balancer by running
- kubectl --namespace default get services -o wide -w nginx-ingress-ingress-ingress-controller



Loadbalance and Routing - Ingress

- Kubectl apply –f app-ingress.yaml
- kubectl get ingress hello-kubernetes-ingress
- kubectl describe ingress hello-kubernetes-ingress



What is a ConfigMap in Kubernetes?

- A ConfigMap is a dictionary of configuration settings.
- This dictionary consists of key-value pairs of strings.
- Kubernetes provides these values to your containers.
- Like with other dictionaries (maps, hashes, ...) the key lets you get and set the configuration value.

Why would you use a ConfigMap in Kubernetes?



- Use a ConfigMap to keep your application code separate from your configuration.
- It is an important part of creating a Twelve-Factor Application.
- A ConfigMap stores configuration settings for your code. Store connection strings, public credentials, hostnames, and URLs in your ConfigMap.



How does a ConfigMap work?





ConfigMaps

- ConfigMaps can be used to:
 - Populate the values of environment variables
 - Set command-line arguments in a container
 - Populate config files in a volume



Creating ConfigMaps from file

- `kubectl create configmap` command to create configmaps easily from literal values, files, or directories.
- `kubectl create configmap configmap-example-1 --from-file=./config-map-data.txt`
- `kubectl describe configmap/configmap-example-1`
- `kubectl delete configmap/configmap-example-1`



Create ConfigMaps from Literal Values

- `kubectl create configmap literal-config-1 --from-literal=literal-key1=value1 --from-literal=literal-key2=22`
- `kubectl create configmap literal-config-mysql --from-literal=literal-port=3306 --from-literal=literal-host=127.0.0.1 --from-literal=literal-user=root --from-literal=literal-password=password`



Create ConfigMaps from-literal

- `kubectl create configmap literal-config-1 --from-literal=city=London --from-literal=population=millions`



Create ConfigMaps from Directories

- `kubectl create configmap from-folder-configmap --from-file=test`
- `kubectl describe configmap/from-folder-configmap`

```
error: error reading test: The system cannot find the file specified.  
G:\Local disk\ Docker\configmaps\test>cd..  
  
G:\Local disk\ Docker\configmaps>kubectl create configmap from-folder-configmap --from-file=test  
configmap/from-folder-configmap created  
  
G:\Local disk\ Docker\configmaps>
```



Use Configmap as Env variable

- `kubectl get configmaps literal-config-1 -o yaml`
- `kubectl create -f Config-Map-demo1.yaml`

Use Configmap as Env variable



```
Administrator: Command Prompt
Ready: False
Restart Count: 0
Environment:
  ENV_Literal: <set to the key 'literal-key1' of config map 'literal-config-1'> Optional: false
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-zxhp (ro)
Conditions:
  Type          Status
  Initialized   True
  Ready         False
  ContainersReady False
  PodScheduled  True
Volumes:
  default-token-zxhp:
    Type:      Secret (a volume populated by a Secret)
    SecretName: default-token-zxhp
    Optional:   false
    QoS Class: BestEffort
    Node-Selectors: <none>
    Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s
                  node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type  Reason  Age   From           Message
  ----  -----  ---   ----
  Normal Scheduled <unknown>   kubelet, docker-desktop  Successfully assigned default/configmap-pod-1 to docker-desktop
  Normal Pulling   19m    kubelet, docker-desktop  Pulling image "alpine"
  Normal Pulled    19m    kubelet, docker-desktop  Successfully pulled image "alpine" in 9.9872422s
  Normal Created   19m    kubelet, docker-desktop  Created container cm-container-1
  Normal Started   19m    kubelet, docker-desktop  Started container cm-container-1

G:\Local disk\Dockers\configmaps>
```

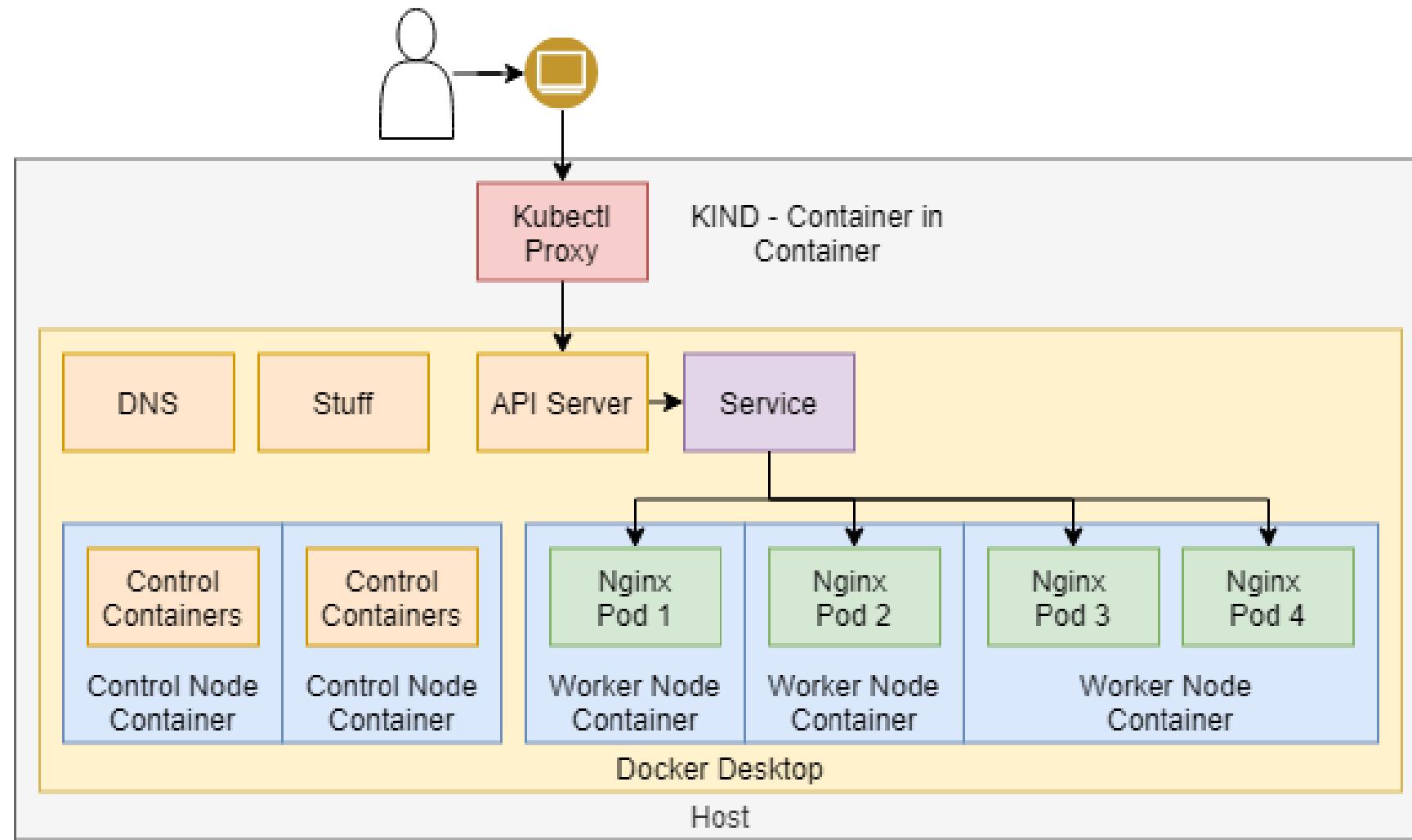
The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The output displays the status of a Kubernetes pod named "default/configmap-pod-1". Key details include:

- Ready:** False
- Mounts:** A volume named "default-token-zxhp" is mounted at "/var/run/secrets/kubernetes.io/serviceaccount" with read-only (ro) access.
- Conditions:** The pod is not ready because its containers are not ready.
- Volumes:** A secret volume named "default-token-zxhp" is defined, which contains a "Secret" type volume populated by a secret named "default-token-zxhp". It is optional and has a QoS class of "BestEffort".
- Events:** The pod has been scheduled to a node, and its containers have been successfully created and started.

The command prompt's path is "G:\Local disk\Dockers\configmaps>". The taskbar at the bottom shows various application icons, and the system tray indicates the date and time as "22/04/2021 08:37".



Docker-Desktop wsl2 Multinode cluster





Docker-Desktop wsl2 Multinode cluster

- curl.exe -Lo kind-windows-amd64.exe
<https://kind.sigs.k8s.io/dl/v0.10.0/kind-windows-amd64>
- copy .\kind-windows-amd64.exe d:\kube\kind.exe
- Go to d:\kube
- kind create cluster # Default cluster context name is `kind`.
- kind create cluster --name kind-2

Docker-Desktop wsl2 Multinode cluster

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>curl.exe -Lo kind-windows-amd64.exe https://kind.sigs.k8s.io/dl/v0.10.0/kind-windows-amd64
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload   Total Spent   Left  Speed
100  100  100  100    0     0      33      0  0:00:03  0:00:03  ---:--   32
100  626    0  626    0     0     156      0  --:---:--  0:00:04  --:---:-- 39125
100 7214k  100 7214k    0     0   1442k      0  0:00:05  0:00:05  --:---:-- 6449k

C:\WINDOWS\system32>Move-Item .\kind-windows-amd64.exe c:\some-dir-in-your-PATH\kind.exe
'Move-Item' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>curl.exe -Lo kind-windows-amd64.exe https://kind.sigs.k8s.io/dl/v0.10.0/kind-windows-amd64
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload   Total Spent   Left  Speed
100  100  100  100    0     0     100      0  0:00:01  ---:--  0:00:01  182
100  626  100  626    0     0     626      0  0:00:01  0:00:01  ---:--   0
100 7214k  100 7214k    0     0   2404k      0  0:00:03  0:00:03  --:---:-- 5479k

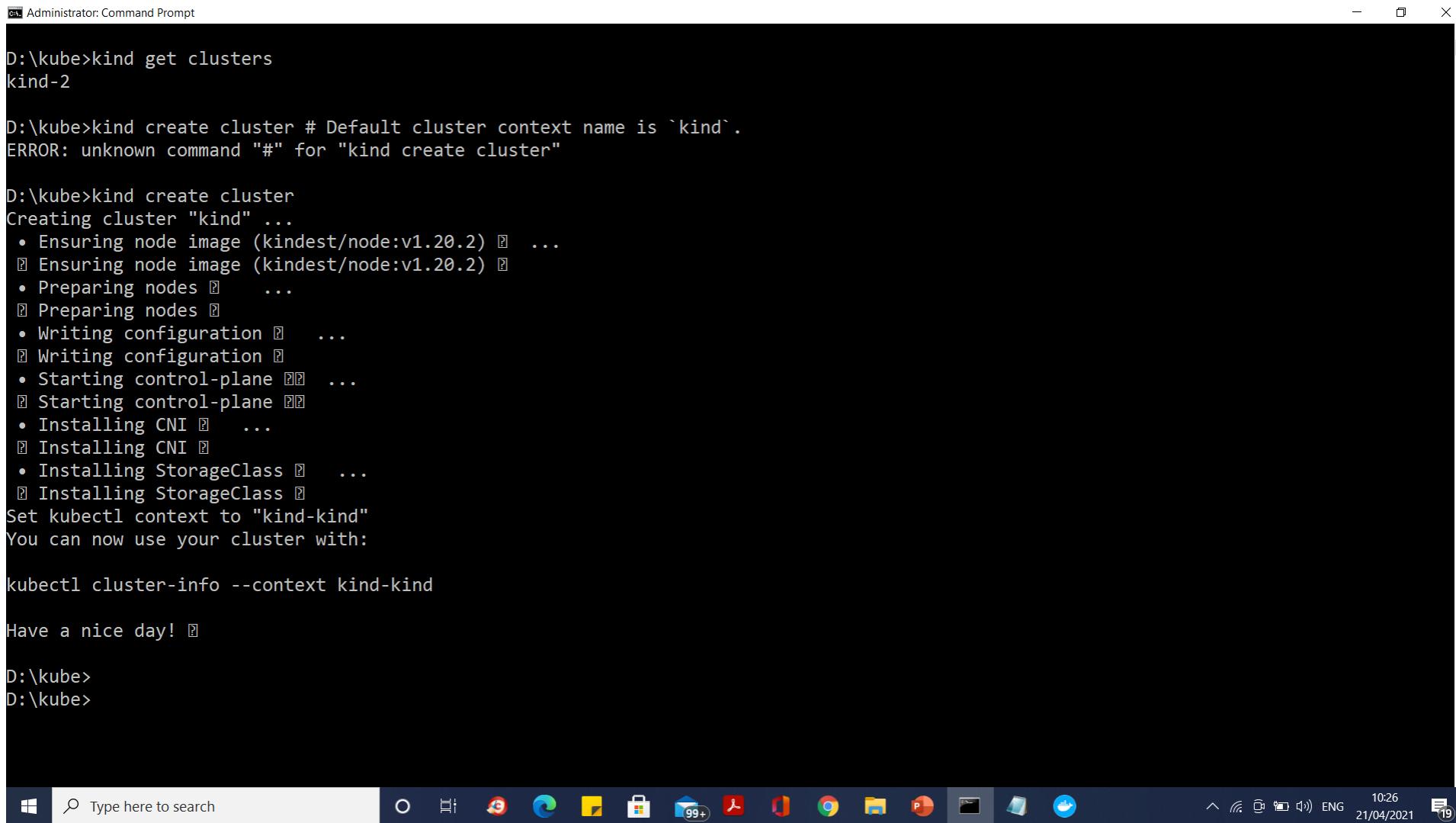
C:\WINDOWS\system32>Move-Item .\kind-windows-amd64.exe d:\kube\kind.exe
'Move-Item' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>copy .\kind-windows-amd64.exe d:\kube\kind.exe
      1 file(s) copied.

