Kubernetes

Nived Varma

Microsoft Certified Trainer

Expert Azure Architect

Module-1 Introduction to containers and Kubernetes

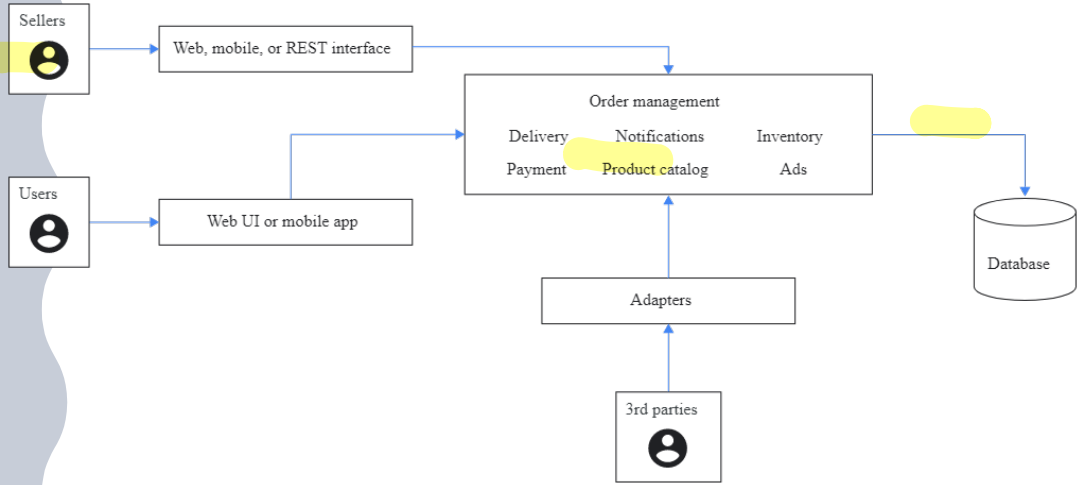
- Microservices
  - Advantages of running microservices
  - Disadvantages of running microservices
- Fundamentals of containers
  - Container images
- Kubernetes
  - Kubernetes as a container orchestration platform
  - Pods in Kubernetes
  - Deployments in Kubernetes
  - Services in Kubernetes
  - Azure Kubernetes Service

Microservices

What is Microservices?

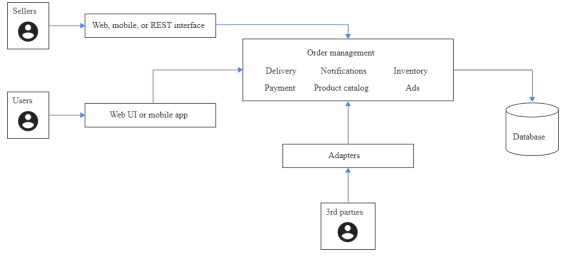
But Before that

What is monolithic applications ?

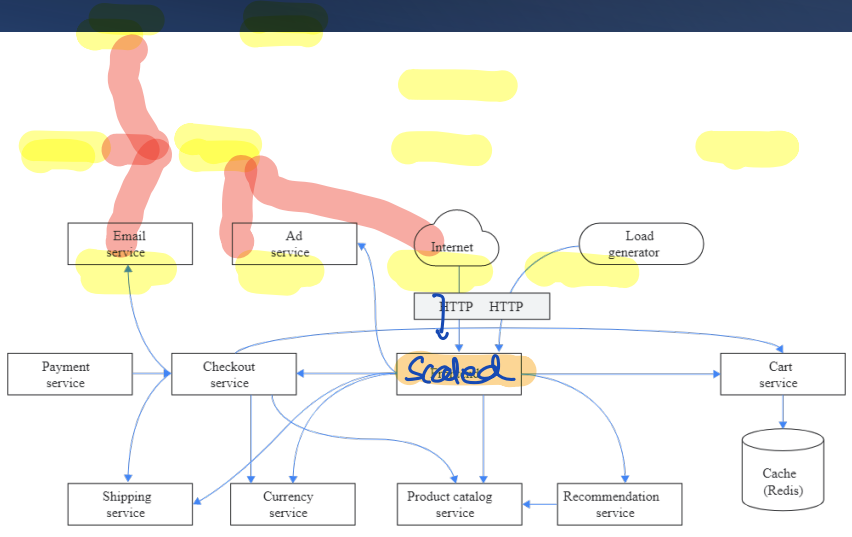


Monolithic – Benefits & Challenges

- Benefits
  - Possible to implement end-to-end testing (using selenium)
  - For deployment – copy the packaged application to a server
  - Provides performance advantage – modules can call each other
- Challenges
  - As application grows – it becomes complicated to implement changes in a large and complex solution.
  - Change in any code affects the whole system – the changes have to be thoroughly coordinated.
  - Complicated to achieve CI/CD with large monolith. The entire application needs to be redeployed.
  - Monolithic applications can be difficult to scale- when different modules have conflicting resource requirements
  - Difficult to adopt new frameworks and language – entire application need to be rewritten.



Microservices Based Applications



A dedicated microservice implements each functional area of the ecommerce application.

