

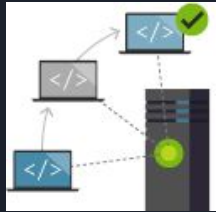


# MLOPS Tools

By: Ashish Pal

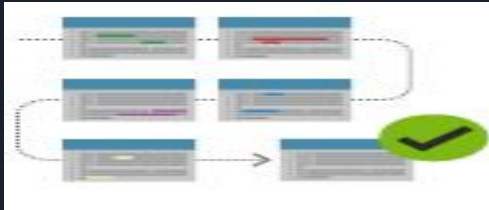
# What is version control?

Version control systems are software that help track changes made in code over time. As a developer edits code, the version control system takes a snapshot of the files. It then saves that snapshot permanently so it can be recalled later if needed.



## Work with versions

Version control workflows prevent the chaos of everyone using their own development process with different and incompatible tools. Version control systems provide process enforcement and permissions so everyone stays on the same page.



## Code together

Version control synchronizes versions and makes sure that changes don't conflict with changes from others. The team relies on version control to help resolve and prevent conflicts, even when people make changes at the same time.



## Keep a history

Version control keeps a history of changes as the team saves new versions of code. Team members can review history to find out who, why, and when changes were made. History gives teams the confidence to experiment since it's easy to roll back to a previous good version at any time. History lets anyone base work from any version of code, such as to fix a bug in a previous release.



## Automate tasks

Version control automation features save time and generate consistent results. Automate testing, code analysis, and deployment when new versions are saved to version control are three examples.



GITHUB



**GitHub**

# Git Handson



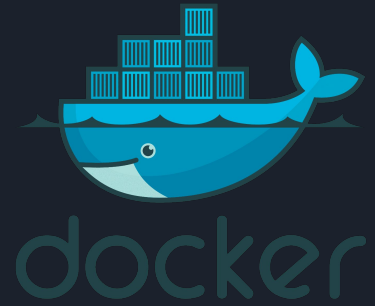


# ML Code Repo

[https://github.com/ashishpal2702/ML\\_template](https://github.com/ashishpal2702/ML_template)



# Docker



Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

With Docker, you can manage your infrastructure in the same ways you manage your applications.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.