C:\WINDOWS\system32>choco install kind
'choco' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>kind create cluster --name kind-2
```

Docker-Desktop wsl2 Multinode cluster



A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the following terminal session:

```
D:\kube>kind get clusters
kind-2

D:\kube>kind create cluster # Default cluster context name is `kind` .
ERROR: unknown command "#" for "kind create cluster"

D:\kube>kind create cluster
Creating cluster "kind" ...
• Ensuring node image (kindest/node:v1.20.2) ...
  Ensuring node image (kindest/node:v1.20.2)
• Preparing nodes ...
  Preparing nodes ...
• Writing configuration ...
  Writing configuration ...
• Starting control-plane ...
  Starting control-plane ...
• Installing CNI ...
  Installing CNI ...
• Installing StorageClass ...
  Installing StorageClass ...
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a nice day! ...

D:\kube>
D:\kube>
```

The taskbar at the bottom of the screen shows various pinned icons, including File Explorer, Edge, and Microsoft Office applications. The system tray indicates the date as 21/04/2021 and the time as 10:26.

Docker-Desktop wsl2 Multinode cluster

```
C:\ Administrator: Command Prompt
C:\WINDOWS\system32>kind create cluster --name kind-2
'kind' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>d:

D:\>cd kube

D:\kube>kind create cluster --name kind-2
Creating cluster "kind-2" ...
  • Ensuring node image (kindest/node:v1.20.2) ...
  □ Ensuring node image (kindest/node:v1.20.2) ...
  • Preparing nodes ...
  □ Preparing nodes ...
  • Writing configuration ...
  □ Writing configuration ...
  • Starting control-plane ...
  □ Starting control-plane ...
  • Installing CNI ...
  □ Installing CNI ...
  • Installing StorageClass ...
  □ Installing StorageClass ...
Set kubectl context to "kind-kind-2"
You can now use your cluster with:

kubectl cluster-info --context kind-kind-2

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#community
D:\kube>
D:\kube>
```

Docker-Desktop wsl2 Multinode cluster

```
Administrator: Command Prompt
D:\kube>kind get clusters
kind
kind-2

D:\kube>kubectl cluster-info --context kind-kind
Kubernetes master is running at https://127.0.0.1:56075
KubeDNS is running at https://127.0.0.1:56075/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

D:\kube>kubectl cluster-info --context kind-kind-2
Kubernetes master is running at https://127.0.0.1:54442
KubeDNS is running at https://127.0.0.1:54442/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

D:\kube>kubectl cluster-info
Kubernetes master is running at https://127.0.0.1:56075
KubeDNS is running at https://127.0.0.1:56075/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

D:\kube>
```



Kubernetes multiple control planes

- # a cluster with 3 control-plane nodes and 3 workers
- kind: Cluster
- apiVersion: kind.x-k8s.io/v1alpha4
- nodes:
 - - role: control-plane
 - - role: control-plane
 - - role: control-plane
 - - role: worker
 - - role: worker
 - - role: worker



Kubernetes multiple control planes

- kind: Cluster
- apiVersion: kind.x-k8s.io/v1alpha4
- nodes:
 - - role: control-plane
 - extraPortMappings:
 - - containerPort: 80
 - hostPort: 80
 - listenAddress: "0.0.0.0" # Optional, defaults to "0.0.0.0"
 - protocol: udp # Optional, defaults to tcp



Kubernetes multiple control planes

```
C:\ Administrator: Command Prompt
Deleting cluster "kind" ...

D:\kube>kind get clusters
kind-2

D:\kube>kind create cluster --config "G:\Local disk\ Docker\ kind-example-config.yaml"
Creating cluster "kind" ...
  • Ensuring node image (kindest/node:v1.20.2)  ...
  ② Ensuring node image (kindest/node:v1.20.2)  ...
  • Preparing nodes ② ②  ...
  ② Preparing nodes ② ②  ...
  • Writing configuration ②  ...
  ② Writing configuration ②
  • Starting control-plane ②  ...
  ② Starting control-plane ②
  • Installing CNI ②  ...
  ② Installing CNI ②
  • Installing StorageClass ②  ...
  ② Installing StorageClass ②
  • Joining worker nodes ②  ...
  ② Joining worker nodes ②
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next? ② Check out https://kind.sigs.k8s.io/docs/user/quick-start/

D:\kube>kind get clusters
kind
kind-2

D:\kube>
```



Kubernetes multiple control planes

```
C:\ Administrator: Command Prompt
D:\kube>kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
kind-control-plane   Ready    control-plane,master   6m46s  v1.20.2
kind-worker       Ready    <none>     6m10s  v1.20.2
kind-worker2      Ready    <none>     6m10s  v1.20.2

D:\kube>kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
docker-desktop   Ready    master    2d1h  v1.19.3

D:\kube>kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
kind-2-control-plane   Ready    control-plane,master   42m   v1.20.2

D:\kube>
```



Multi node cluster dashboard

- Repeat dashboard installation steps for every cluster



Kubernetes Deployment Policies

- recreate: terminate the old version and release the new one
- ramped: release a new version on a rolling update fashion, one after the other
- blue/green: release a new version alongside the old version then switch traffic
- canary: release a new version to a subset of users, then proceed to a full rollout
- a/b testing: release a new version to a subset of users in a precise way (HTTP headers, cookie, weight, etc.).
- A/B testing is really a technique for making business decisions based on statistics but we will briefly describe the process.
- This doesn't come out of the box with Kubernetes, it implies extra work to setup a more advanced infrastructure (Istio, Linkerd, Traefik, custom nginx/haproxy, etc).



Recreate - best for development environment

- spec:
- replicas: 3
- strategy:
- type: Recreate
-

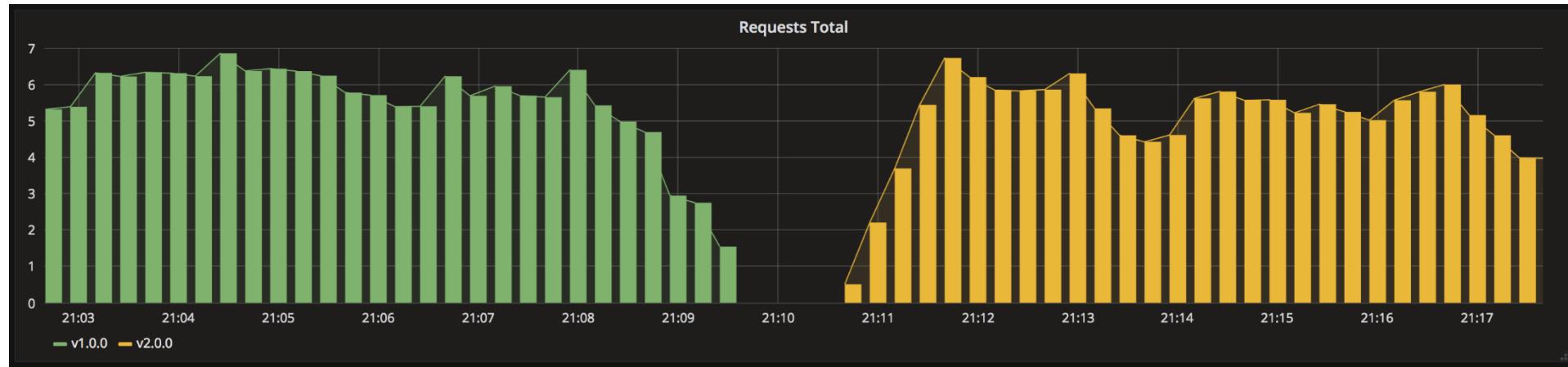


Recreate - best for development environment

- The recreate strategy is a dummy deployment which consists of shutting down version A then deploying version B after version A is turned off.
- This technique implies downtime of the service that depends on both shutdown and boot duration of the application.



Recreate - best for development environment





Recreate - best for development environment

- Deploy the first application
- \$ kubectl apply -f app-v1.yaml
- # Test if the deployment was successful
- Kubectl logs deploy/my-app
- # Then deploy version 2 of the application
- \$ kubectl apply -f app-v2.yaml
- Kubectl logs deploy/my-app

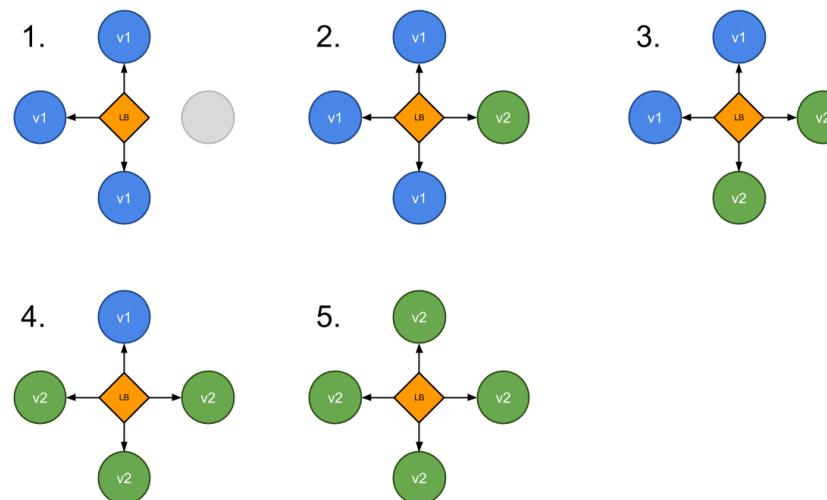


Recreate - best for development environment

- Pro:
 - application state entirely renewed
- Cons:
 - downtime that depends on both shutdown and boot duration of the application

Ramped - slow rollout

- A ramped deployment updates pods in a rolling update fashion, a secondary ReplicaSet is created with the new version of the application, then the number of replicas of the old version is decreased and the new version is increased until the correct number of replicas is reached.





Ramped – Slow Rollout

- spec:
- replicas: 3
- strategy:
- type: RollingUpdate
- rollingUpdate:
 - maxSurge: 2 # how many pods we can add at a time
 - maxUnavailable: 0 # maxUnavailable define how many pods can be unavailable
 - # during the rolling update
-



Ramped- best for development environment

- Deploy the first application
- \$ kubectl apply -f app-v1.yaml
- # Test if the deployment was successful
- Kubectl logs deploy/my-app
- # Then deploy version 2 of the application
- \$ kubectl apply -f app-v2.yaml
- Kubectl logs deploy/my-app



Ramped- best for development environment

- # In case you discover some issue with the new version, you can undo the rollout
 - \$ kubectl rollout undo deploy my-app
- # If you can also pause the rollout if you want to run the application for a subset of users
 - \$ kubectl rollout pause deploy my-app
- # Then if you are satisfy with the result, rollout
 - \$ kubectl rollout resume deploy my-app



Ramped- best for development environment

- When setup together with horizontal pod autoscaling it can be handy to use a percentage based value instead of a number for maxSurge and maxUnavailable.
- If you trigger a deployment while an existing rollout is in progress, the deployment will pause the rollout and proceed to a new release by overriding the rollout.



Ramped- best for development environment

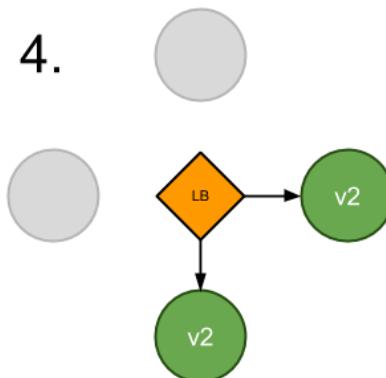
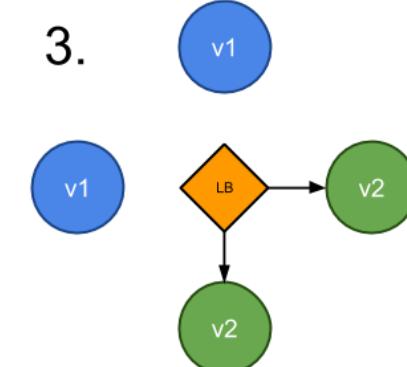
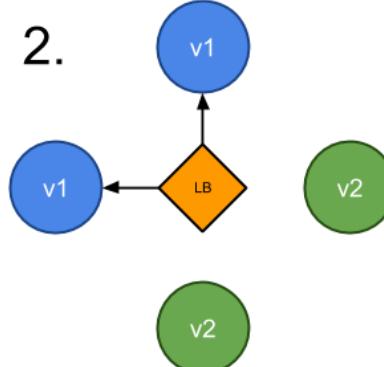
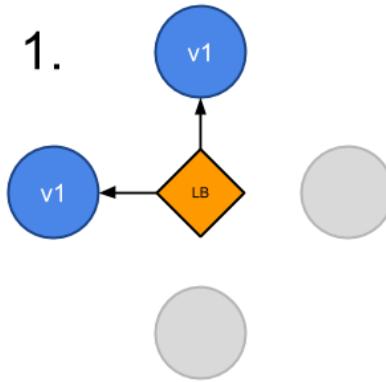
- Pro:
 - version is slowly released across instances
 - convenient for stateful applications that can handle rebalancing of the data
- Cons:
 - rollout/rollback can take time
 - supporting multiple APIs is hard
 - no control over traffic

Blue/Green - best to avoid API versioning issues



- A blue/green deployment differs from a ramped deployment because the "green" version of the application is deployed alongside the "blue" version.
- After testing that the new version meets the requirements, we update the Kubernetes Service object that plays the role of load balancer to send traffic to the new version by replacing the version label in the selector field.

Blue/Green - best to avoid API versioning issues





ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.
- As such, it is often used to guarantee the availability of a specified number of identical Pods.



ReplicaSet

- A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.
- A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number.
- When a ReplicaSet needs to create new Pods, it uses its Pod template.



ReplicaSet

- When to use a ReplicaSet
- A ReplicaSet ensures that a specified number of pod replicas are running at any given time.
- However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.
- Therefore, we recommend using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.

controllers/frontend.yaml