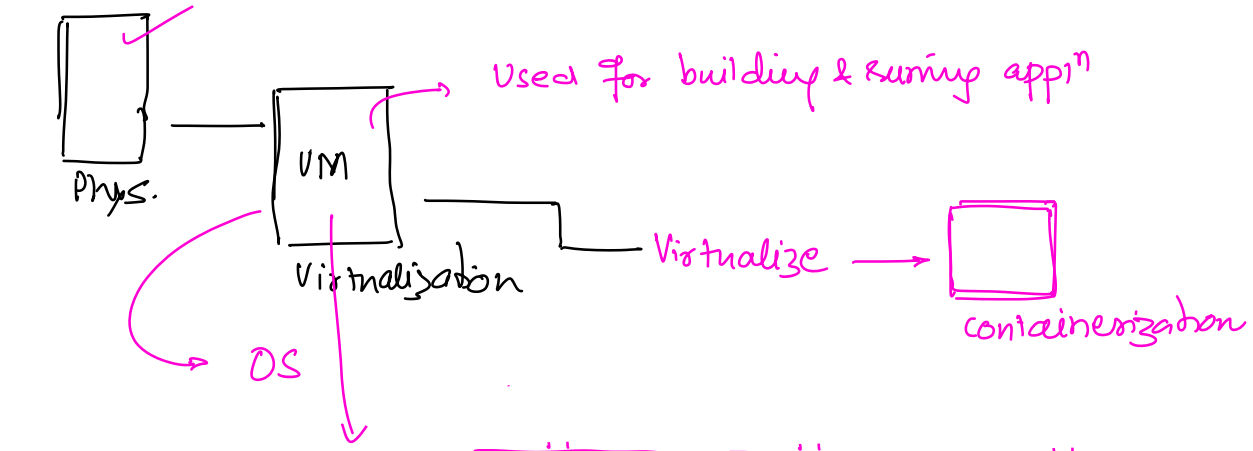
Each backend service might expose an API, and services consume APIs provided by other services.

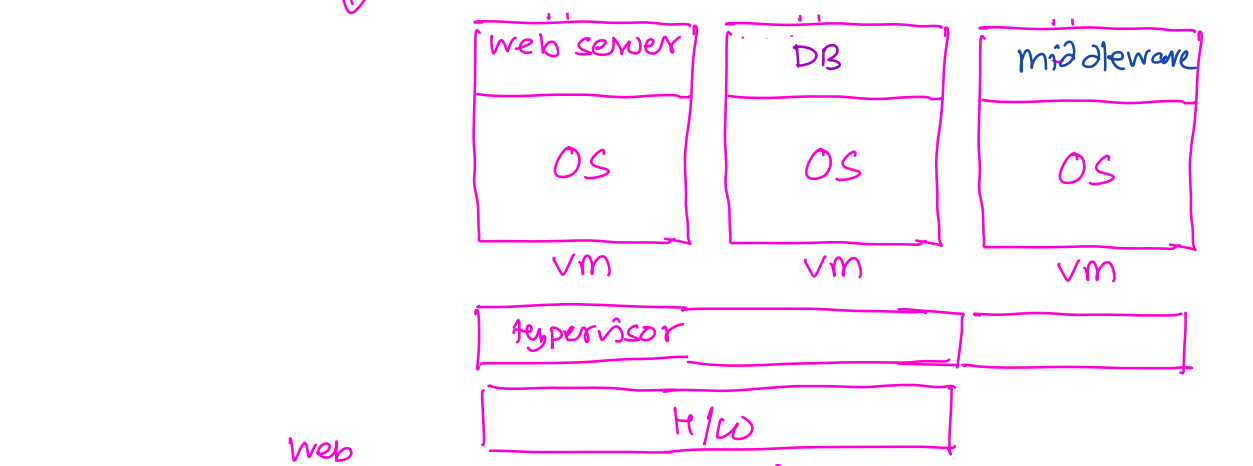
For example, to render web pages, the UI services invoke the checkout service and other services.

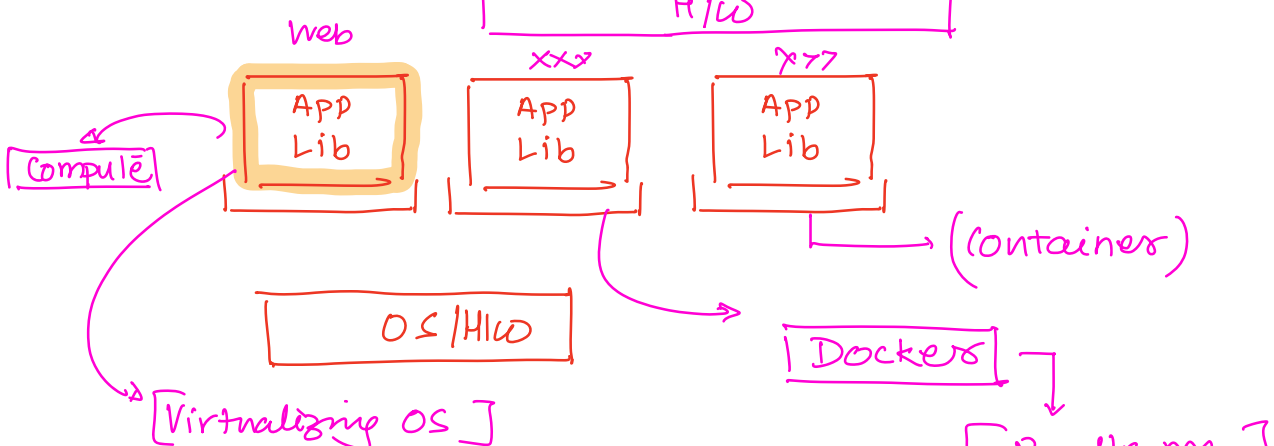
Services might also use asynchronous, message-based communication.