# Docker

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

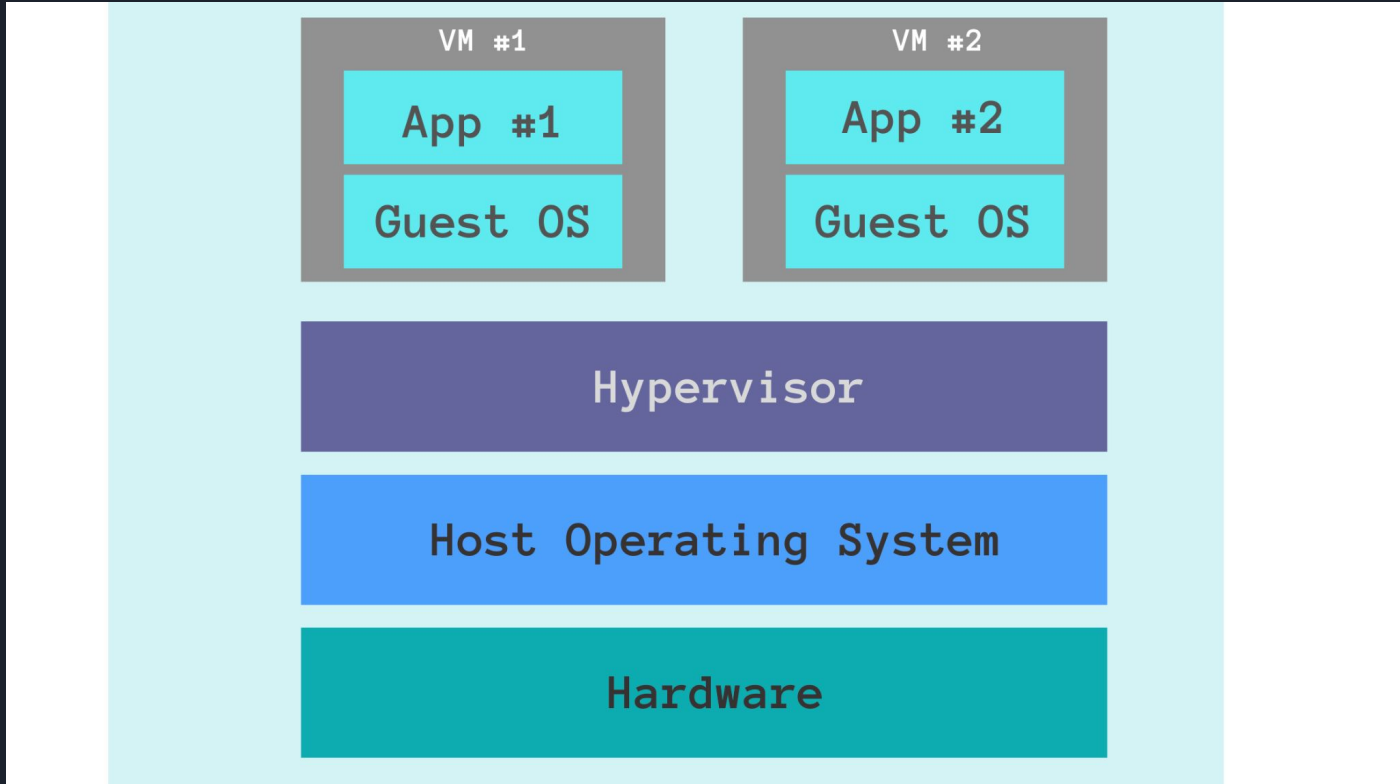




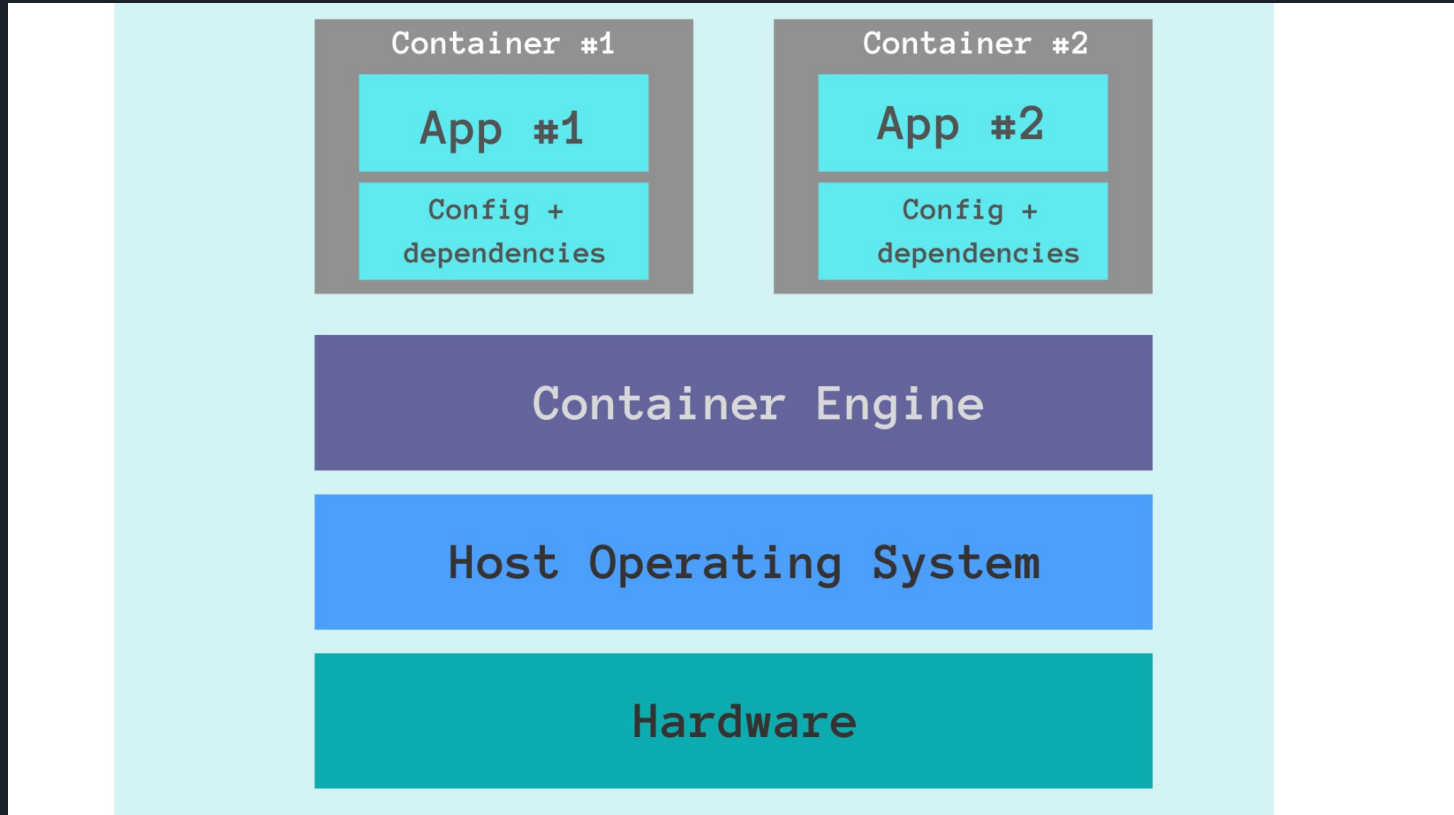
docker