```
apiVersion: apps/v1
kind: Replicaset
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
```

kubectl get rs



What is a Deployment?

- A Deployment provides declarative updates for Pods and Replica Sets (the next-generation Replication Controller).
- You only need to describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you.
- You can define Deployments to create new resources, or replace existing ones by new ones.



What is a Deployment?

- A typical use case is:
- Create a Deployment to bring up a Replica Set and Pods.
- Check the status of a Deployment to see if it succeeds or not.
- Later, update that Deployment to recreate the Pods (for example, to use a new image).
- Rollback to an earlier Deployment revision if the current Deployment isn't stable.
- Pause and resume a Deployment.



Creating a Deployment

- Here is an example Deployment. It creates a Replica Set to bring up 3 nginx Pods.
- {% include code.html language="yaml" file="nginx-deployment.yaml" ghlink="/docs/concepts/workloads/controllers/nginx-deployment.yaml" %}
- Run the example by downloading the example file and then running this command:
- \$ kubectl create -f docs/user-guide/nginx-deployment.yaml --record
- deployment "nginx-deployment" created



Creating a Deployment

- \$ kubectl get deployments
- | NAME | DESIRED | CURRENT | UP-TO-DATE | AVAILABLE | AGE |
|------------------|---------|---------|------------|-----------|-----|
| nginx-deployment | 3 | 0 | 0 | 0 | 1s |
- \$ kubectl get rs
- | NAME | DESIRED | CURRENT | READY | AGE |
|-----------------------------|---------|---------|-------|-----|
| nginx-deployment-2035384211 | 3 | 3 | 0 | 18s |



Creating a Deployment

- \$ kubectl get pods --show-labels



Rolling Back a Deployment

- Sometimes you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping.
- By default, two previous Deployment's rollout history are kept in the system so that you can rollback anytime you want.



Rolling Back a Deployment

- Note: a Deployment's revision is created when a Deployment's rollout is triggered.
- This means that the new revision is created if and only if the Deployment's pod template (i.e. `.spec.template`) is changed, e.g. updating labels or container images of the template.
- Other updates, such as scaling the Deployment, will not create a Deployment revision -- so that we can facilitate simultaneous manual- or auto-scaling.
- This implies that when you rollback to an earlier revision, only the Deployment's pod template part will be rolled back.



Rolling Back a Deployment

- `kubectl set image deployment/nginx-deployment nginx=nginx:1.91`
- `kubectl rollout status deployments nginx-deployment`
- `kubectl rollout history deployment/nginx-deployment`
- `kubectl rollout history deployment/nginx-deployment --revision=2`



Scaling a Deployment

- `kubectl scale deployment nginx-deployment --replicas 10`
- `kubectl get deploy`
- `kubectl set image deploy/nginx-deployment nginx=nginx:sometag`



Working with Kubernetes Objects

- Kubernetes objects are persistent entities in the Kubernetes system.
- Kubernetes uses these entities to represent the state of your cluster.
- Specifically, they can describe:
- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance



Init containers

- A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.
- Init containers are exactly like regular containers, except:
- Init containers always run to completion.
- Each init container must complete successfully before the next one starts.
- If a Pod's init container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds.
- However, if the Pod has a restartPolicy of Never, Kubernetes does not restart the Pod.



Differences from regular containers

- Init containers support all the fields and features of app containers, including resource limits, volumes, and security settings.
- However, the resource requests and limits for an init container are handled differently, as documented in Resources.
- Also, init containers do not support lifecycle, livenessProbe, readinessProbe, or startupProbe because they must run to completion before the Pod can be ready.



Create a Pod that has an Init Container

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html
      # These containers are run during pod initialization
  initContainers:
    - name: install
      image: busybox
      command:
        - wget
        - "-O"
        - "/work-dir/index.html"
        - http://kubernetes.io
      volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
        - name: workdir
          emptyDir: {}
```



Working with Kubernetes Objects

- To work with Kubernetes objects--whether to create, modify, or delete them--you'll need to use the Kubernetes API.
- When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.



Object Spec and Status

- Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the object spec and the object status.
- For objects that have a spec, you have to set this when you create the object, providing a description of the characteristics you want the resource to have: its desired state.
- The status describes the current state of the object, supplied and updated by the Kubernetes system and its components.
- The Kubernetes control plane continually and actively manages every object's actual state to match the desired state you supplied.



Object Spec and Status

- For example: in Kubernetes, a Deployment is an object that can represent an application running on your cluster.
- When you create the Deployment, you might set the Deployment spec to specify that you want three replicas of the application to be running.
- The Kubernetes system reads the Deployment spec and starts three instances of your desired application--updating the status to match your spec.
- If any of those instances should fail (a status change), the Kubernetes system responds to the difference between spec and status by making a correction--in this case, starting a replacement instance.



Describing a Kubernetes object

- Deployment.yaml
- apiVersion - Which version of the Kubernetes API you're using to create this object
- kind - What kind of object you want to create
- metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
- spec - What state you desire for the object



Network Terms

- Layer 2 Networking
- Layer 2 is the data link layer providing Node-to-Node data transfer.
- It defines the protocol to establish and terminate a connection between two physically connected devices. It also defines the protocol for flow control between them.



Network Terms

- Layer 4 Networking
- The transport layer controls the reliability of a given link through flow control. In TCP/IP, this layer refers to the TCP protocol for exchanging data over an unreliable network.



Network Terms

- Layer 7 Networking
- The application layer is the layer closest to the end user, which means both the application layer and the user interact directly with the software application.
- This layer interacts with software applications that implement a communicating component. Typically, Layer 7 Networking refers to HTTP.



Network Terms

- NAT — Network Address Translation
- NAT or network address translation is an IP-level remapping of one address space into another. The mapping happens by modifying network address information in the IP header of packets while they are in transit across a traffic routing device.



Network Terms

- SNAT — Source Network Address Translation
- SNAT simply refers to a NAT procedure that modifies the source address of an IP packet. This is the typical behaviour for the NAT described above.
- **DNAT — Destination Network Address Translation**
- DNAT refers to a NAT procedure that modifies the destination address of an IP packet. DNAT is used to publish a service resting in a private network to a publicly addressable IP address.

Network Terms

- Network Namespace
- In networking, each machine (real or virtual) has an Ethernet device (that we will refer to as eth0).
- All traffic flowing in and out of the machine is associated with that device.
- In truth, Linux associates each Ethernet device with a network namespace — a logical copy of the entire network stack, with its own routes, firewall rules, and network devices.
- Initially, all the processes share the same default network namespace from the init process, called the root namespace.
- By default, a process inherits its network namespace from its parent and so, if you don't make any changes, all network traffic flows through the Ethernet device specified for the root network namespace.



Network Terms

- veth — Virtual Ethernet Device Pairs
- Computer systems typically consist of one or more networking devices — eth0, eth1, etc — that are associated with a physical network adapter which is responsible for placing packets onto the physical wire.
- Veth devices are virtual network devices that are always created in interconnected pairs.
- They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.



Network Terms

- bridge — Network Bridge
- A network bridge is a device that creates a single aggregate network from multiple communication networks or network segments.
- Bridging connects two separate networks as if they were a single network.
- Bridging uses an internal data structure to record the location that each packet is sent to as a performance optimization.



Network Terms

- **iptables** — Packet Mangling Tool
- **iptables** is a program that allows a Linux system administrator to configure the netfilter and the chains and rules it stores.
- Each rule within an IP table consists of a number of classifiers (**iptables matches**) and one connected action (**iptables target**).



Network Terms

- IPVS — IP Virtual Server
- IPVS implements transport-layer load balancing as part of the Linux kernel.
- IPVS is a tool similar to iptables. It is based on the Linux kernel's netfilter hook function, but uses a hash table as the underlying data structure.
- That means, when compared to iptables, IPVS redirects traffic much faster, has much better performance when syncing proxy rules, and provides more load balancing algorithms.



Network Terms

- DNS — The Domain Name System
- The Domain Name System (DNS) is a decentralized naming system for associating system names with IP addresses.
- It translates domain names to numerical IP addresses for locating computer services.



The Kubernetes Networking Model

- Kubernetes makes opinionated choices about how Pods are networked. In particular, Kubernetes dictates the following requirements on any networking implementation:
 - all Pods can communicate with all other Pods without using network address translation (NAT).
 - all Nodes can communicate with all Pods without NAT.
 - the IP that a Pod sees itself as is the same IP that others see it as.

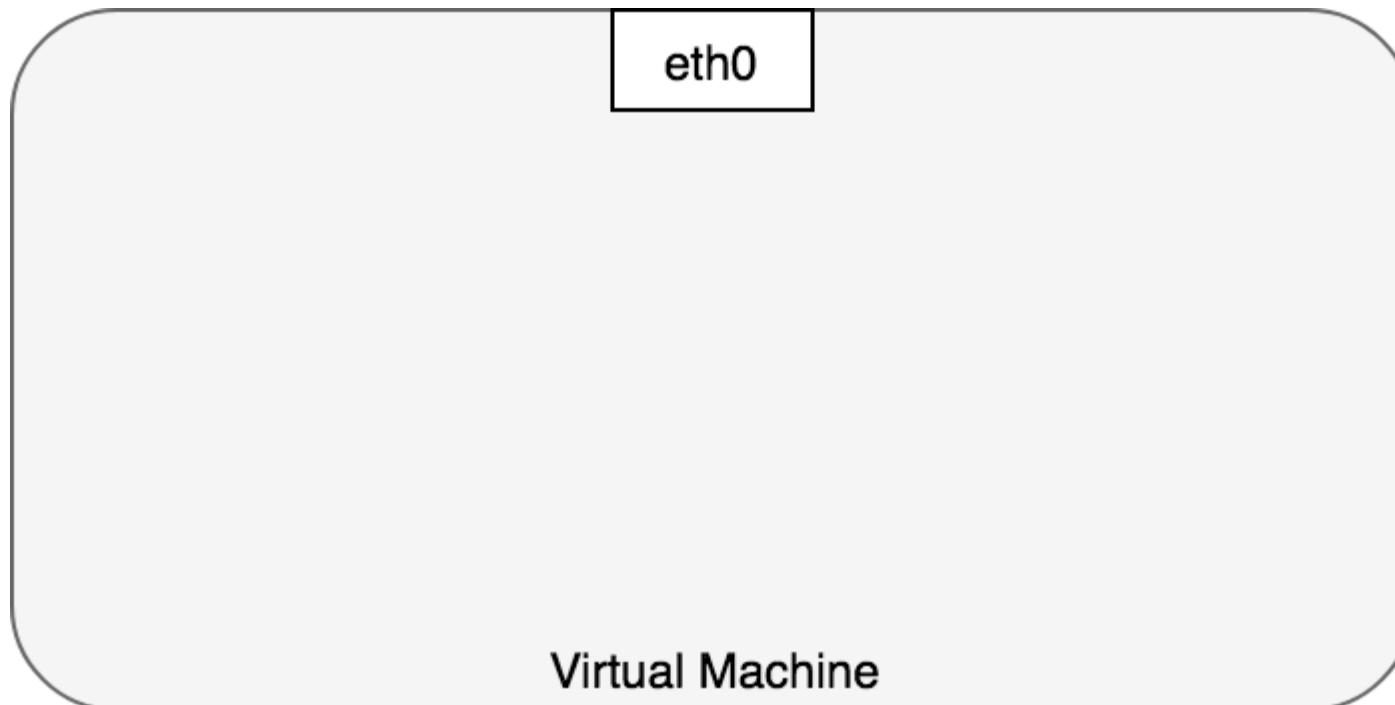


The Kubernetes Networking Model

- Given these constraints, we are left with four distinct networking problems to solve:
 - Container-to-Container networking
 - Pod-to-Pod networking
 - Pod-to-Service networking
 - Internet-to-Service networking



Container-to-Container Networking



- **Typically, we view network communication in a virtual machine as interacting directly with an Ethernet device,**



Container-to-Container Networking

- Linux, each running process communicates within a network namespace that provides a logical networking stack with its own routes, firewall rules, and network devices.
- In essence, a network namespace provides a brand new network stack for all the processes within the namespace.
- As a Linux user, network namespaces can be created using the ip command.
- For example, the following command will create a new network namespace called ns1.
- **ip netns add ns1**



Container-to-Container Networking

- When the namespace is created, a mount point for it is created under `/var/run/netns`, allowing the namespace to persist even if there is no process attached to it.
- You can list available namespaces by listing all the mount points under `/var/run/netns`, or by using the `ip` command.
 - `$ ls /var/run/netns`
 - `ns1`
 - `$ ip netns`
 - `ns1`