Containers

Containers







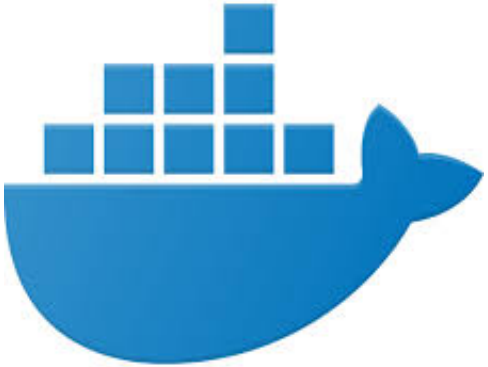

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
        return "Hello"
@app.route("/version")
def version():
```

Docker Fundamentals

Nived Varma

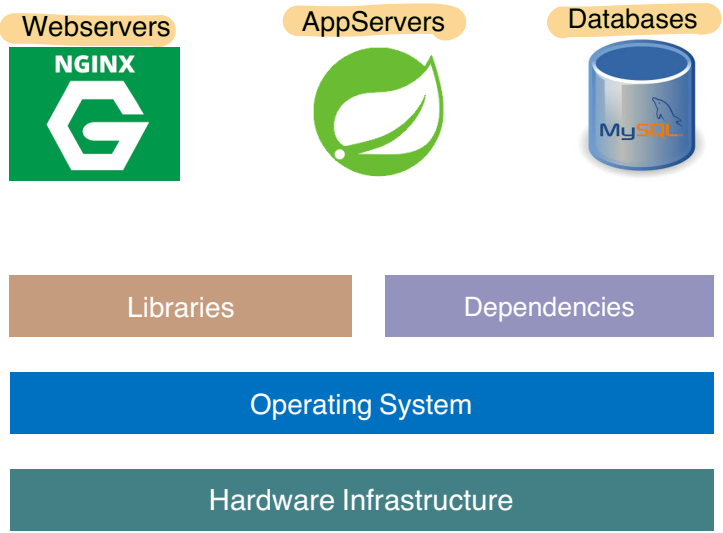
Docker

Introduction



What problems we have with Traditional Infra?

- Traditional Approach
- Installation & Configuration
  - Time consuming
  - Need to perform install/configs on every server and every environment (dev, qa, staging, production)
- Compatibility & Dependency
  - Need to keep resolving issues related to libraries and dependencies
- Inconsistencies across Environments
  - Very hard to track changes across Dev/QA/Staging and Prod environments and they end up with inconsistencies
- Operational Support
  - Need more resources to handle operational issues on day to day basis
    - Server Support (hardware, software)
    - Patching releases
- Developer Environments