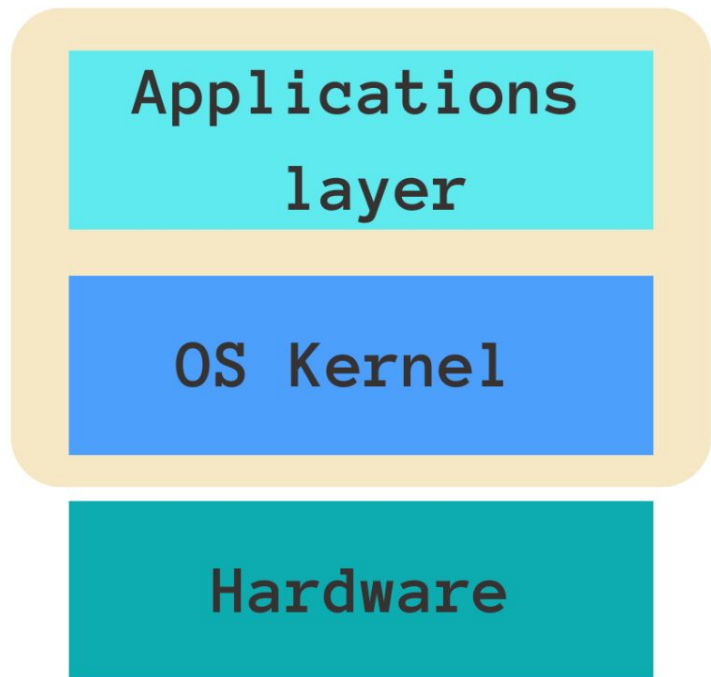
# How Does a Virtual Machine Work?



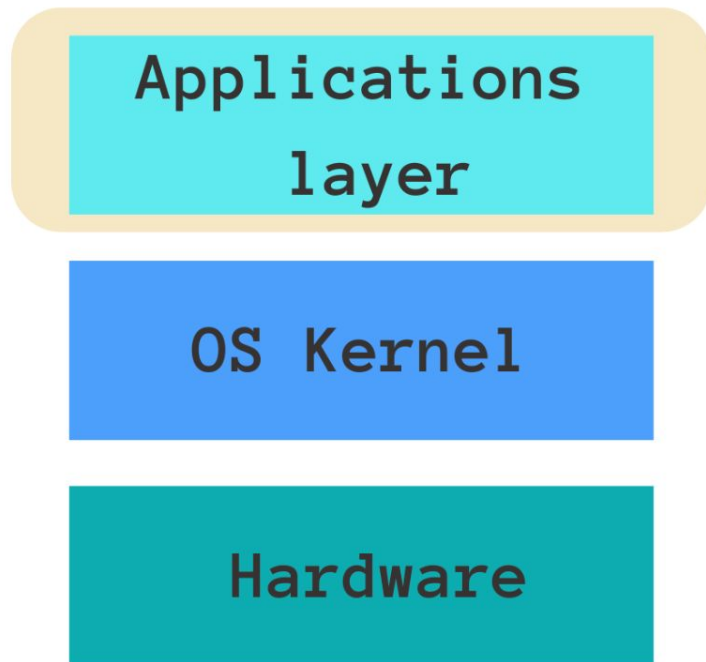
# How Does a Docker Container Work?