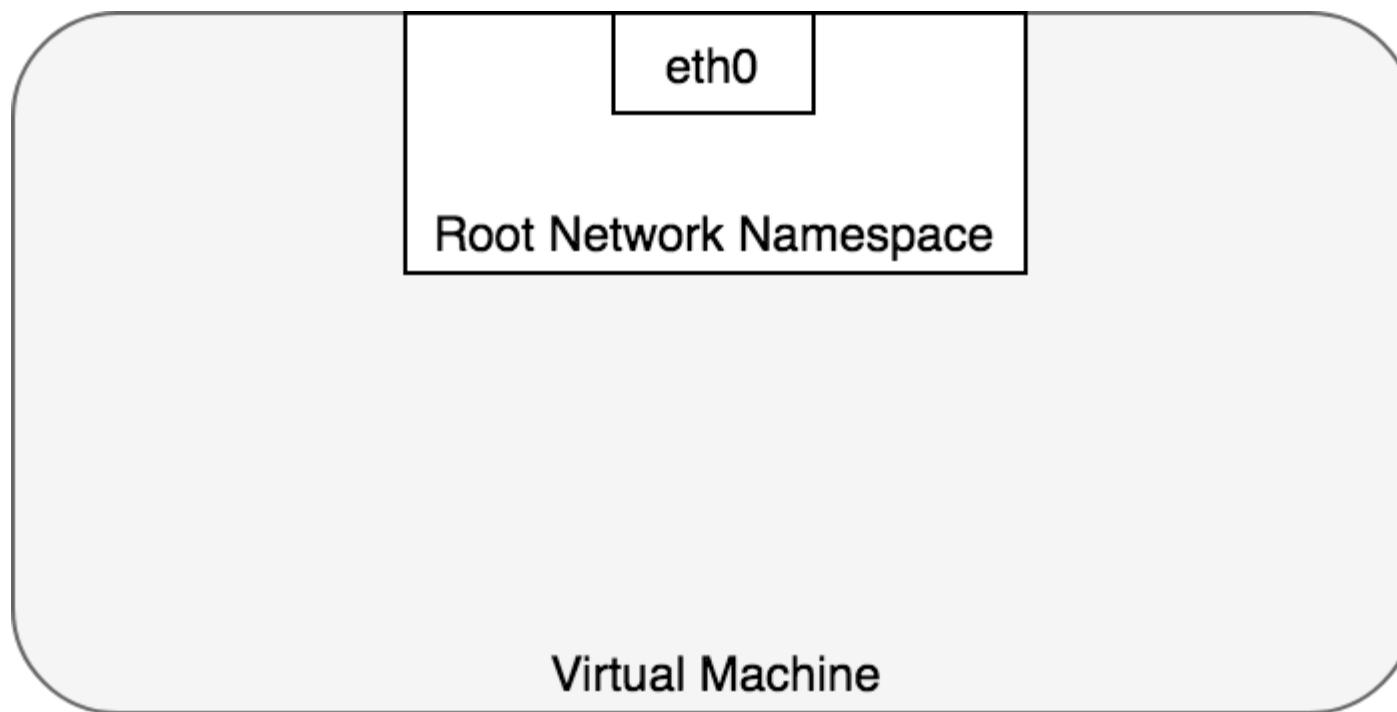
eswaribala@DESKTOP-55AGI0I: ~

```
This message is shown once once a day. To disable it please create the  
/home/eswaribala/.hushlogin file.
```

```
eswaribala@DESKTOP-55AGI0I:~$ ip netns add ns1  
mkdir /run/netns failed: Permission denied  
eswaribala@DESKTOP-55AGI0I:~$ sudo ip netns add ns1  
[sudo] password for eswaribala:  
eswaribala@DESKTOP-55AGI0I:~$ ls /var/run/netns  
ns1  
eswaribala@DESKTOP-55AGI0I:~$ ip netns  
ns1  
eswaribala@DESKTOP-55AGI0I:~$
```

Root Network Namespace

- By default, Linux assigns every process to the root network namespace to provide access to the external world



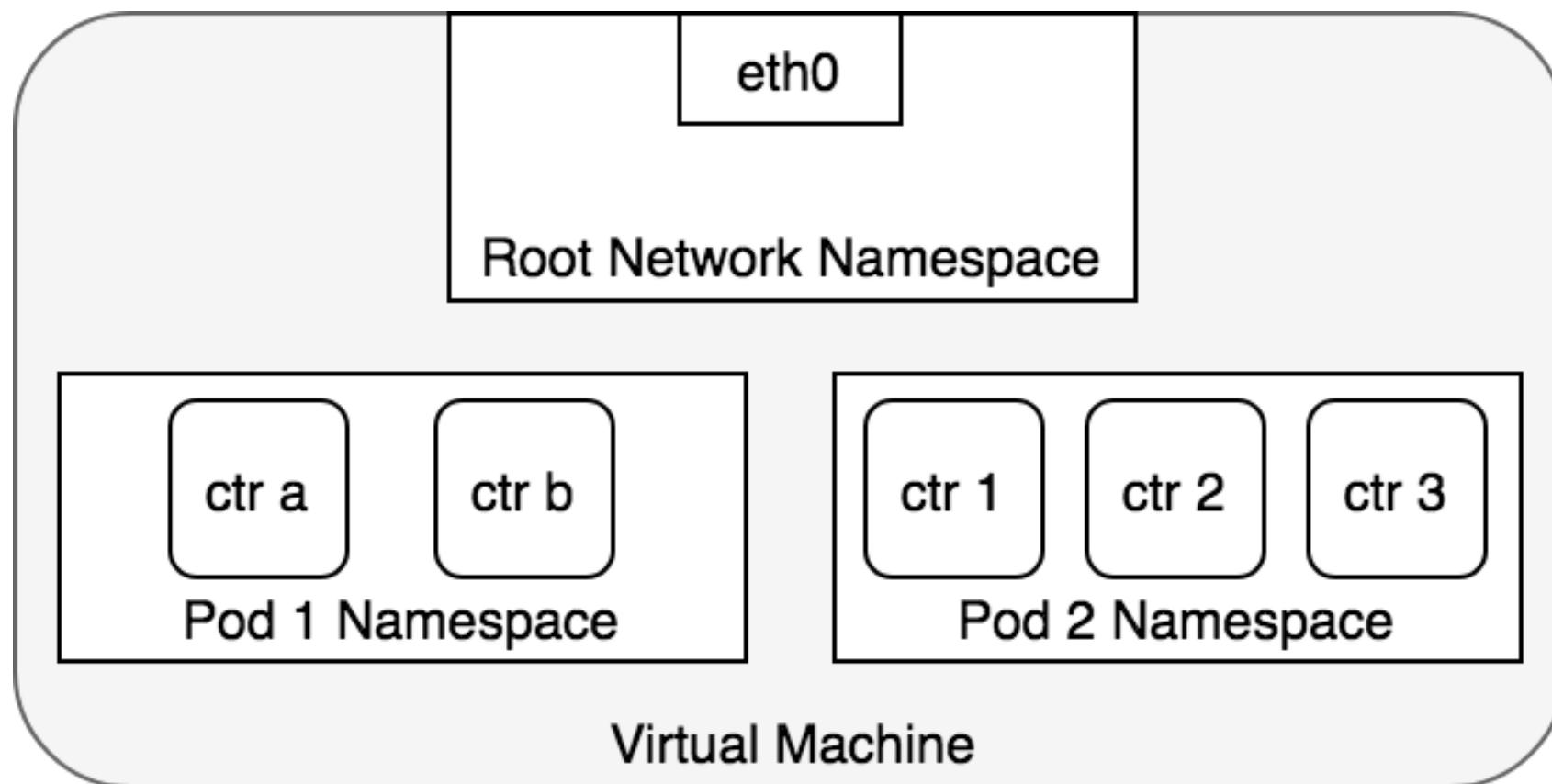


Network Namespace/pod

- In terms of Docker constructs, a Pod is modelled as a group of Docker containers that share a network namespace.
- Containers within a Pod all have the same IP address and port space assigned through the network namespace assigned to the Pod, and can find each other via localhost since they reside in the same namespace.
- We can create a network namespace for each Pod on a virtual machine.
- This is implemented, using Docker, as a “Pod container” which holds the network namespace open while “app containers” (the things the user specified) join that namespace with Docker’s `–net=container:` function.

Network Namespace/pod

Applications within a Pod also have access to shared volumes, which are defined as part of a Pod and are made available to be mounted into each application's filesystem.





Pod-to-Pod Networking

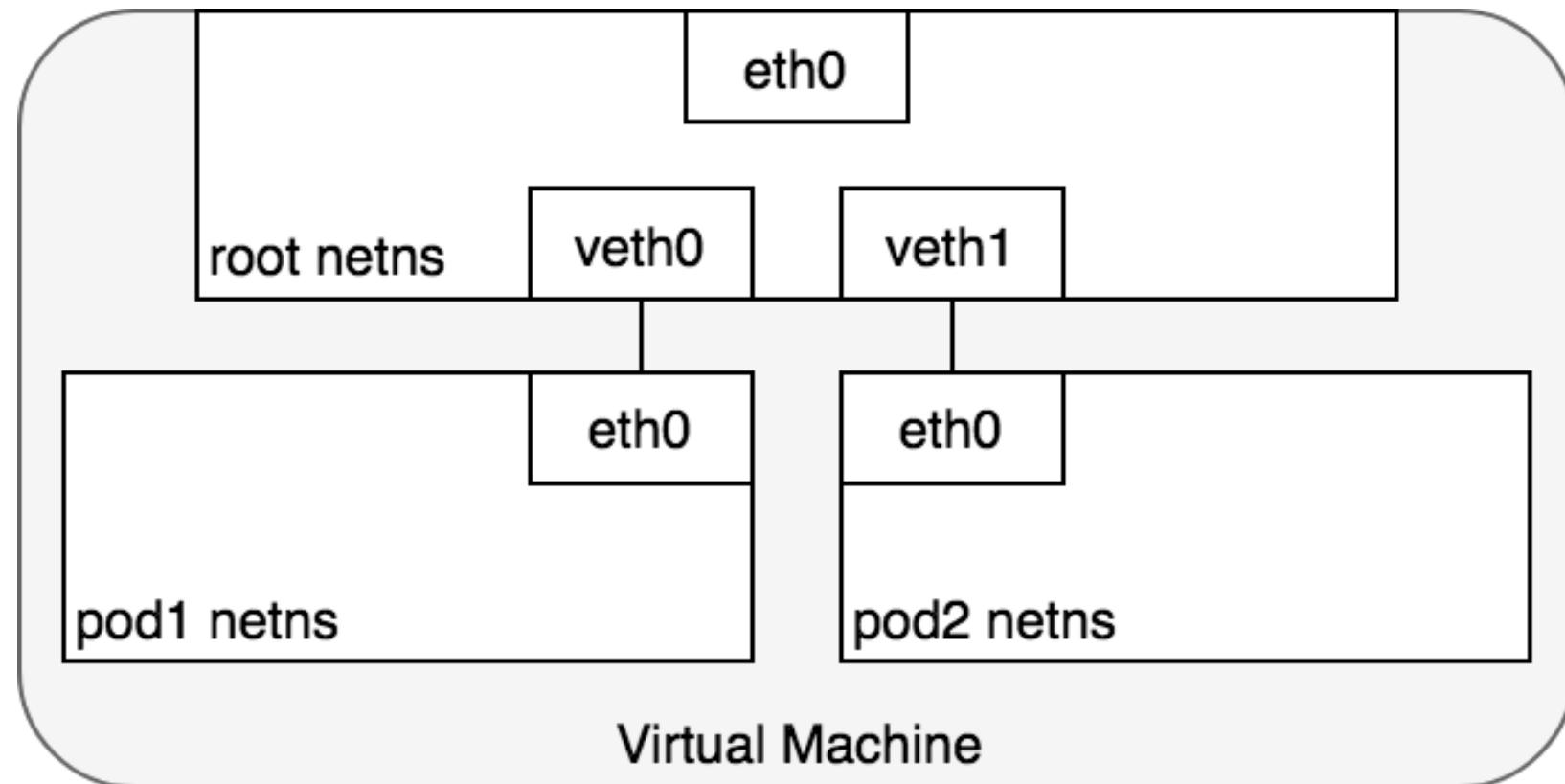
- In Kubernetes, every Pod has a real IP address and each Pod communicates with other Pods using that IP address.
- The task at hand is to understand how Kubernetes enables Pod-to-Pod communication using real IPs, whether the Pod is deployed on the same physical Node or different Node in the cluster.
- We are considering that pods reside on the same machine to avoid the complications of going over the internal network to communicate across Nodes.



Pod-to-Pod Networking

- From the Pod's perspective, it exists in its own Ethernet namespace that needs to communicate with other network namespaces on the same Node.
- Namespaces can be connected using a Linux Virtual Ethernet Device or veth pair consisting of two virtual interfaces that can be spread over multiple namespaces.
- To connect Pod namespaces, we can assign one side of the veth pair to the root network namespace, and the other side to the Pod's network namespace.
- Each veth pair works like a patch cable, connecting the two sides and allowing traffic to flow between them.
- This setup can be replicated for as many Pods as we have on the machine.

Pod-to-Pod Networking





Pod-to-Pod Networking

- At this point, we've set up the Pods to each have their own network namespace so that they believe they have their own Ethernet device and IP address, and they are connected to the root namespace for the Node.
- Now, we want the Pods to talk to each other through the root namespace, and for this we use a network bridge.



Pod-to-Pod Networking

- A Linux Ethernet bridge is a virtual Layer 2 networking device used to unite two or more network segments, working transparently to connect two networks together.
- The bridge operates by maintaining a forwarding table between sources and destinations by examining the destination of the data packets that travel through it and deciding whether or not to pass the packets to other network segments connected to the bridge.
- The bridging code decides whether to bridge data or to drop it by looking at the MAC-address unique to each Ethernet device in the network.