  - When a new developer joins the team, time it takes to provision his development environment in traditional approach is time taking.



Physical Machines

Webservers

AppServers

Databases

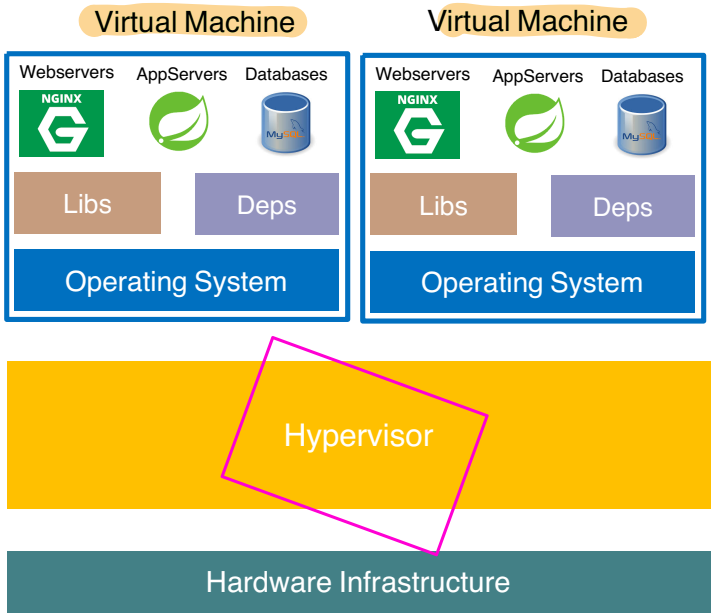
Libraries

Dependencies

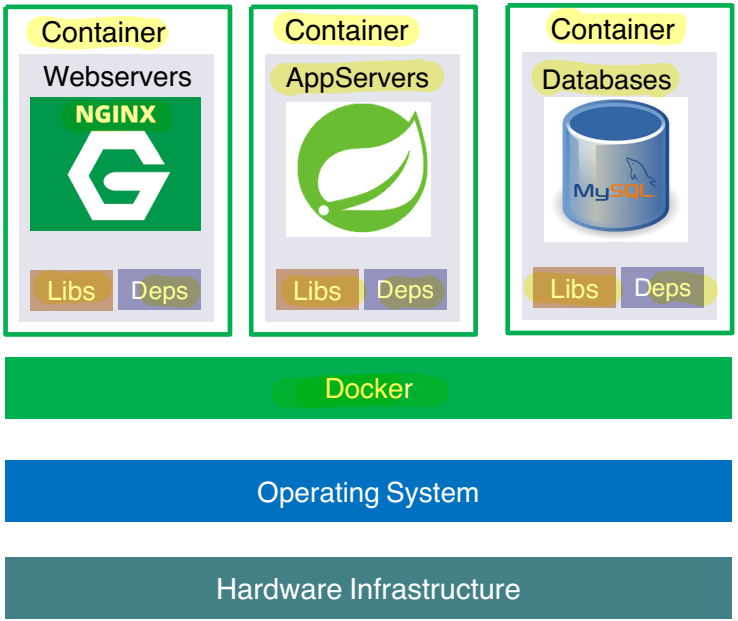
Operating System

Hardware Infrastructure

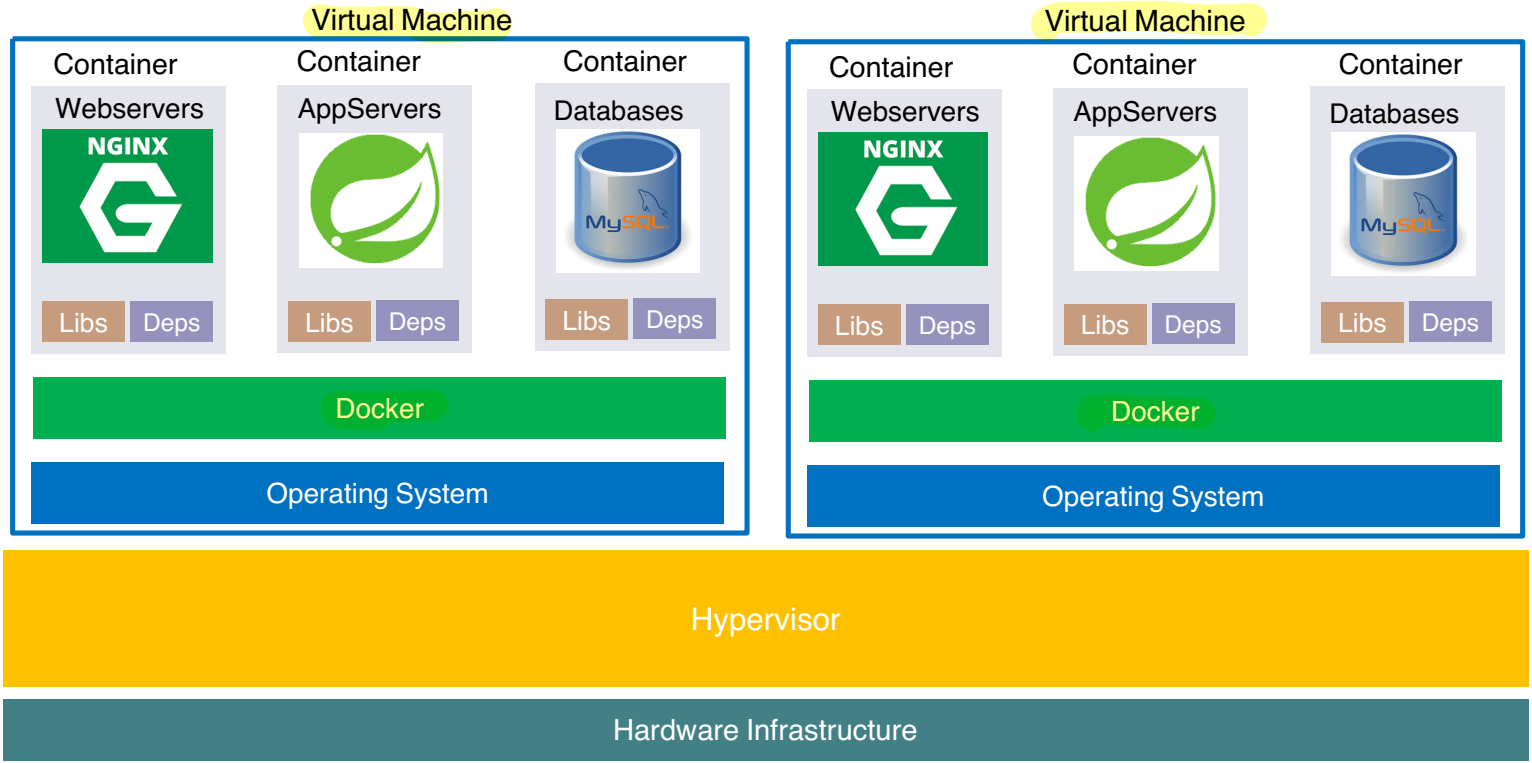
Virtual Machines



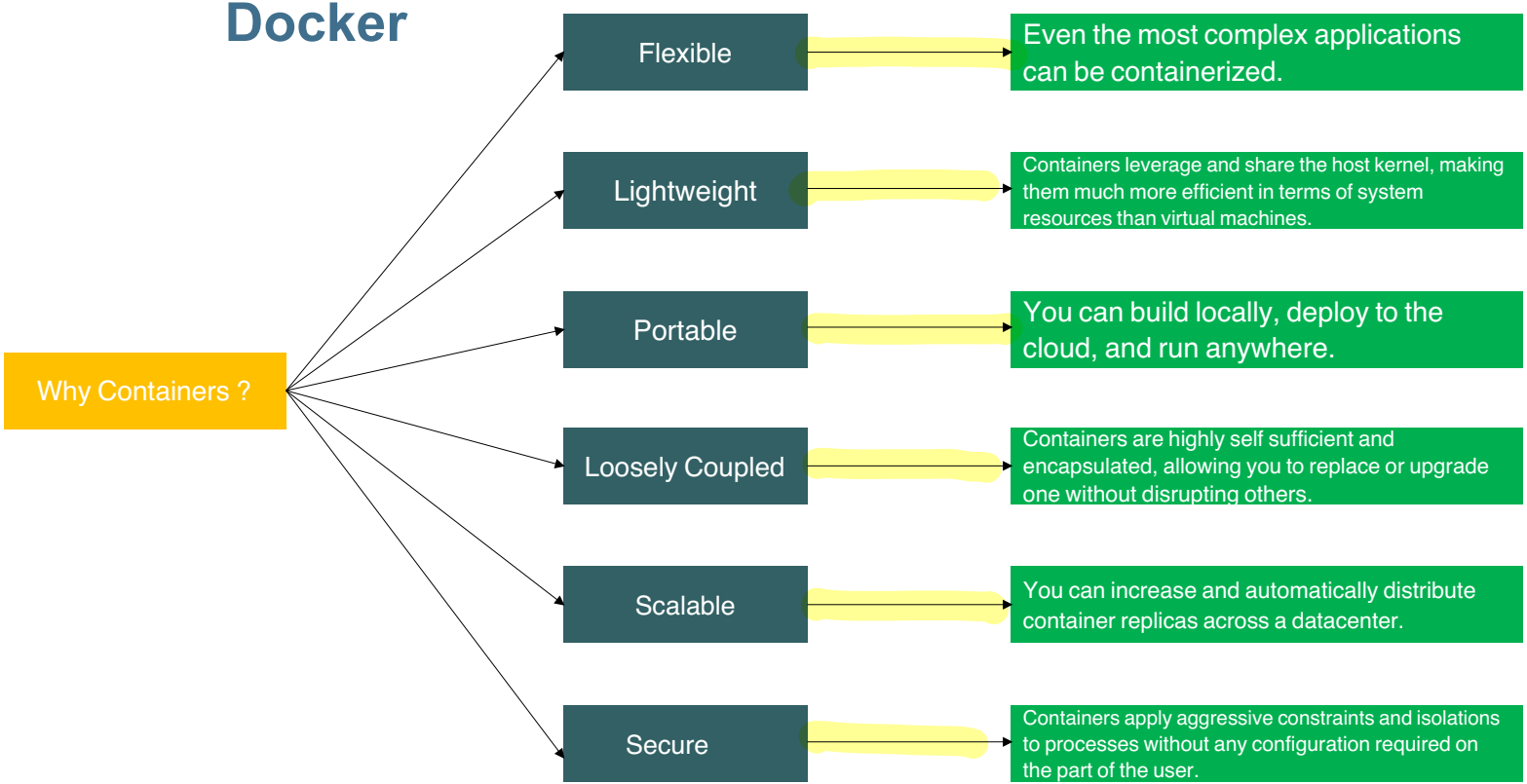
Physical Machines with Docker



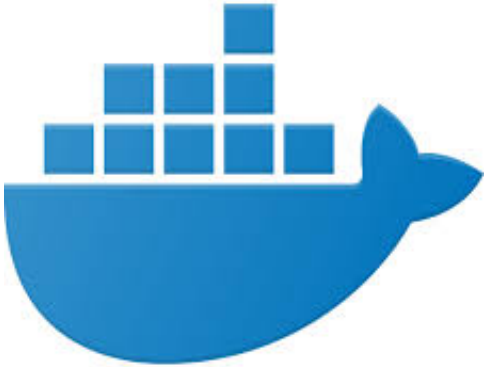
Virtual Machines with Docker



Advantages of using Docker

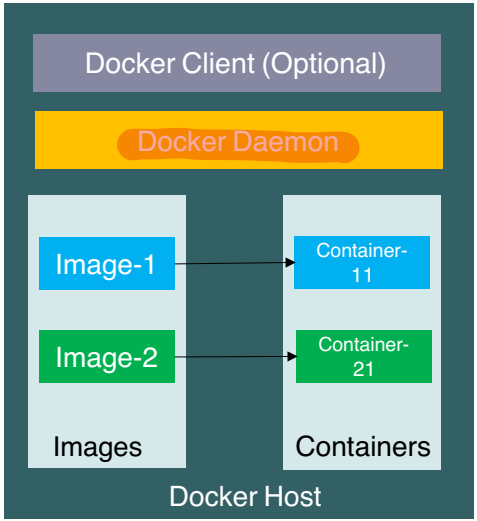


Docker
Architecture

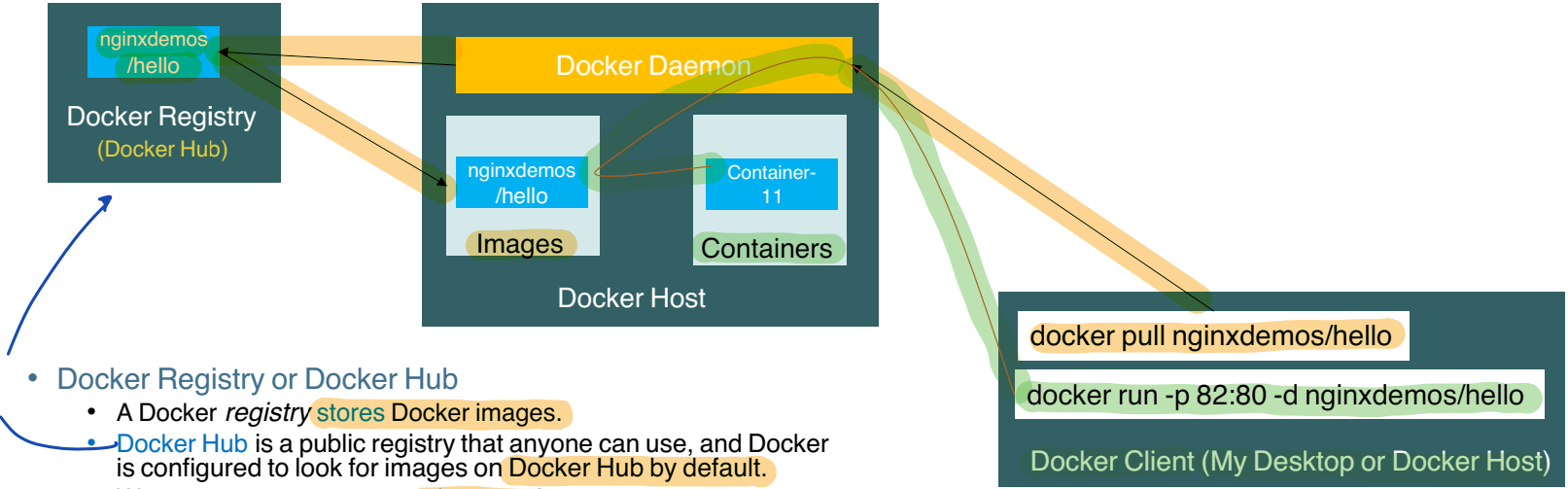


Docker - Terminology

- Docker Daemon
  - The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- Docker Client
  - Docker client can be present on either Docker Host or any other machine.
  - The Docker client (docker) is the primary way that many Docker users interact with Docker.
