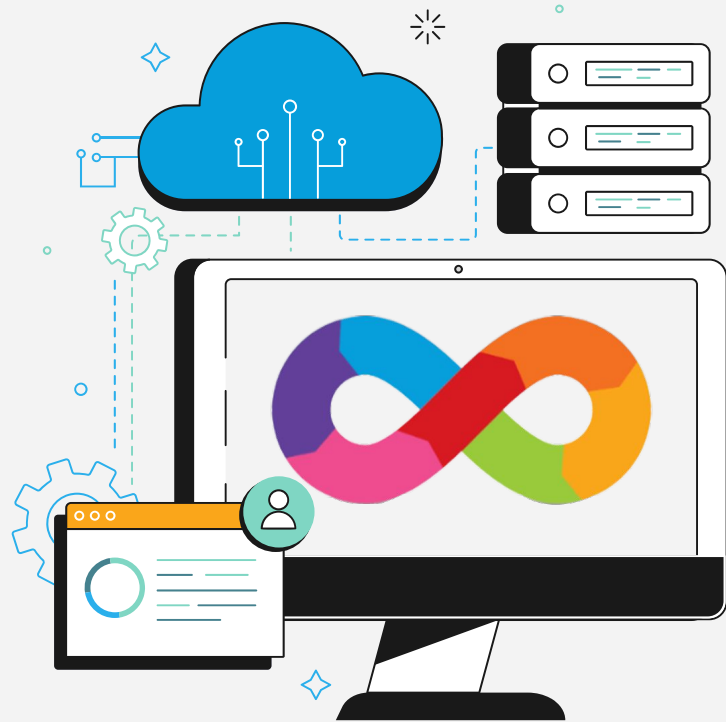


Introduction to DevOps

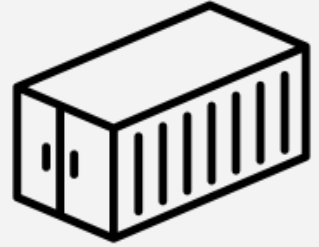
@ IBA - SMCS

Week 02



Obaid ur Rehman
Software Architect / Engineering Manager @ Folio3

Agenda for this week

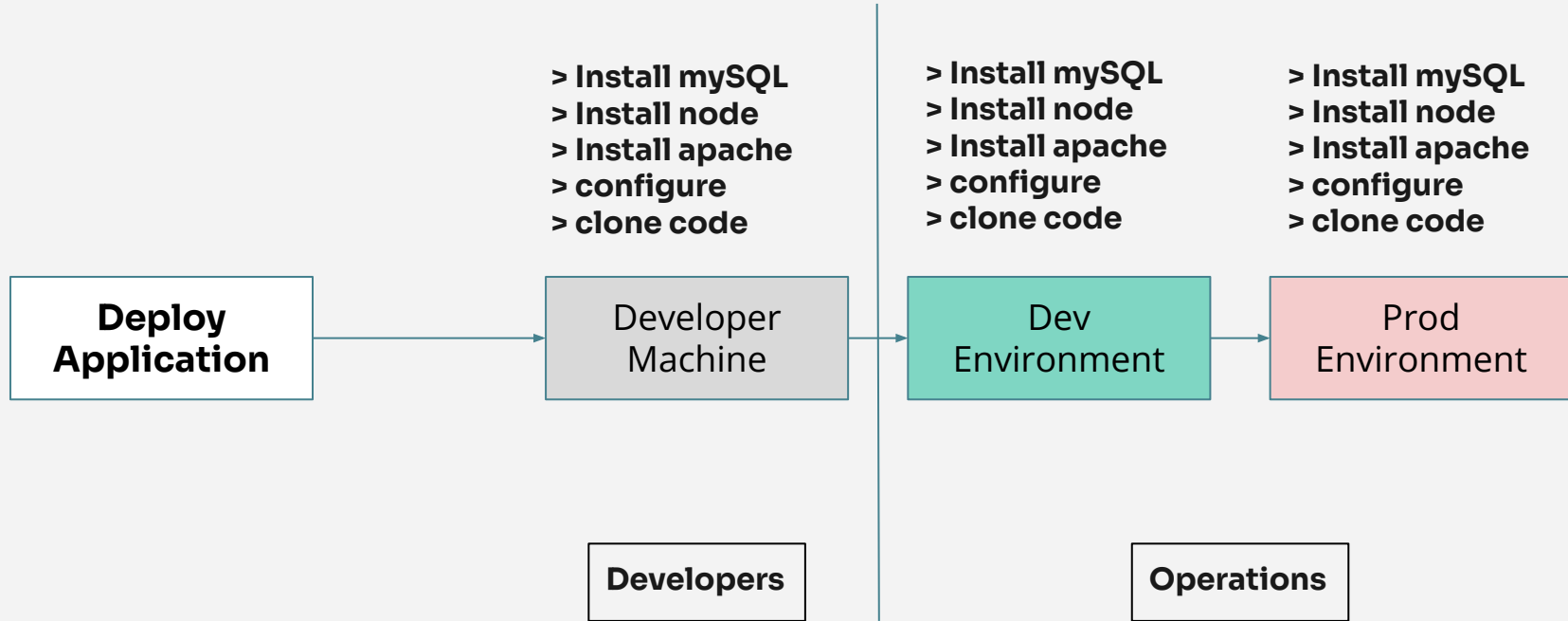


1. Before Containerization
2. What is Containerization
3. Virtual Machines V/s Containers
4. Docker Architecture
5. Container and Image Architecture
6. Dockerfile, Docker compose, Volumes, Networks, Registry, etc.
7. Building docker images, Multi-Stage Builds.
8. Running containers on Cloud: General overview.
9. Container Design Pattern: SideCar Pattern
10. Docker Alternatives
11. Hands-on & Assignment

Containerization in DevOps Land.