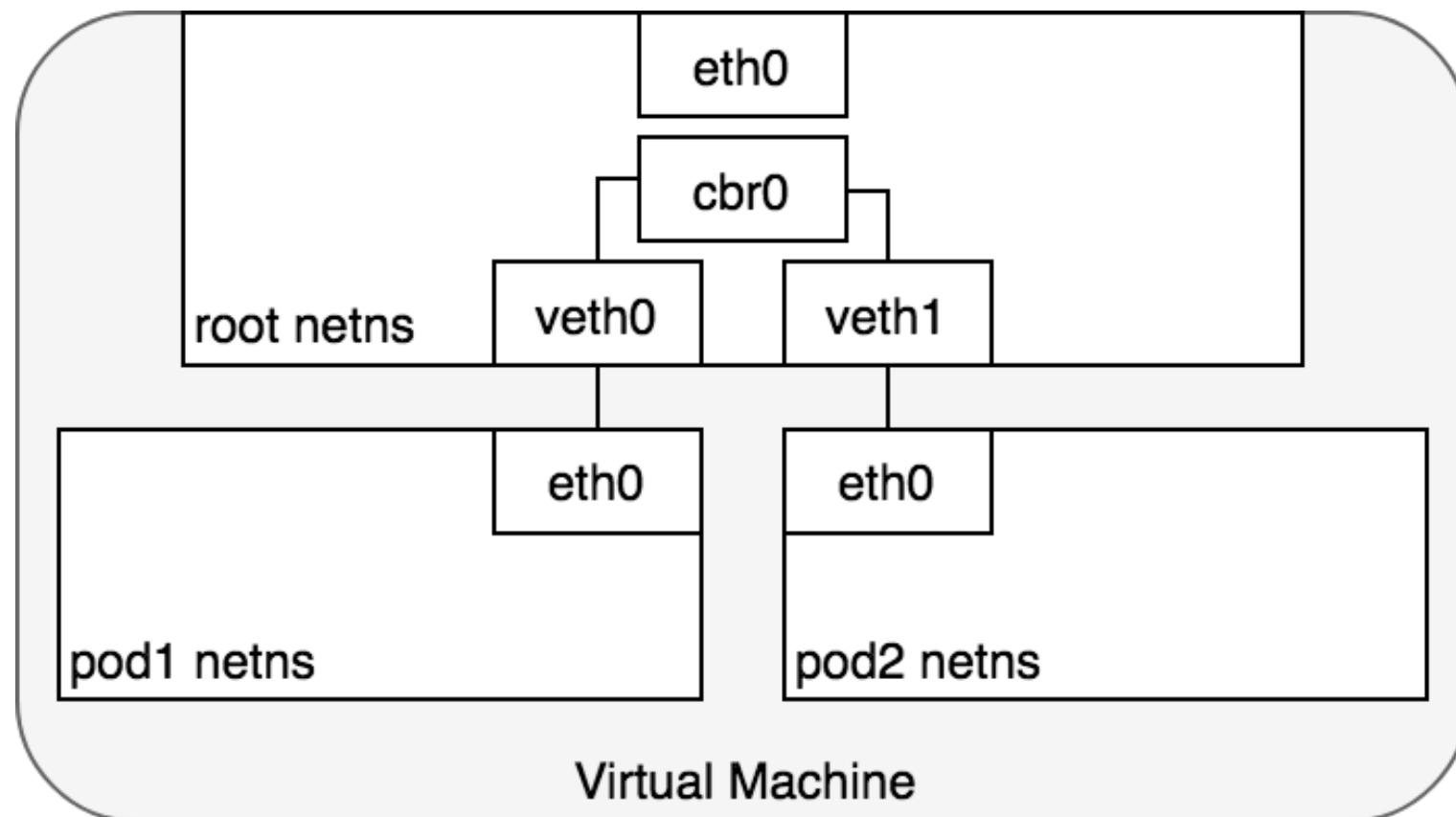


Pod-to-Pod Networking

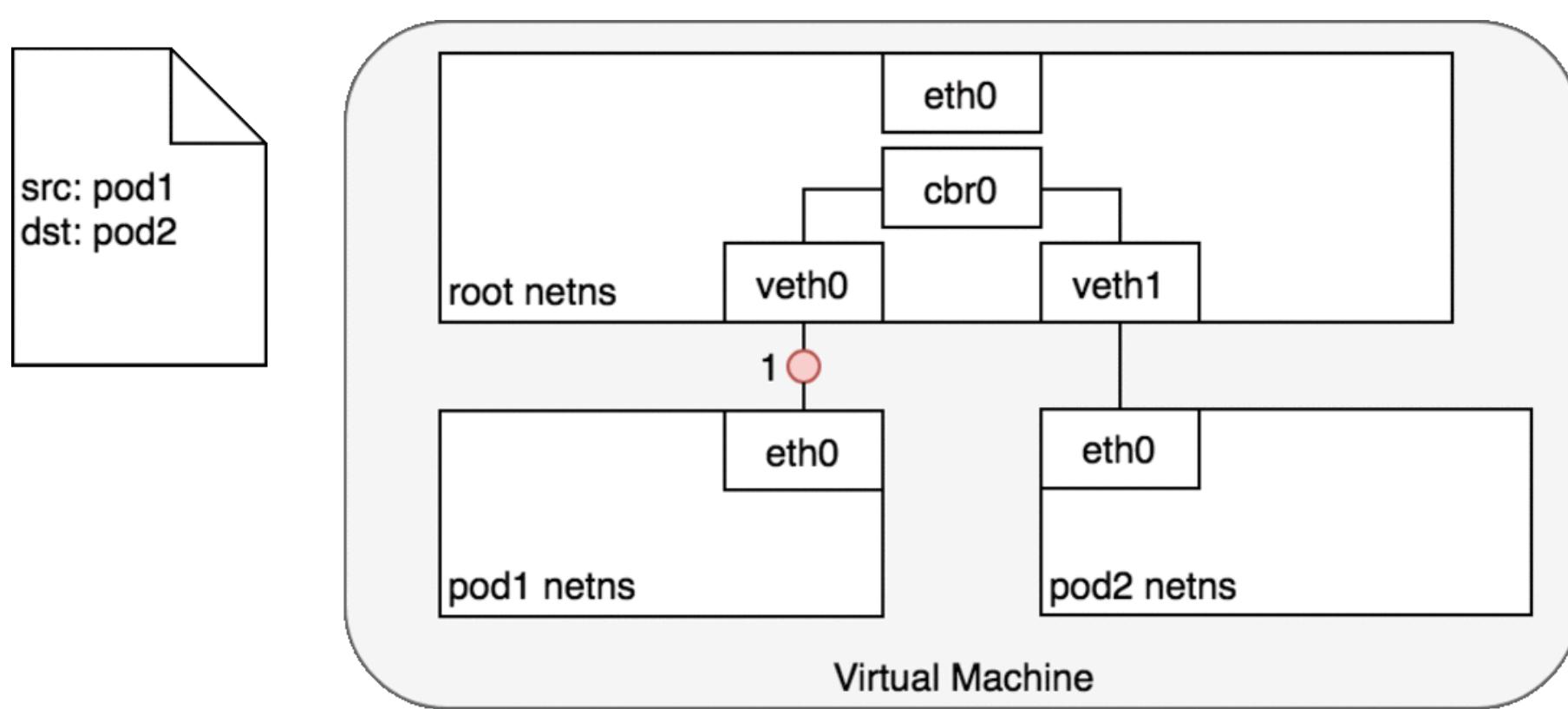
- Bridges implement the ARP protocol to discover the link-layer MAC address associated with a given IP address.
- When a data frame is received at the bridge, the bridge broadcasts the frame out to all connected devices (except the original sender) and the device that responds to the frame is stored in a lookup table.
- Future traffic with the same IP address uses the lookup table to discover the correct MAC address to forward the packet to.



Pod-to-Pod Networking



Life of a Packet





Life of a Packet

- Pod 1 sends a packet to its own Ethernet device eth0 which is available as the default device for the Pod.
- For Pod 1, eth0 is connected via a virtual Ethernet device to the root namespace, veth0 (1).
- The bridge cbr0 is configured with veth0 a network segment attached to it.
- Once the packet reaches the bridge, the bridge resolves the correct network segment to send the packet to — veth1 using the ARP protocol (3).



Life of a Packet

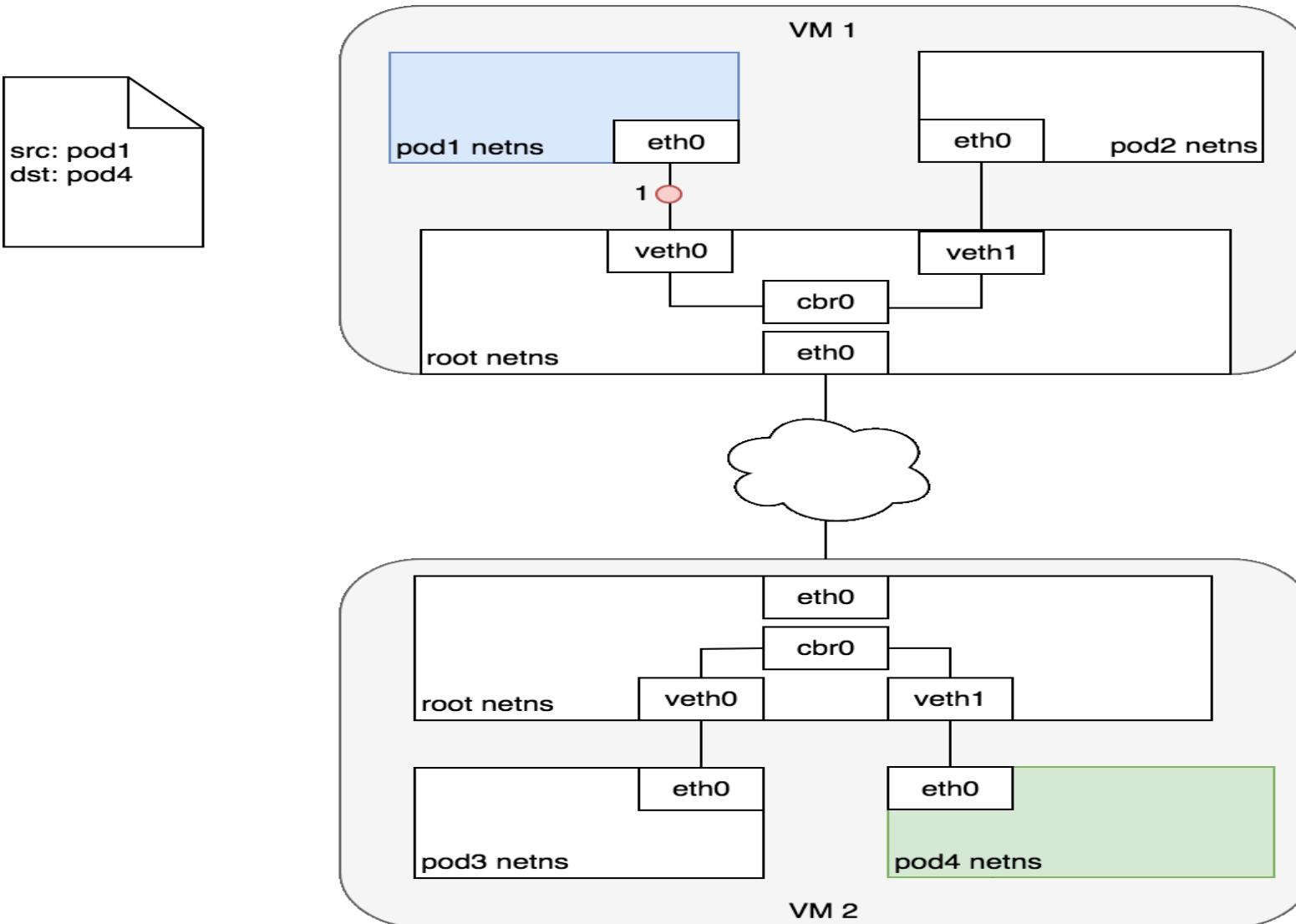
- When the packet reaches the virtual device veth1, it is forwarded directly to Pod 2's namespace and the eth0 device within that namespace (4).
- Throughout this traffic flow, each Pod is communicating only with eth0 on localhost and the traffic is routed to the correct Pod.
- The development experience for using the network is the default behaviour that a developer would expect.



Life of a Packet

- Kubernetes' networking model dictates that Pods must be reachable by their IP address across Nodes.
- That is, the IP address of a Pod is always visible to other Pods in the network, and each Pod views its own IP address as the same as how other Pods see it.
- We now turn to the problem of routing traffic between Pods on different Nodes.

Life of a packet: Pod-to-Pod, across Nodes





Pod-to-Service Networking

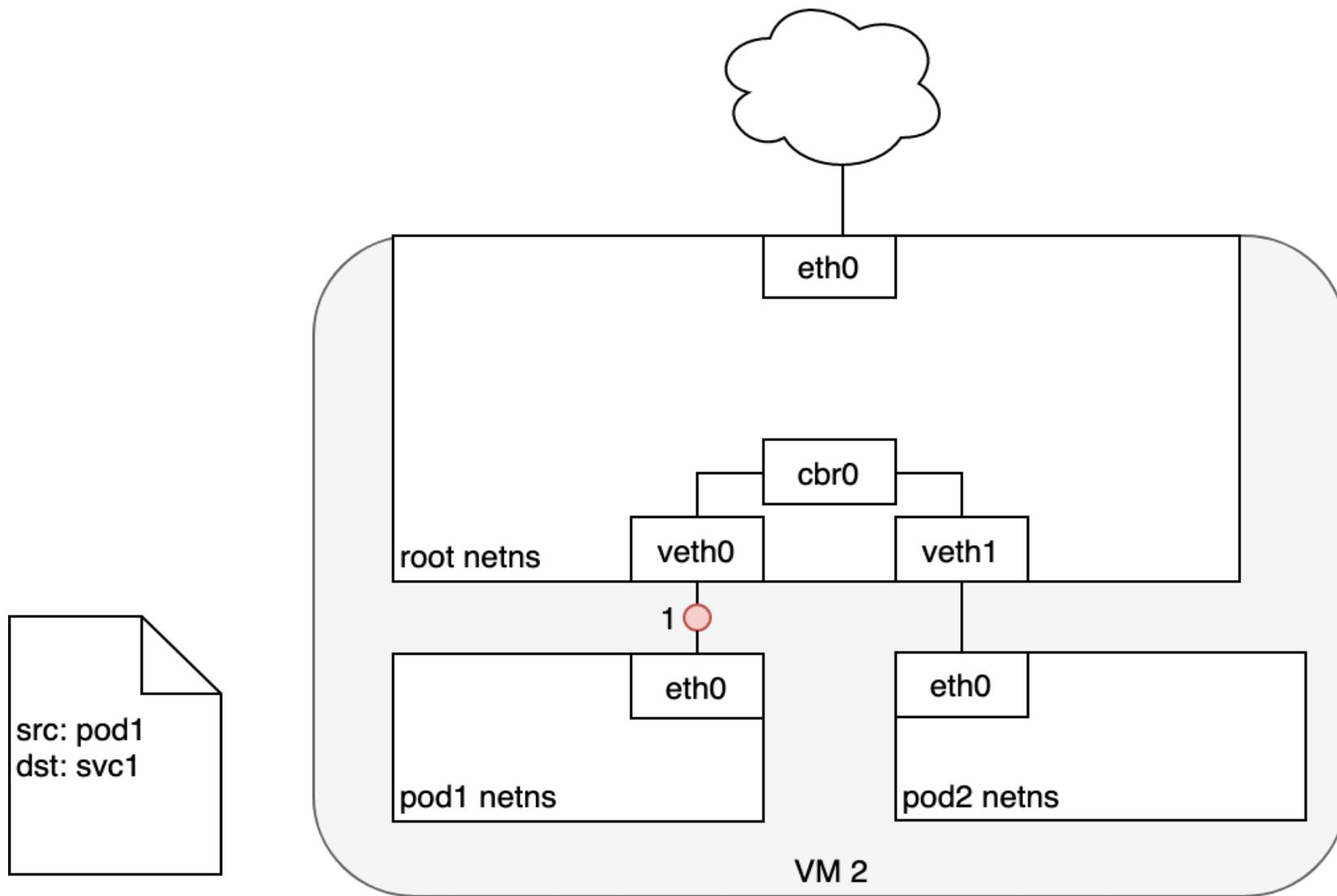
- A Kubernetes Service manages the state of a set of Pods, allowing you to track a set of Pod IP addresses that are dynamically changing over time.
- Services act as an abstraction over Pods and assign a single virtual IP address to a group of Pod IP addresses.
- Any traffic addressed to the virtual IP of the Service will be routed to the set of Pods that are associated with the virtual IP.
- This allows the set of Pods associated with a Service to change at any time — clients only need to know the Service's virtual IP, which does not change.