  - When you use commands such as docker run, the client sends these commands to dockerd (Docker Daemon), which carries them out.
  - The docker command uses the Docker API.
  - The Docker client can communicate with more than one daemon.
- Docker Images
  - An *image* is a read-only template with instructions for creating a Docker container.
  - Often, an image is *based on* another image, with some additional customization.
  - For example, we may build an image which is based on the ubuntu image, but installs the Apache web server and our application, as well as the configuration details needed to make our application run.
- Docker Containers
  - A container is a runnable instance of an image.
  - We can create, start, stop, move, or delete a container using the Docker API or CLI.
  - We can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
  - When a container is removed, any changes to its state that are not stored in persistent storage disappear.



Docker - Terminology



- Docker Registry or Docker Hub
  - A Docker *registry* stores Docker images.
  - Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  - We can even run our own private registry.
  - When we use the docker pull or docker run commands, the required images are pulled from our configured registry.
  - When we use the docker push command, our image is pushed to our configured registry.

Create LAB VM
Docker Installation
Create and run containers.

Demo

– Docker fundamentals

[Workflow]

Scenario-1

Pull image from docker hub & Run locally.

- ✓ docker pull <repo on dock.hub>/<image name>:<v>
- ✓ docker run .....

Scenario-2

pull the image – whalesay
update ⟵ (via Dockerfile) ⟶ installed fortunes
run locally
push image to docker hub (own repo)

Scenario-3

pull the image
update
run locally
push the image to ACR (cloud hosted private repo)
↳ tag
↳ az acr login
↳ docker push
↳ update the tag