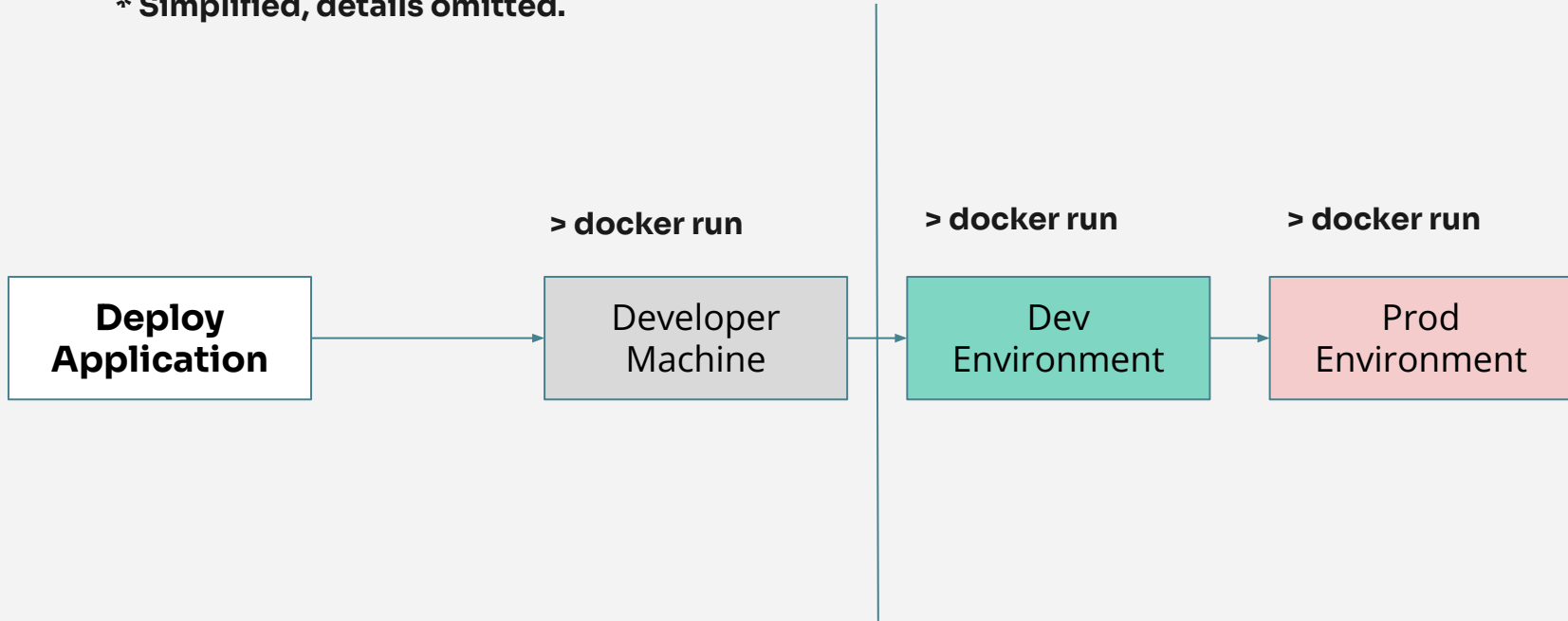


DevOps before containers



DevOps after Containers

* Simplified, details omitted.



Where does this fit in DevOps?

Continuous Integration and Continuous Deployment (CI/CD):

Containers facilitate a smooth integration into CI/CD pipelines. CI/CD processes benefit from the speed and consistency that containers provide during build, test, and deployment phases. Containers make it easier to automate the packaging and deployment of applications, enabling faster and more reliable delivery of software.

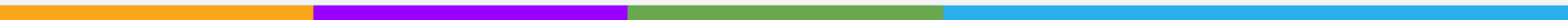


Where does this fit in DevOps?

Scalability and Resource Efficiency

Containers are lightweight and share the host OS kernel, which makes them more resource-efficient compared to traditional virtual machines.

DevOps teams can scale applications up or down more easily by running multiple instances of containers, allowing for efficient resource utilization.



Where does this fit in DevOps?

Isolation and Security


Containers provide a level of isolation between applications and their dependencies, enhancing security by preventing conflicts and minimizing the impact of one application on another.

DevOps teams can manage security policies for containers, making it easier to control and monitor access to resources.

Collaboration and Portability

Containers encapsulate the application and its dependencies, making it easier for developers, testers, and operations teams to collaborate seamlessly.

Containers are portable, enabling applications to run consistently across various environments, whether on-premises or in the cloud.



What is containerization?

Containerization is bundling of all the required pieces of a specific application into a single unit.

This means containers include all the

- binaries,
- Libraries or packages
- configurations etc.

However, containers do **NOT** include virtualized hardware or kernel resources.



What is containerization?

For example (Simplified):



You containerize it 📦



How do containers run?

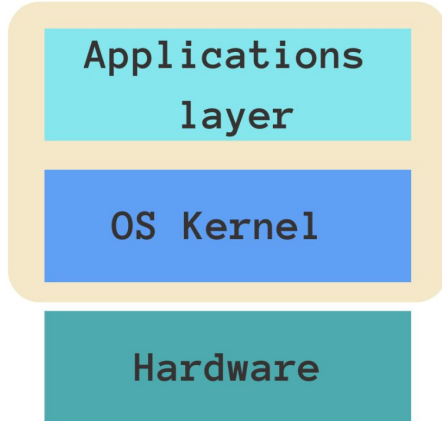
Containers run “**on top**” of a container runtime platform that abstracts the resources.