Pod-to-Service Networking

- When creating a new Kubernetes Service, a new virtual IP (also known as a cluster IP) is created on your behalf.
- Anywhere within the cluster, traffic addressed to the virtual IP will be load-balanced to the set of backing Pods associated with the Service.
- In effect, Kubernetes automatically creates and maintains a distributed in-cluster load balancer that distributes traffic to a Service's associated healthy Pods.

Pod to Service





Pod to Service

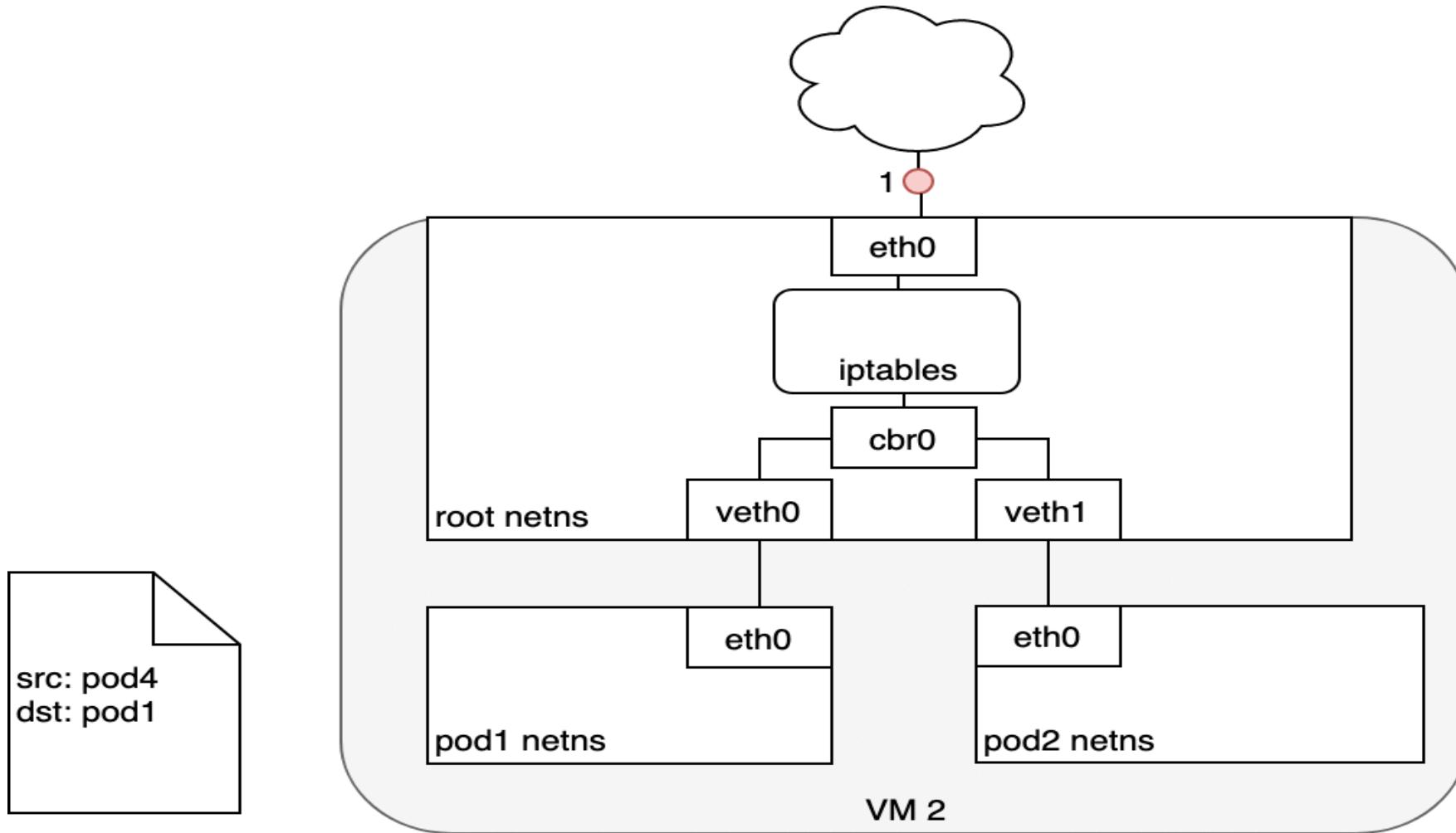
- The packet first leaves the Pod through the eth0 interface attached to the Pod's network namespace (1).
- Then it travels through the virtual Ethernet device to the bridge (2).
- The ARP protocol running on the bridge does not know about the Service and so it transfers the packet out through the default route — eth0 (3).
- Here, something different happens. Before being accepted at eth0, the packet is filtered through iptables.
- After receiving the packet, iptables uses the rules installed on the Node by kube-proxy in response to Service or Pod events to rewrite the destination of the packet from the Service IP to a specific Pod IP (4).



Pod to Service

- The packet is now destined to reach Pod 4 rather than the Service's virtual IP.
- The Linux kernel's conntrack utility is leveraged by iptables to remember the Pod choice that was made so future traffic is routed to the same Pod (barring any scaling events).
- In essence, iptables has done in-cluster load balancing directly on the Node. Traffic then flows to the Pod using the Pod-to-Pod routing we've already examined (5).

Service to POD





Using DNS

- Kubernetes can optionally use DNS to avoid having to hard-code a Service's cluster IP address into your application.
- Kubernetes DNS runs as a regular Kubernetes Service that is scheduled on the cluster.
- It configures the kubelets running on each Node so that containers use the DNS Service's IP to resolve DNS names.
- Every Service defined in the cluster (including the DNS server itself) is assigned a DNS name.
- DNS records resolve DNS names to the cluster IP of the Service or the IP of a POD, depending on your needs.
- SRV records are used to specify particular named ports for running Services.



Using DNS

- A DNS Pod consists of three separate containers:
- `kubedns`: watches the Kubernetes master for changes in Services and Endpoints, and maintains in-memory lookup structures to serve DNS requests.
- `dnsmasq`: adds DNS caching to improve performance.
- `sidecar`: provides a single health check endpoint to perform healthchecks for `dnsmasq` and `kubedns`.

Ingress — Routing Internet traffic to Kubernetes

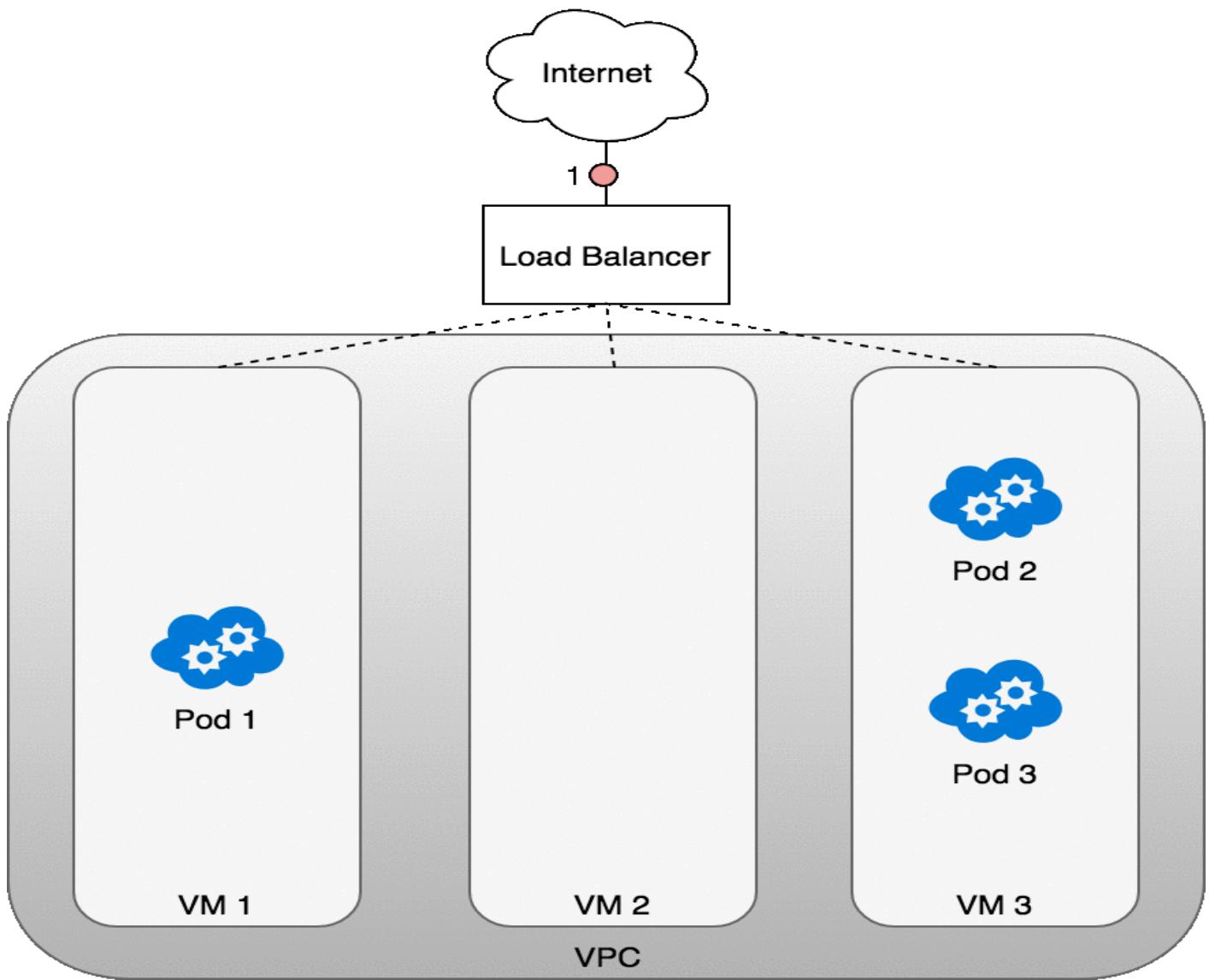


- Ingress — getting traffic into your cluster — is a problem to solve.
- Again, this is specific to the network you are running, but in general, Ingress is divided into two solutions that work on different parts of the network stack:
- (1) a Service LoadBalancer and
- (2) an Ingress controller.