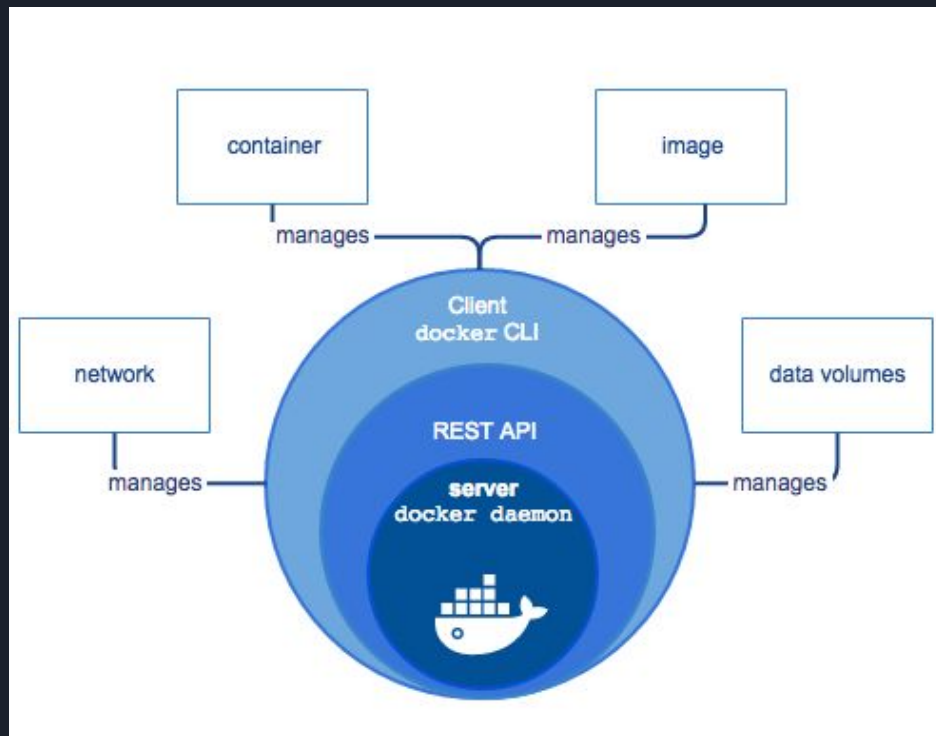
VM: Virtualization of the OS Kernel +  
Applications layer



Container: Virtualization of the  
Applications layer only



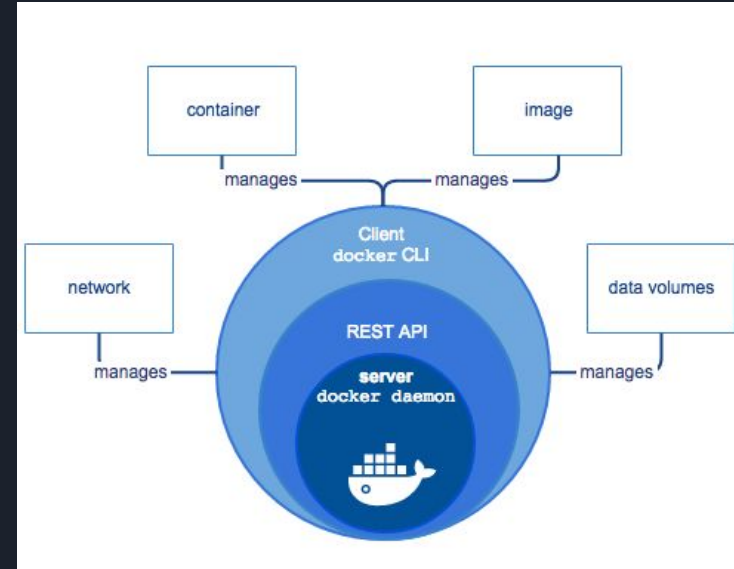
# DOCKER



# Docker Engine

Is an app that is comprised of

- a server (dockerd) that handles creating containers
- an API to talk to the server
- a CLI (docker) that uses the API
- The client and the daemon they interact via the API to handle creation and deployment of containers.