You can run it in a Virtual Machine too right? 🤔

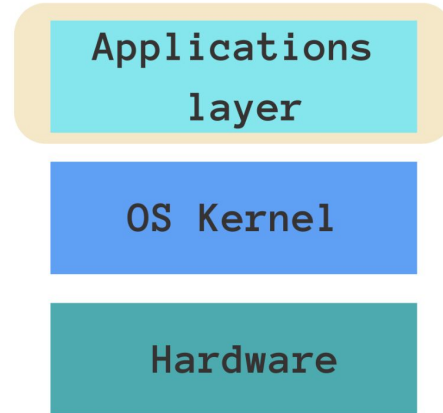


Container Runtime V/s VM

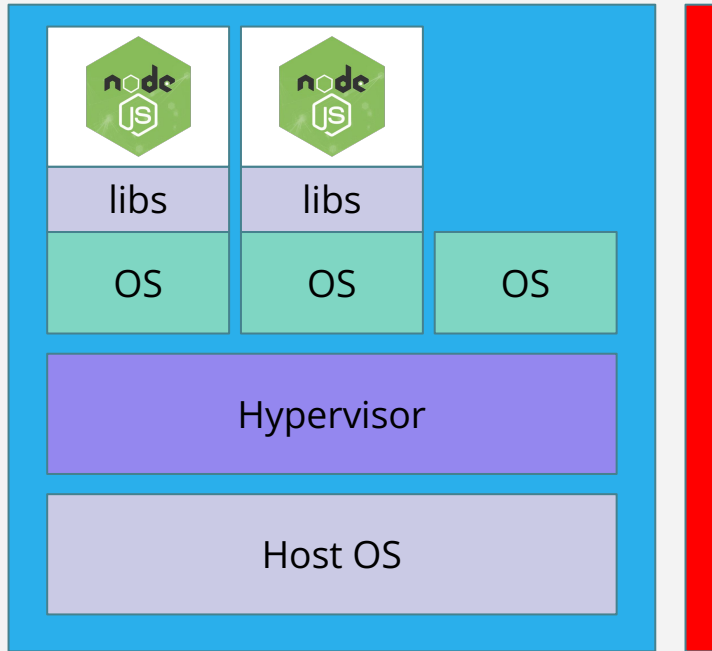
VM: Virtualization of the OS Kernel + Applications layer



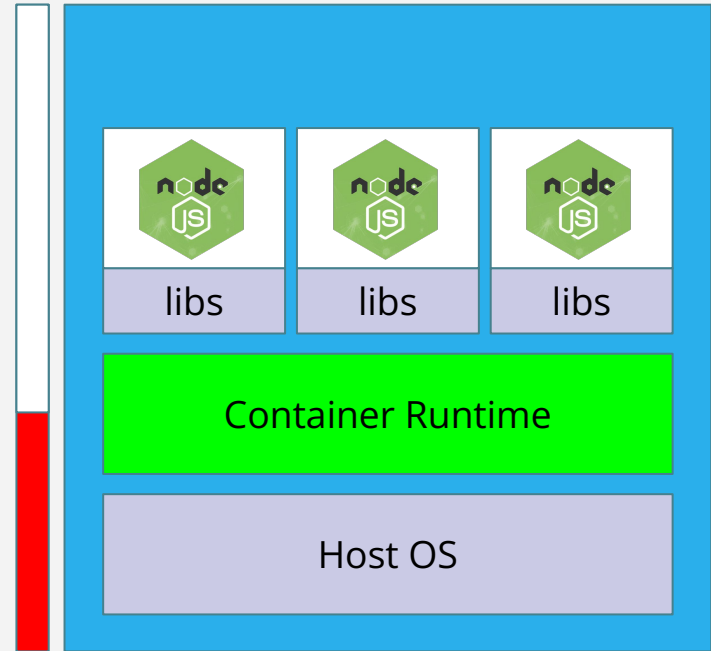
Container: Virtualization of the Applications layer only