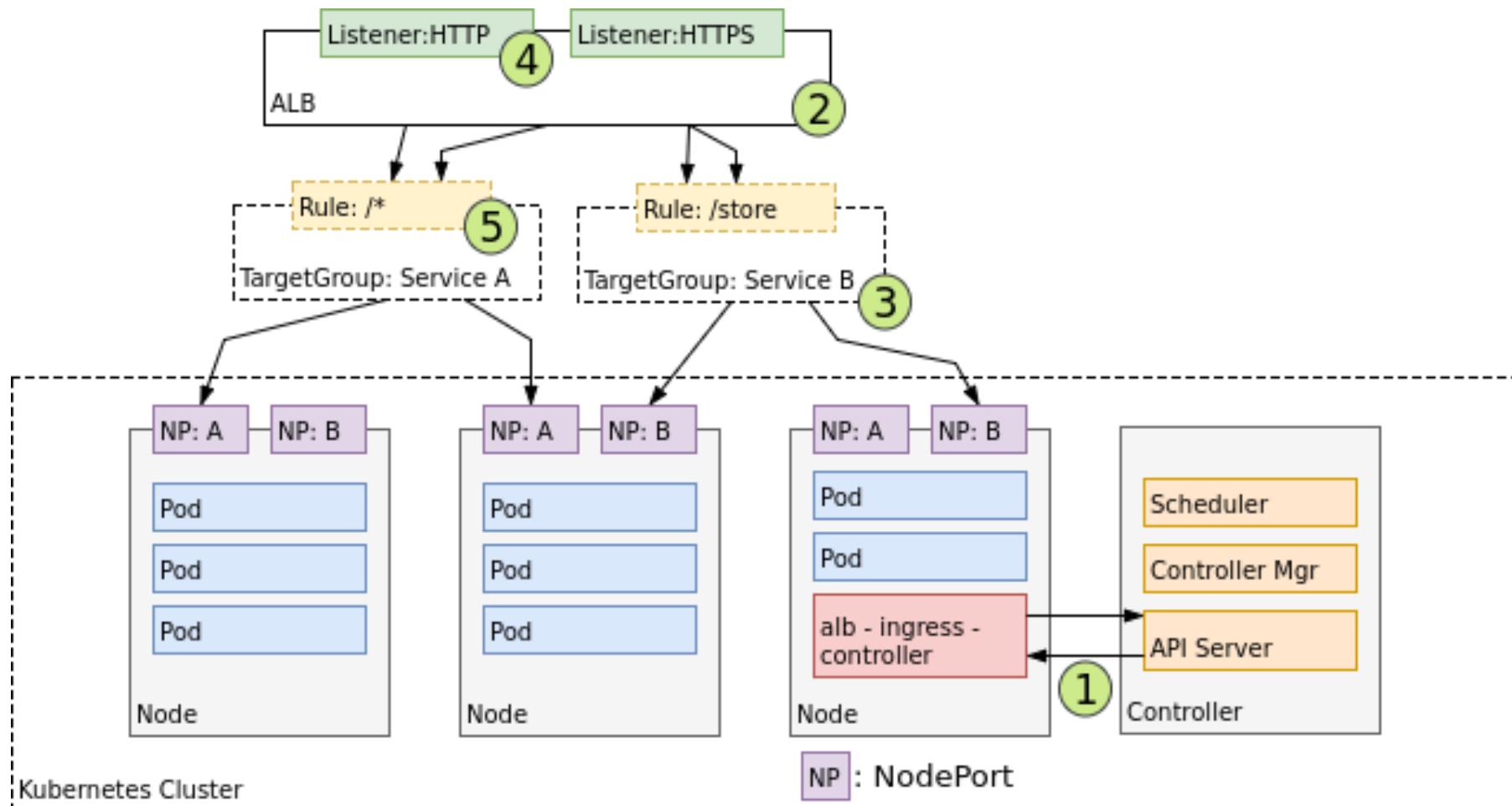


Layer 4 Ingress: LoadBalancer

- When you create a Kubernetes Service you can optionally specify a LoadBalancer to go with it.
- The implementation of the LoadBalancer is provided by a cloud controller that knows how to create a load balancer for your service.
- Once your Service is created, it will advertise the IP address for the load balancer.
- As an end user, you can start directing traffic to the load balancer to begin communicating with your Service



Ingress Controller

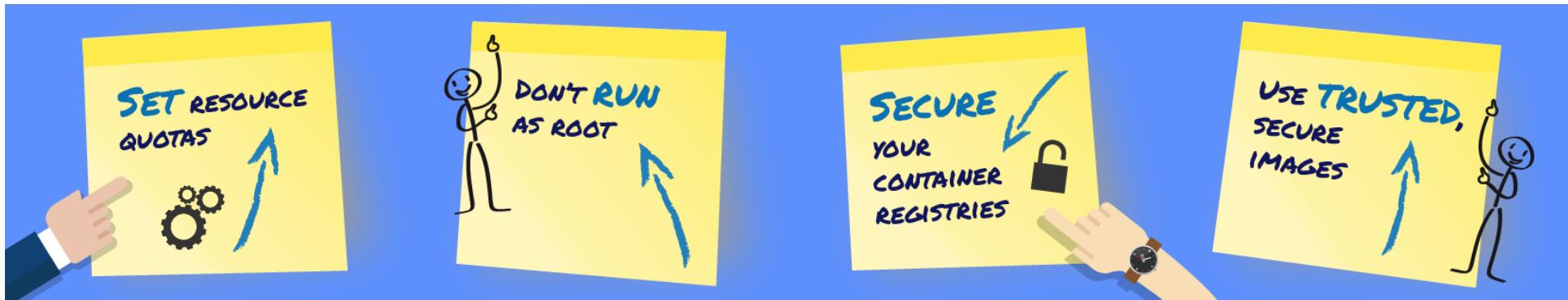




Docker Security

- Docker container security is more complicated, largely because a typical Docker environment has many more moving parts. Those parts include:
 - Your containers. You probably have multiple Docker container images, each hosting individual microservices. You probably also have multiple instances of each image running at a given time. Each of those images and instances needs to be secured and monitored separately.
 - The Docker daemon, which needs to be secured to keep the containers it hosts safe.
 - The host server, which could be bare metal or a virtual machine.
 - If you host your containers in the cloud using a service like ECS, that is another layer to secure.
 - Overlay networks and APIs that facilitate communication between containers.
 - Data volumes or other storage systems that exist externally from your containers

Docker Security





Docker Security

- Whether image comes from an official one, such as NGINX, Redis, Ubuntu, or Alpine Linux; or one from a total stranger.
- USE PRIVATE OR TRUSTED REPOSITORIES
- In Docker Hub, the images are always scanned and reviewed by Docker's Security Scanning Service.
- Docker Cloud and Docker Hub can scan images in private repositories to verify that they are free from known security vulnerabilities or exposures, and report the results of the scan for each image tag.



Docker Security

- USE DOCKER CONTENT TRUST
- Before a publisher pushes an image to a remote registry, Docker Engine signs the image locally with the publisher's private key.
- When you later pull this image, Docker Engine uses the publisher's public key to verify that the image you are about to run is exactly what the publisher created, has not been tampered with and is up to date.



Docker Security

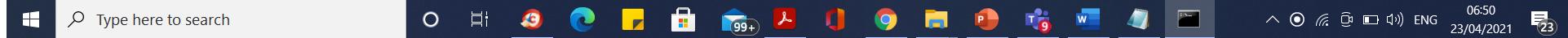
- Docker Bench Security
- The tool was based on the recommendations in the CIS Docker 1.13 Benchmark, and run checks against the following six areas:
 - Host configuration
 - Docker daemon configuration
 - Docker daemon configuration files
 - Container images and build files
 - Container runtime
 - Docker security operations



Docker Security

```
C:\WINDOWS\system32>docker run -it --net host --pid host --cap-add audit_control -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST -v /var/lib:/var/lib -v /var/run/docker.sock:/var/run/docker.sock -v /etc:/etc --label docker_bench_security docker/docker-bench-security
Unable to find image 'docker/docker-bench-security:latest' locally
latest: Pulling from docker/docker-bench-security
cd784148e348: Pull complete
48fe0d48816d: Pull complete
164e5e0f48c5: Pull complete
378ed37ea5ff: Pull complete
Digest: sha256:ddbdf4f86af4405da4a8a7b7cc62bb63bfcb75e85bf22d2ece70c204d7cfabb8
Status: Downloaded newer image for docker/docker-bench-security:latest
#
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
#
Initializing Fri Apr 23 01:14:11 UTC 2021

[INFO] 1 - Host Configuration
[WARN] 1.1 - Ensure a separate partition for containers has been created
[NOTE] 1.2 - Ensure the container host has been Hardened
[INFO] 1.3 - Ensure Docker is up to date
[INFO] * Using 20.10.5, verify it is up to date as deemed necessary
[INFO] * Your operating system vendor may provide support and security maintenance for Docker
[INFO] 1.4 - Ensure only trusted users are allowed to control Docker daemon
[WARN] 1.5 - Ensure auditing is configured for the Docker daemon
[WARN] 1.6 - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.7 - Ensure auditing is configured for Docker files and directories - /etc/docker
[INFO] 1.8 - Ensure auditing is configured for Docker files and directories - docker.service
```





Docker Security

```
C:\ Administrator: Command Prompt
f-4ced-4612-8fac-d0968e03f8cd_12
[WARN]      * PIDs limit not set: kind-worker
[WARN]      * PIDs limit not set: kind-control-plane
[WARN]      * PIDs limit not set: kind-worker2
[PASS] 5.29 - Ensure Docker's default bridge docker0 is not used
[PASS] 5.30 - Ensure the host's user namespaces is not shared
[PASS] 5.31 - Ensure the Docker socket is not mounted inside any containers

[INFO] 6 - Docker Security Operations
[INFO] 6.1 - Avoid image sprawl
[INFO]      * There are currently: 44 images
[INFO] 6.2 - Avoid container sprawl
[INFO]      * There are currently a total of 126 containers, with only 20 of them currently running

[INFO] 7 - Docker Swarm Configuration
[WARN] 7.1 - Ensure swarm mode is not Enabled, if not needed
[PASS] 7.2 - Ensure the minimum number of manager nodes have been created in a swarm
[WARN] 7.3 - Ensure swarm services are binded to a specific host interface
[WARN] 7.4 - Ensure data exchanged between containers are encrypted on different nodes on the overlay network
[WARN]      * Unencrypted overlay network: ingress (swarm)
[INFO] 7.5 - Ensure Docker's secret management commands are used for managing secrets in a Swarm cluster
[WARN] 7.6 - Ensure swarm manager is run in auto-lock mode
[NOTE] 7.7 - Ensure swarm manager auto-lock key is rotated periodically
[INFO] 7.8 - Ensure node certificates are rotated as appropriate
[INFO] 7.9 - Ensure CA certificates are rotated as appropriate
[INFO] 7.10 - Ensure management plane traffic has been separated from data plane traffic

[INFO] Checks: 105
[INFO] Score: -5

C:\WINDOWS\system32>
```



Docker Security

- With respect to Docker focus on two points:
 - Containers running in privileged mode
 - Excess privileges used by containers
- Starting with the first point, you can run a Docker container with the --privileged switch.
- What this does is give extended privileges to this container.
- It gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller.
- In other words, the container can then do almost everything that the host can do.
- This flag exists to allow special use-cases, like running Docker within Docker.



Docker Security

- DROP UNNECESSARY PRIVILEGES AND CAPABILITIES.
- Consider these questions:
 - What kind of network connectivity does your application need?
 - Does it need raw socket access?
 - Does it need to send and receive UDP requests? If not, then deny it those capabilities.
 - If not, then deny it those capabilities.
- To do so, make use of the --cap-drop and --cap-add switches.



Docker Security

- docker run \
- --cap-drop SETPCAP \
- --cap-drop NET_BIND_SERVICE \
- --cap-add SYS_MODULE \
- -ti /bin/sh



Docker Security

- System Security
- While Docker is increasingly very safe by default, thanks to having namespaces and cgroups as its foundation, you don't need to only rely on these features.
- You can go further and make use of other Linux security options, such as AppArmor, SELinux, grsecurity and Seccomp.



Docker Security

- AppArmor is a Linux kernel security module that allows the system administrator to restrict programs' capabilities with per-program profiles.
- Profiles can allow capabilities like network access, raw socket access, and the permission to read, write, or execute files on matching paths.



Docker Security

- Limit Available Resource Consumption
- What does your application need?
- Is it a quite light application, requiring no more than 50MB of memory?
- Then why let it access more?
- Does it perform more intensive processing, requiring 4+ CPUs?
- Then allow it to have access to that.



Docker Security

- Assuming that analysis, profiling, and benchmarking are part of your continuous development processes, then this information will be readily available.
- As a result, when deploying your containers, ensure that they only have the resources that they need.



Docker Security

- To do so, use the applicable docker run switches, such as the following:
 - -m / --memory: Set a memory limit
 - --memory-reservation: Set a soft memory limit
 - --kernel-memory: Set a kernel memory limit
 - --cpus: Limit the number of CPUs
 - --device-read-bps: Limit the read rate from a device



Docker Security

- Large Attack Surfaces
- As Docker allows you to create and deploy applications so quickly — as well as to destroy them with equal ease — it can be difficult to know exactly what applications your organization has deployed.
- Given that, the potential of a large attack surface grows. Not sure about the deployment statistics of your organization?
- To help you out, ask yourself the following questions:
- What applications are currently deployed by your organization?
- Who deployed them?



Docker Security

- When were they deployed?
- Why were they deployed?
- How long are they to be deployed for?
- Who's responsible for them?
- When was a security scan last run on them?



Docker Security

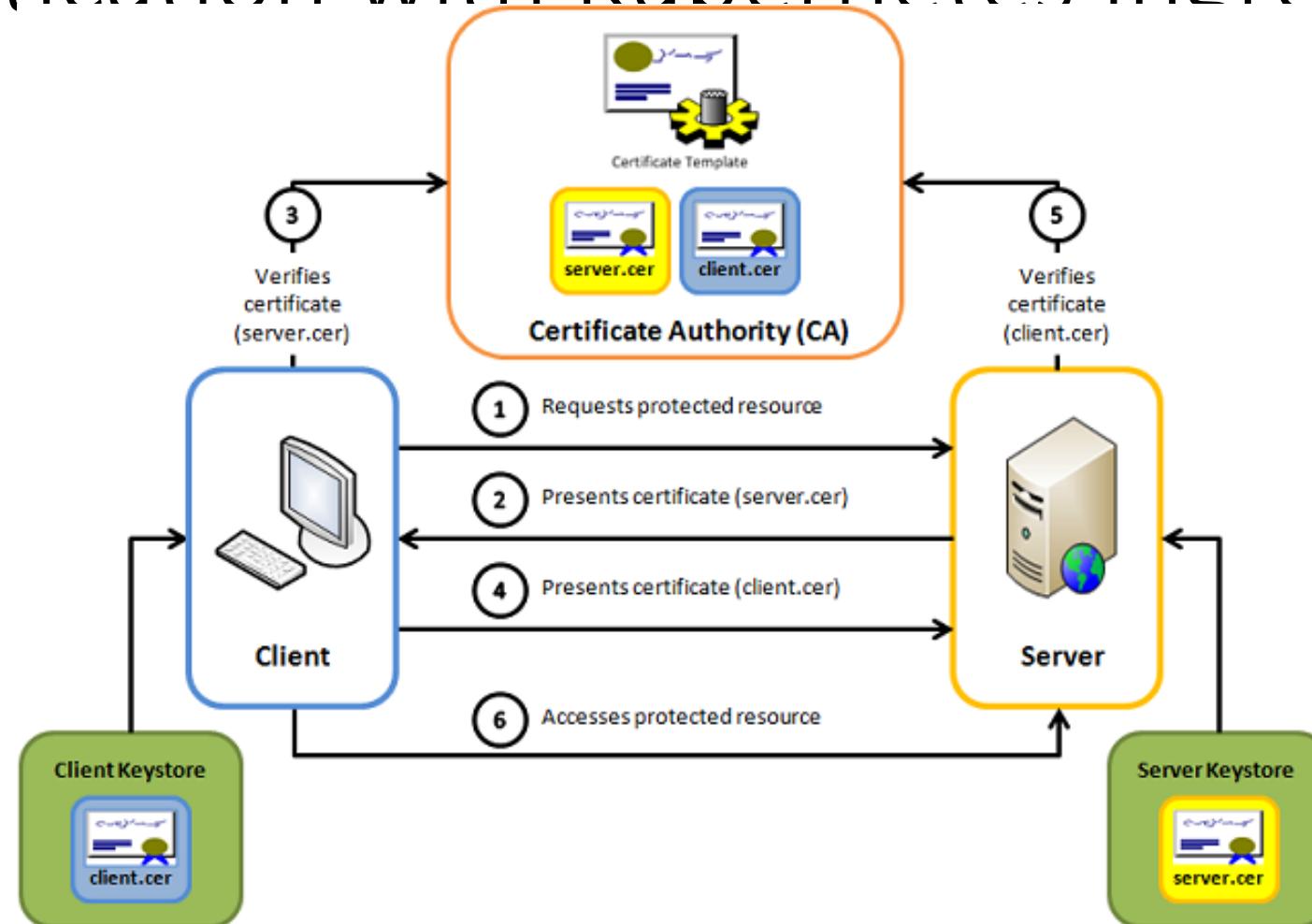
- IMPLEMENT AN AUDIT TRAIL WITH PROPER LOGGING
- As with audit trails within applications, such as when a user created their account when it was activated, and when the user last updated their password, and those more widely within organizations, consider implementing an audit trail around every container that your organization creates and deploys.