# Docker Architecture

- Docker registry: stores Docker images. E.g., Docker hub is a public registry.
- Docker objects:
  - Images
  - Networks
  - Containers
  - Volumes
  - Plugins
- Image:
  - Read only template of instructions to create a container.
  - It is often based on other images.
  - `Dockerfile` creates images: slightly changing it does not mean rebuilding from scratch (lightweight)
- When we run the following command:
  - `docker run -i -t ubuntu /bin/bash`

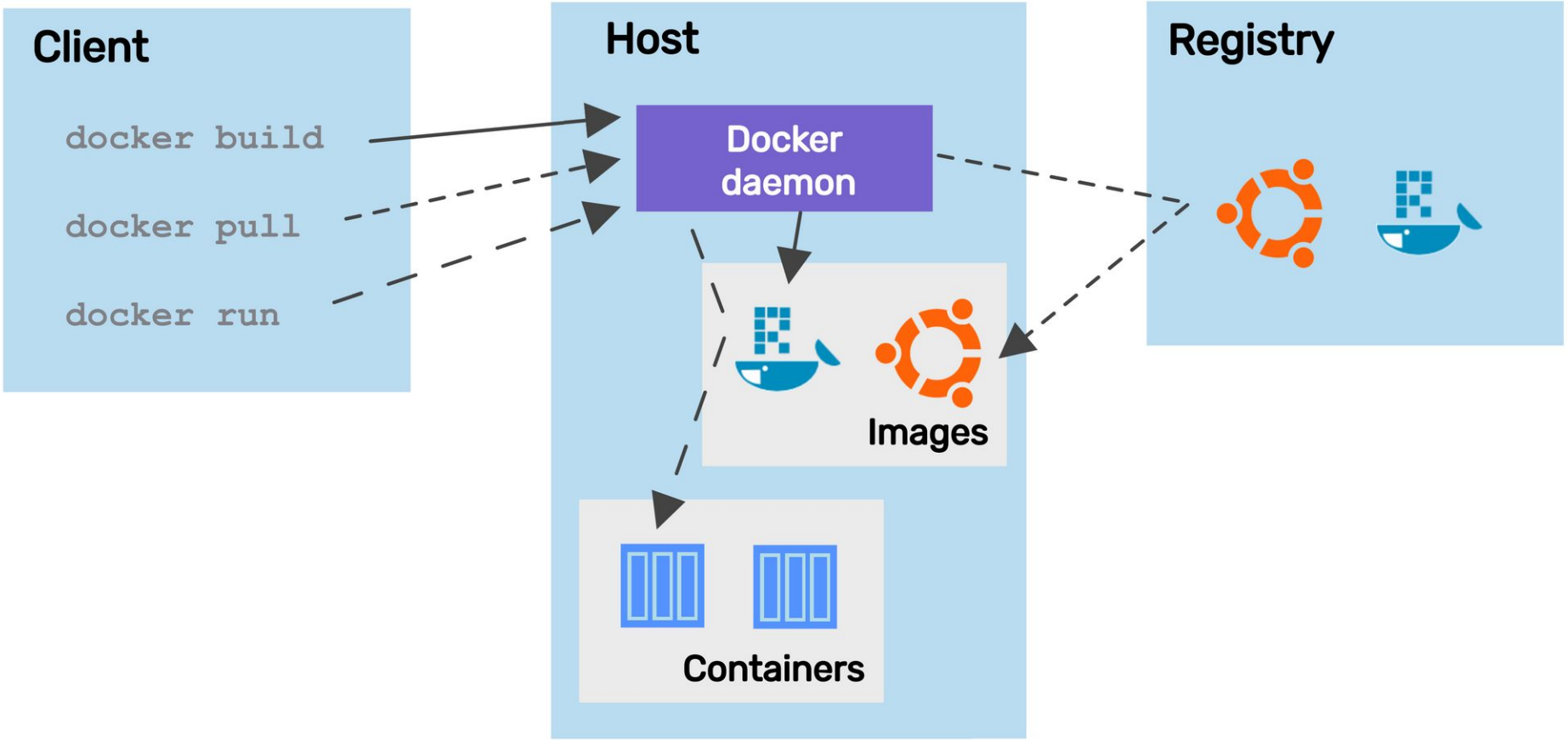


# Docker Architecture

- Docker pulls the image from the registry
- Creates a container
- Allocates a private filesystem
- Creates a network interface
- Executes `/bin/bash` because of the `-i -t` flag
- Services: Docker (> 1.12) has the capability to make multiple daemons work together (as a *swarm*) with load balancing automatically handled.