Deploying on VM v/s using Containers



Virtual Machine



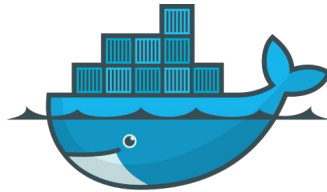
Container

Container runtimes



podman

containerd



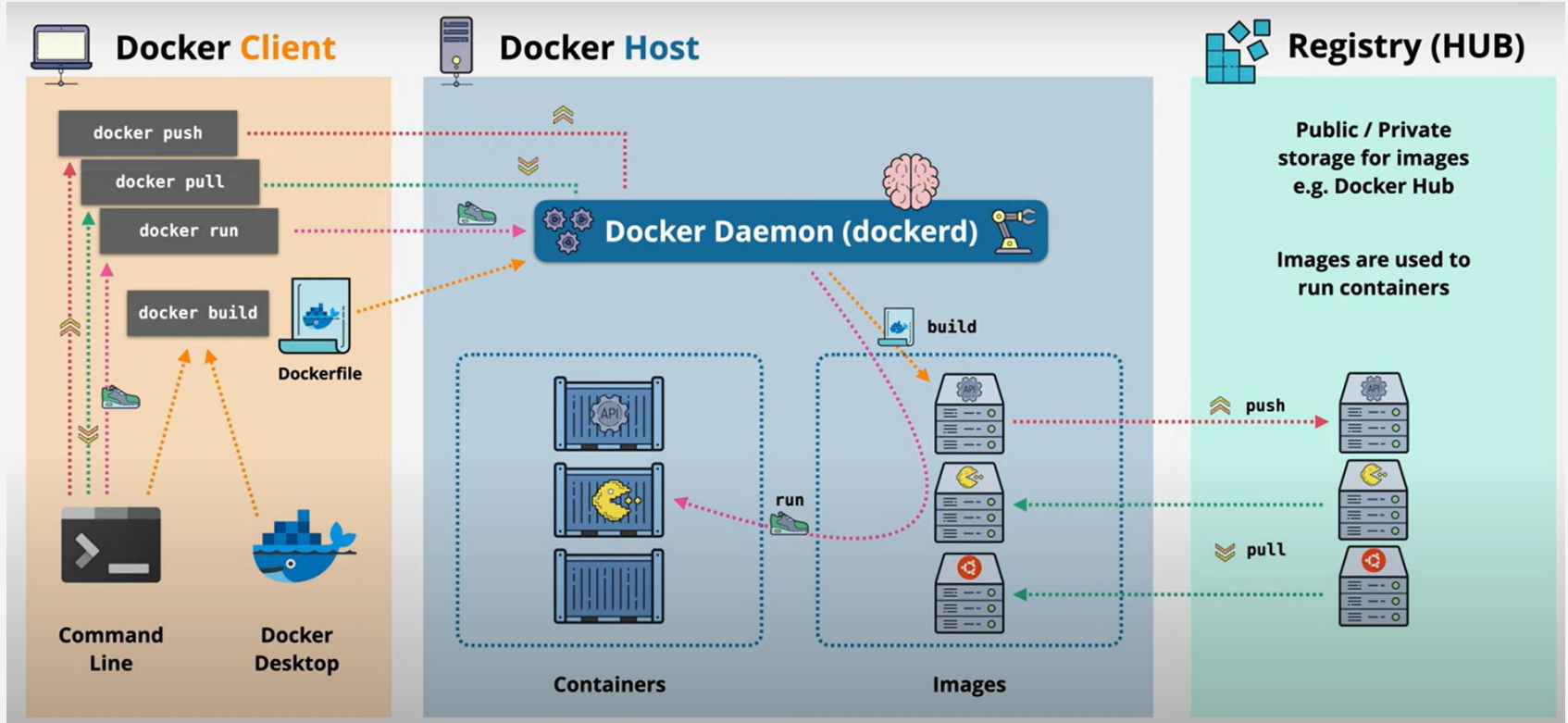
docker

Docker Architecture

Question: How does a Linux container runs on Windows? Or a Linux container runs on Mac?