Configuring Certificate-Based Mutual Authentication with Kubernetes Ingress-Nginx



- Mutual Authentication
 - Mutual authentication is also known as 2-way authentication. It is a process in which both the client and server verify each others identity via a Certificate Authority.
 - Mutual SSL authentication or certificate based mutual authentication refers to two parties authenticating each other through verifying the provided digital certificate so that both parties are assured of the others' identity.

Configuring Certificate-Based Mutual Authentication with Kubernetes Ingress-Nginx



Mutual SSL authentication / Certificate based mutual authentication



Setting Up Mutual Authentication

- Creating the Certificates
- CommonName(CN): Identifies the hostname or owner associated with the certificate.
- Certificate Authority(CA): A trusted 3rd party that issues Certificates. Usually you would obtain this from a trusted source, but for this example we will just create one. The CN is usually the name of the issuer.
- Server Certificate: A Certificate used to identify the server. The CN here is the hostname of the server. The Server Certificate is valid only if it is installed on a server where the hostname matches the CN.
- Client Certificate: A Certificate used to identify a client/user. The CN here is usually the name of the client/user.



Setting Up Mutual Authentication

- Generate the CA Key and Certificate
- ```
$ openssl req -x509 -sha256 -newkey rsa:4096 -keyout ca.key -out ca.crt -days 356 -nodes -subj '/CN=Fern Cert Authority'
```
- # Generate the Server Key, and Certificate and Sign with the CA Certificate
- ```
$ openssl req -new -newkey rsa:4096 -keyout server.key -out server.csr -nodes -subj '/CN=meow.com'
```
- ```
$ openssl x509 -req -sha256 -days 365 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
```
- # Generate the Client Key, and Certificate and Sign with the CA Certificate
- ```
$ openssl req -new -newkey rsa:4096 -keyout client.key -out client.csr -nodes -subj '/CN=Fern'
```
- ```
$ openssl x509 -req -sha256 -days 365 -in client.csr -CA ca.crt -CAkey ca.key -set_serial 02 -out client.crt
```



# Setting Up Mutual Authentication

- Creating the Kubernetes Secrets
- `$ kubectl create secret generic my-certs --from-file=tls.crt=server.crt - -from-file=tls.key=server.key --from-file=ca.crt=ca.crt`
- `$ kubectl get secret my-certs`
- | NAME     | TYPE   | DATA | AGE |
|----------|--------|------|-----|
| my-certs | Opaque | 3    | 1m  |

# Deploying an Application with Mutual Authentication

---



- Refer docker commands.txt



# TLS and mutual authentication

- <https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/>



# Create Secret

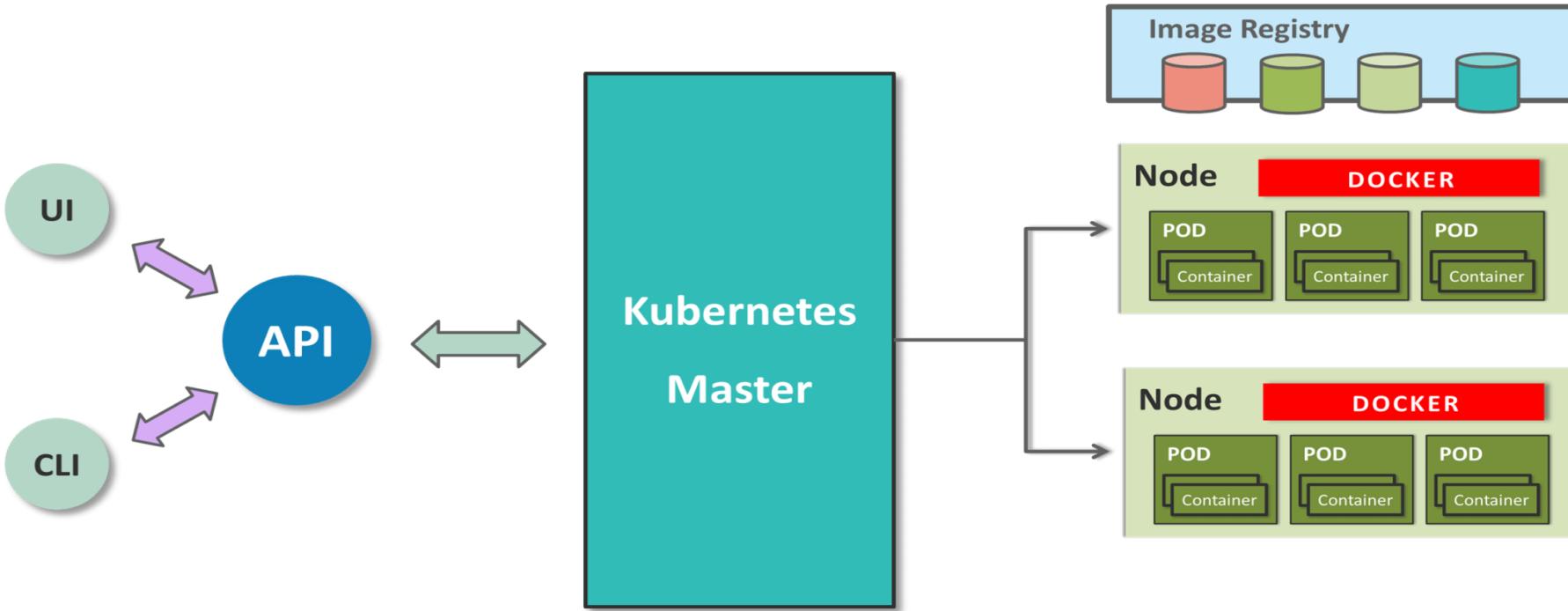
---

- <https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/>



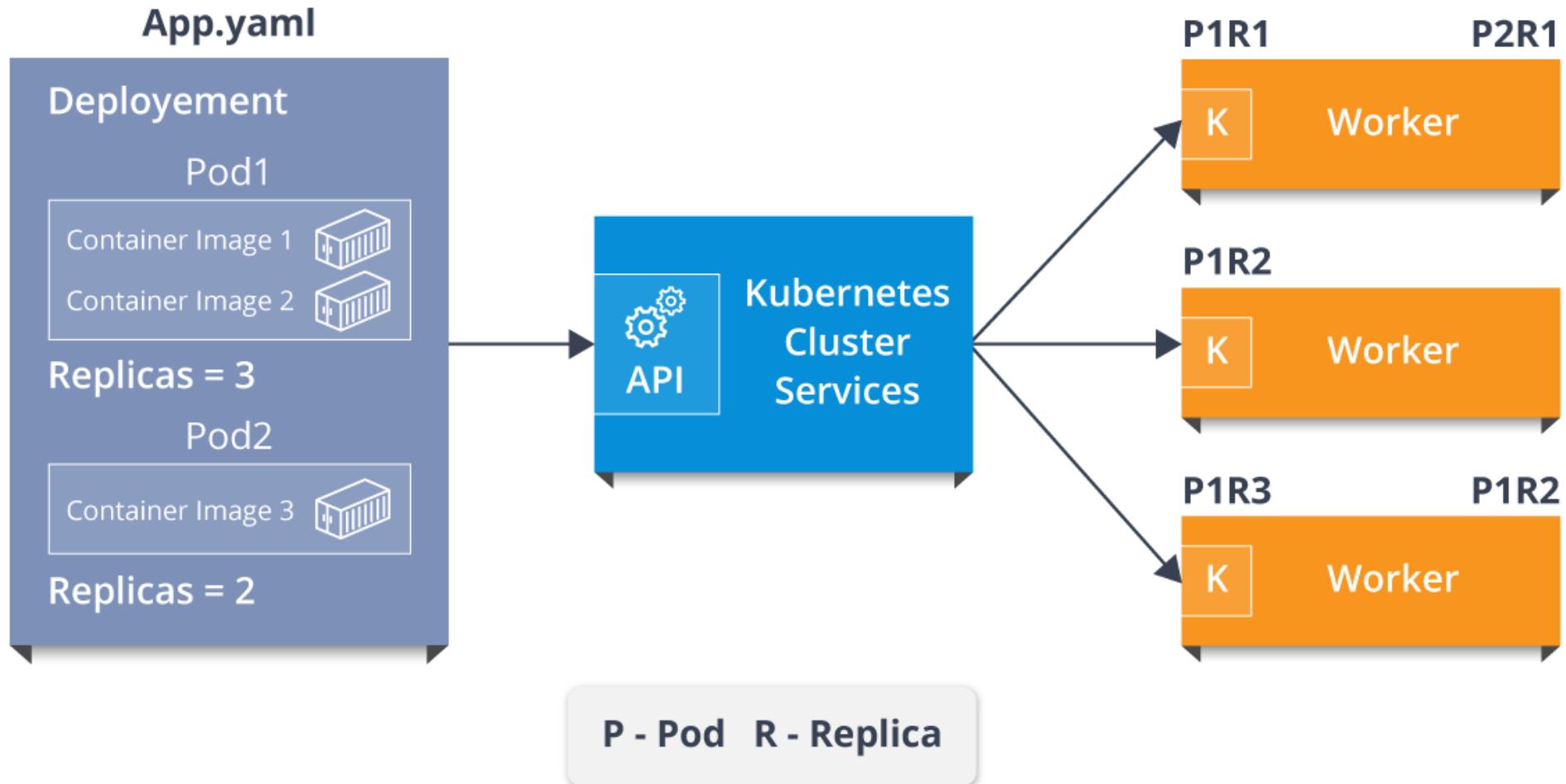
# Overview of RBAC concepts in Kubernetes

- The RBAC model in Kubernetes is based on three elements:
  - Roles: definition of the permissions for each Kubernetes resource type
  - Subjects: users (human or machine users) or groups of users
  - RoleBindings: definition of what Subjects have which Roles



Pod has group of containers that are run on the same host. So, if we regularly deploy single containers, then our container and the pod will be one and the same.

# Representation Of Kubernetes Cluster



# Different types of services in Kubernetes

| Cluster IP                                                                                                                                                                                                      | Node Port                                                                                                                                                                                                 | Load Balancer                                                                                                                                                                                                           | External Name                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• Exposes the service on a cluster-internal IP.</li><li>• Makes the service only reachable from within the cluster.</li><li>• This is the default Service Type.</li></ul> | <ul style="list-style-type: none"><li>• Exposes the service on each Node's IP at a static port.</li><li>• A Cluster IP service to which Node Port service will route, is automatically created.</li></ul> | <ul style="list-style-type: none"><li>• Exposes the service externally using a cloud provider's load balancer.</li><li>• Services, to which the external load balancer will route, are automatically created.</li></ul> | <ul style="list-style-type: none"><li>• Maps the service to the contents of the External Name field by returning a CNAME record with its value.</li><li>• No proxying of any kind is set up.</li></ul> |

# Minikube –p cluster1 dashboard

The screenshot shows a web browser window displaying the Kubernetes Dashboard. The URL in the address bar is `127.0.0.1:52161/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/#!/overview?namespace=default`. The dashboard has a blue header with the Kubernetes logo and a search bar. On the left, there's a sidebar with navigation links for Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, and Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers). The main content area is titled "Overview" and contains two tables: "Services" and "Secrets".

**Services**

| Name       | Labels                                       | Cluster IP | Internal endpoints | External endpoints | Age       |
|------------|----------------------------------------------|------------|--------------------|--------------------|-----------|
| kubernetes | component: apiserver<br>provider: kubernetes | 10.96.0.1  | kubernetes:443 TCP | -                  | 6 minutes |

**Secrets**

| Name                | Type                                | Age       |
|---------------------|-------------------------------------|-----------|
| default-token-r827j | kubernetes.io/service-account-token | 6 minutes |

## Openshift

Openshift is built on top of kubernetes. Openshift project is maintained by Redhat. It has both open source (openshift origin) and enterprise version (openshift container platform).

Along with core Kubernetes features, it offers out of the box components for container management and orchestration.