- Container:
  - A runnable instance of an image
  - Several operations: create, start, stop, move, delete.
  - When deleted, it does NOT store state!
  - In contrast, a VM is a guest operating system that accesses host resources through a *hypervisor*.



Docker Architecture



# Containerization of ML Models using Docker



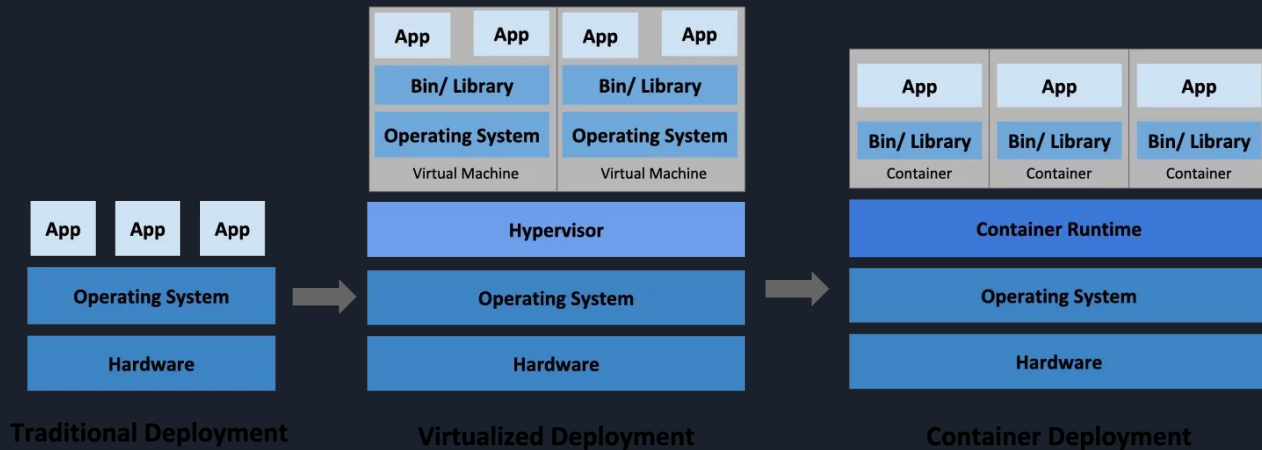



# Docker Commands

1. `docker build . / docker build -t mlops .`
2. `docker run -dp 127.0.0.1:8080:8080 mlops`
3. `docker ps`
4. `docker stop mlops`
5. `docker rm mlops`

# What Is Kubernetes?

Kubernetes is a container orchestration tool. So to understand what it is exactly, we need to understand what containers are and how they revolutionized the industry.