Docker Architecture



Docker file.

A manifest for creating Docker image.

Image

A blueprint for creating Docker Container.

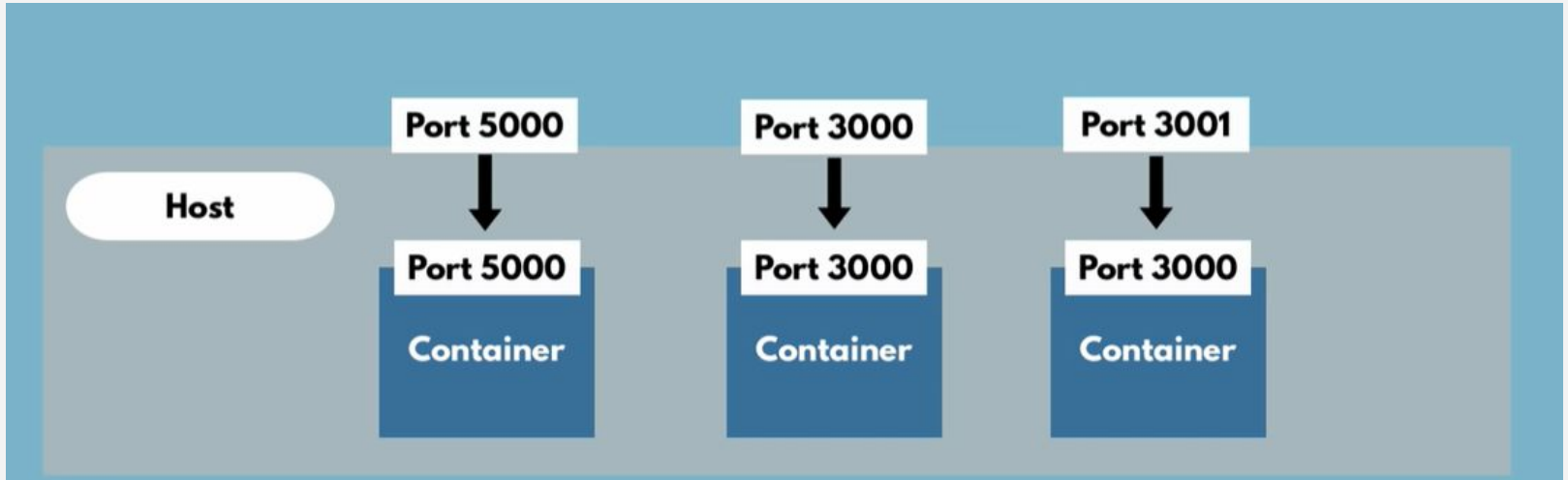
Container

A running unit of an Image.