```
docker ps
docker ps -a
docker ps -a -q
```

connect to the container

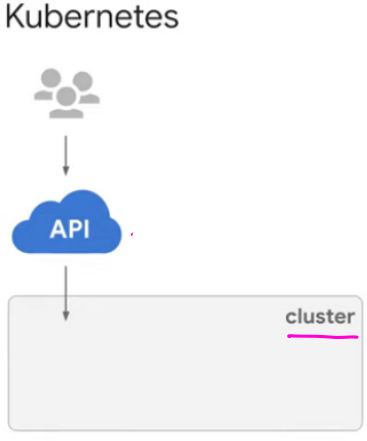
```
docker exec -it <container name> /bin/sh
#
```

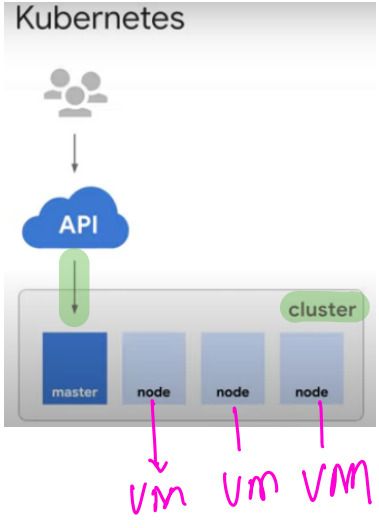
Kubernetes ⟶ means - pilot
helmsman

Kubernetes Architecture

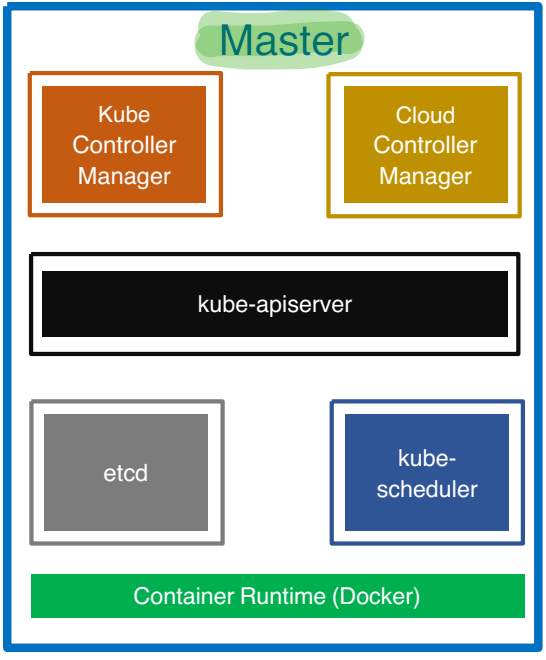
What is Kubernetes?

- Kubernetes is an open -source orchestrator for containers so you can better manage and scale your applications.
- Kubernetes offers an api that lets people that is authorized people not just anybody control

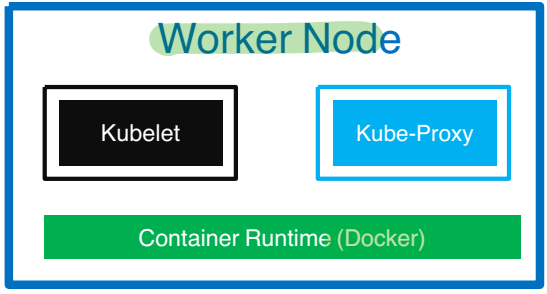


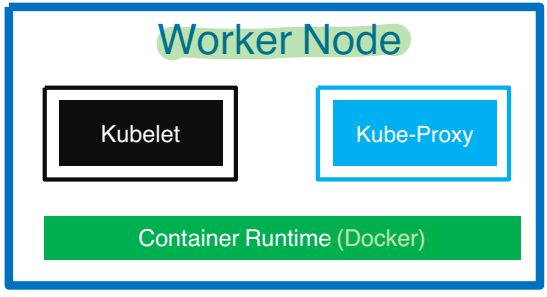


Kubernetes - Architecture

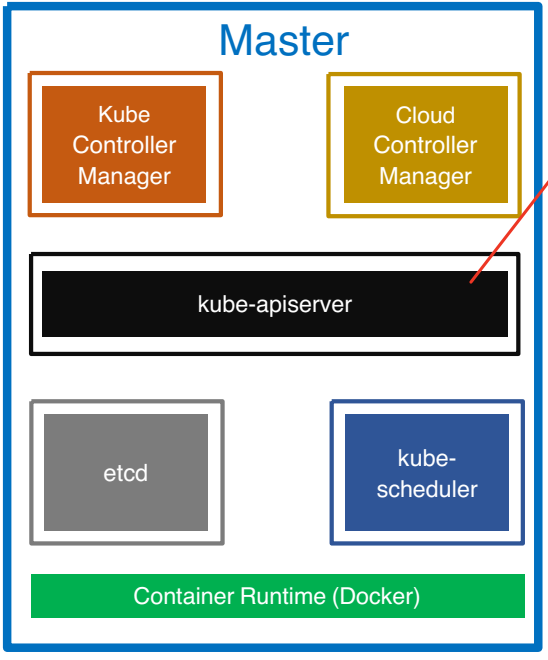


control Plane



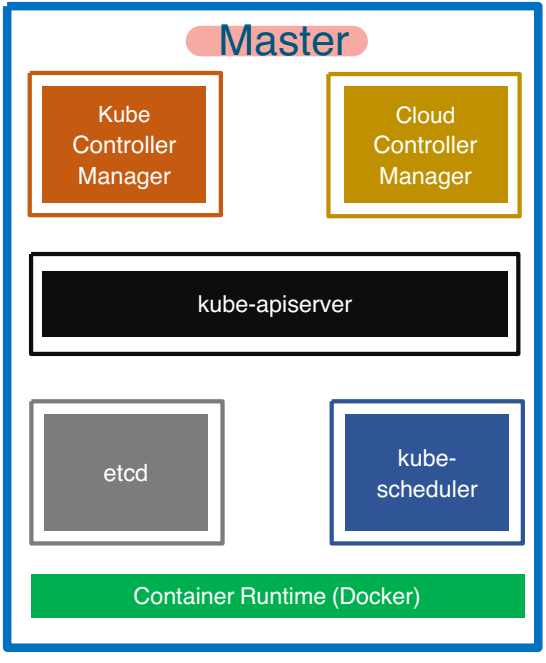


Kubernetes Architecture - Master



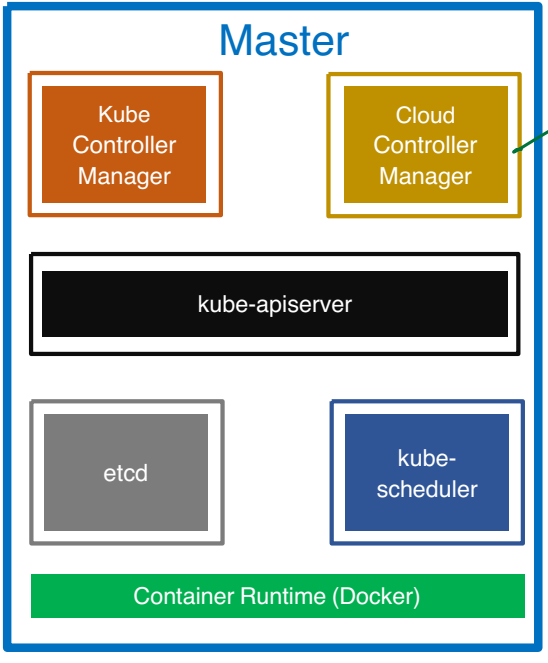
- kube-apiserver
  - It acts as front end for the Kubernetes control plane. It exposes the Kubernetes API
  - Command line tools (like kubectl), Users and even Master components (scheduler, controller manager, etcd) and Worker node components like (Kubelet) everything talk with API Server.
- etcd
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
  - It stores all the masters and worker node information.
- kube-scheduler
  - Scheduler is responsible for distributing containers across multiple nodes.
  - It watches for newly created Pods with no assigned node, and selects a node for them to run on.

Kubernetes Architecture - Master



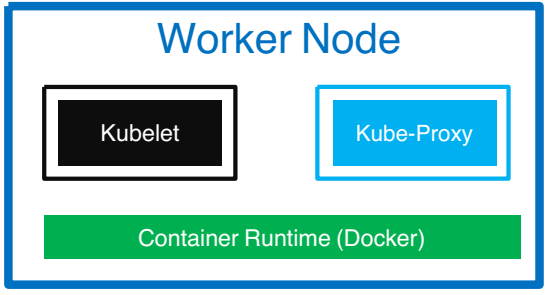
- kube-controller-manager
  - Controllers are responsible for noticing and responding when nodes, containers or endpoints go down. They make decisions to bring up new containers in such cases.
  - Node Controller: Responsible for noticing and responding when nodes go down.
  - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods)
  - Service Account & Token Controller: Creates default accounts and API Access for new namespaces.

Kubernetes Architecture - Master



- cloud-controller-manager
  - A Kubernetes control plane component that embeds cloud-specific control logic.
  - It only runs controllers that are specific to your cloud provider.
  - On-Premise Kubernetes clusters will not have this component.
  - Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
  - Route controller: For setting up routes in the underlying cloud infrastructure
  - Service controller: For creating, updating and deleting cloud provider load balancer

Kubernetes Architecture – Worker Nodes



- Container Runtime
  - Container Runtime is the underlying software where we run all these Kubernetes components.
  - We are using Docker, but we have other runtime options like rkt, container-d etc.

- Kubelet
  - Kubelet is the agent that runs on every node in the cluster
  - This agent is responsible for making sure that containers are running in a Pod on a node.
- Kube-Proxy
  - It is a network proxy that runs on each node in your cluster.
  - It maintains network rules on nodes
  - In short, these network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Kubernetes - Architecture