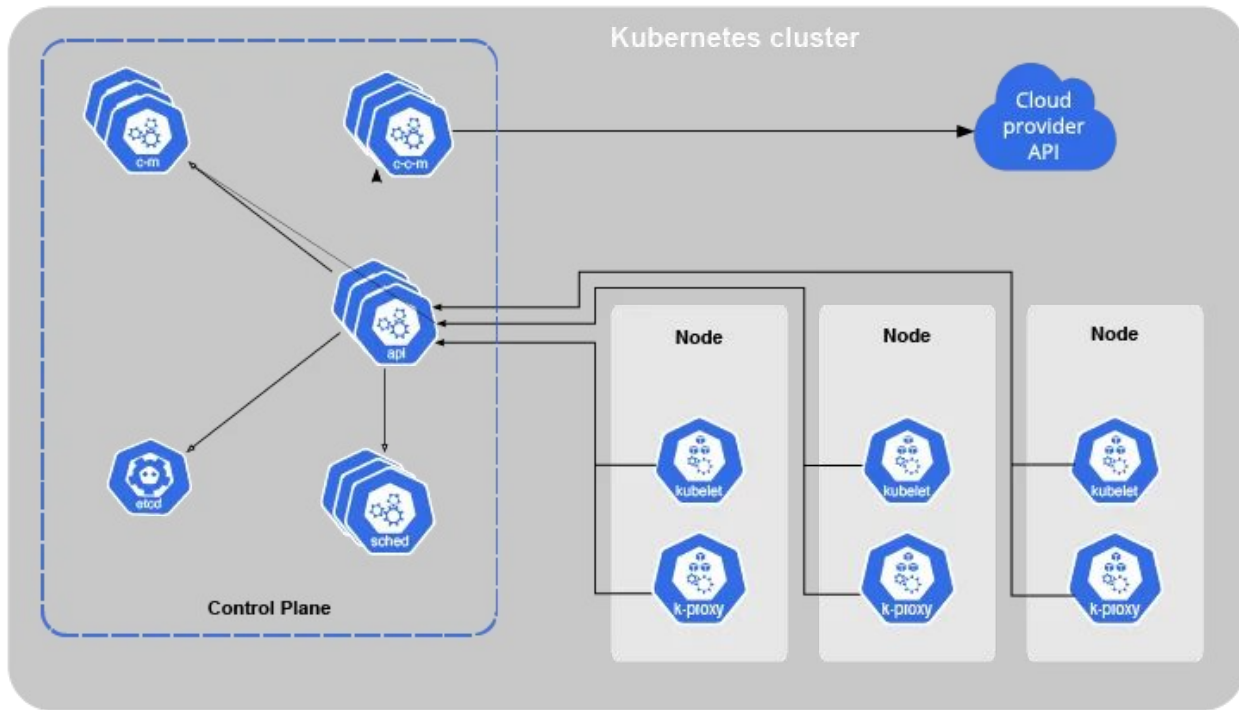


Originally, applications used to run on a dedicated server and were thus constrained by the hardware and OS available.

To speed up deployment and improve portability, virtualization was introduced, allowing for the abstraction of application code and environment in a virtual machine. Multiple virtual machines can run on the same hardware to reduce resource waste, with the hypervisor allocating processors, memory, and storage among them.

Still, virtualizing the physical hardware is a slow process. Containers successfully reduce deployment time to seconds by sharing the machine OS kernel so that each container hosts only the code, configurations, and packages that the application depends on.

Kubernetes lets you manage multiple running containers with zero downtime. It also features autoscaling, failovers, load balancing, and more—all of which would be near to impossible to achieve manually—plus well-defined deployment patterns and amazing community support.





# What Is Docker for Kubernetes?

The collection of installations, application code, and dependencies required to configure an application environment is defined in an “image.” Docker is the preferred tool to create images:

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```



# What Is Docker for Kubernetes?

Images are defined in a Dockerfile and organized in layers, where the lower in the hierarchy a layer is, the less it is expected to change over time. This is to optimize deployment efficiency, as layers are automatically cached after the first build. The example shown covers the typical steps for defining most images:

Import the parent image using the FROM clause; refer to DockerHub for a large list of ready-made parent images.

- Install some custom packages using the RUN clause.
- Copy the relevant code in the preferred working director using the COPY and WORKDIR clauses.
- Specify which command the image should run using the CMD clause.
- Define 3000 as the port the application listens to using the EXPOSE clause.

When the image is running, it is then called a “container” and managed by container orchestration tools such as Kubernetes.





# How to Run an Image on Kubernetes

To run an image on Kubernetes, we would typically have to define a series of YAML configuration files with a minimal setup containing the following files:

- **configmap.yaml** stores non-confidential input key<>value pairs; if confidential, use a secret.yaml instead.
- **deployment.yaml** starts n replicas with the specified container image, mounted volumes, environment variables, configmap/secret, hardware selection, and target port.
- **service.yaml** sends requests from the public node port to the deployment's target port.
- **autoscale.yaml** defines a scaling policy typically based on CPU or memory.



END