Docker 101

1. Docker ps
2. Docker pull
3. Docker start
4. Docker stop
5. Docker push

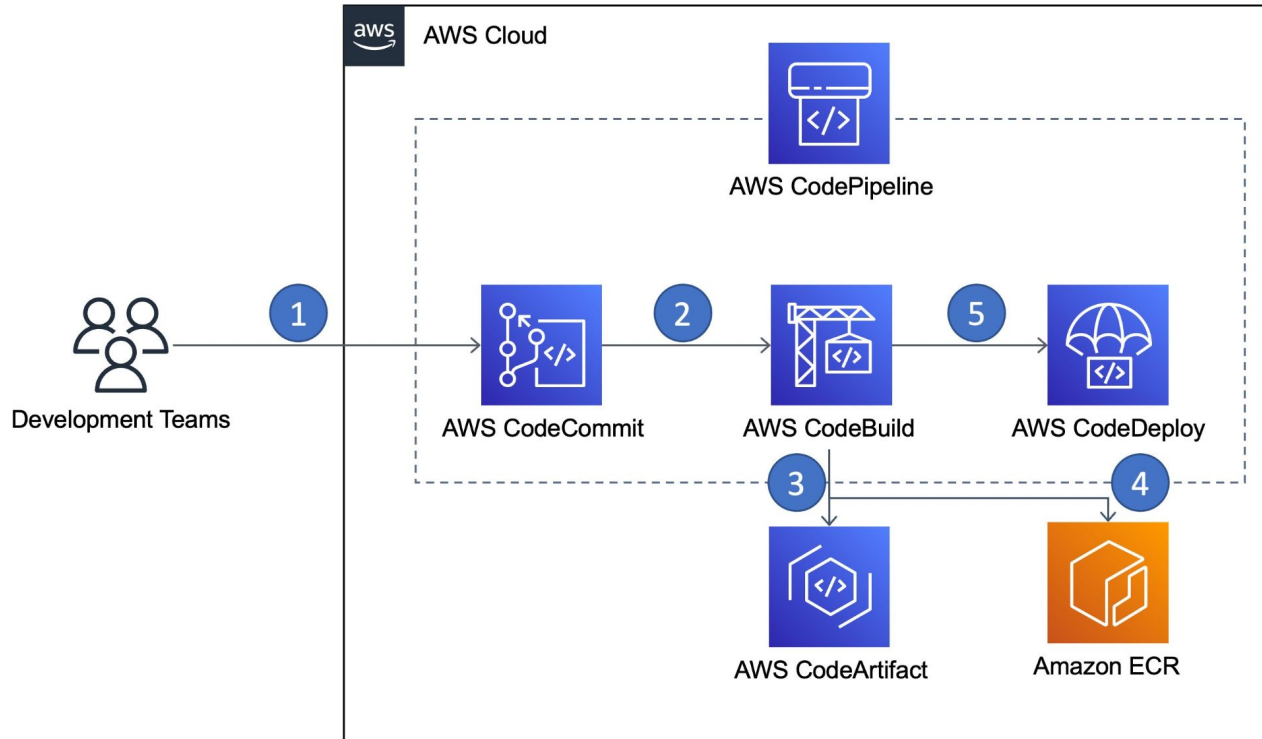
Container Port Bindings



Docker Hub & Container Registries

- Docker Hub
- Azure: Azure Container Repository
- Amazon: Elastic Container Repository

A simple containerized CI/CD on **AWS**



Creating a dockerfile

https://hub.docker.com/_/httpd



Pushing a Docker image to Repo.



Recap

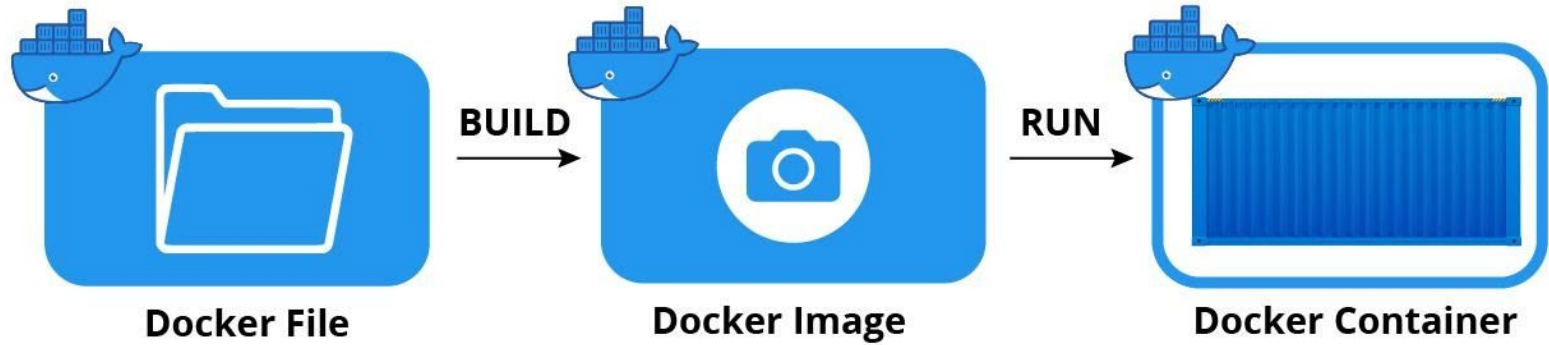
- ★ Containerization
- ★ Containers V/S Virtual Machines
- ★ Docker
- ★ Dockerfile, Images, Containers
- ★ Container Registries
- ★ Docker commands: build, run, push

Class 02 of Containers

- Docker-compose
- Some *docker-networking*, volume
- Container Design pattern: Sidecar pattern
- **Multi-stage builds**
- Running containers on scale and on the cloud



Docker Recap

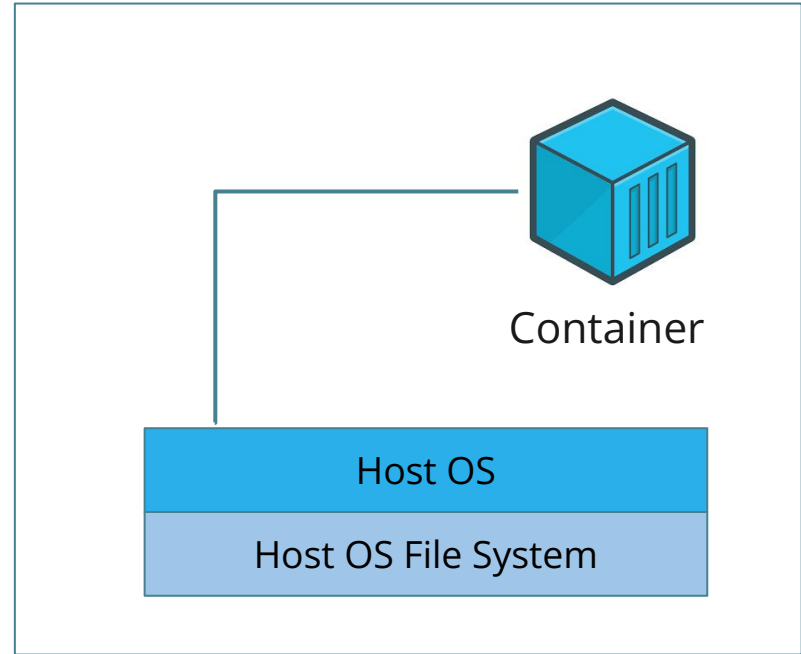


Container file system

- How is data persisted in container
- How to persist data across container lifecycle?
- Demo.

Container Volumes

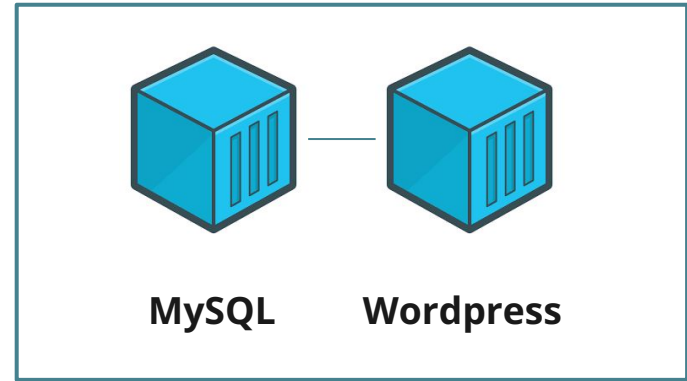
Volumes provide the ability to connect specific filesystem paths of the container back to the host machine. If you mount a directory in the container, changes in that directory are also seen on the host machine. If you mount that same directory across container restarts, you'd see the same files.



More:: <https://docs.docker.com/storage/volumes/>

Docker compose

- A tool that lets you define run multiple containers that communicate with each other.



Layers



Layers

Cache?

```
FROM golang:1.20-alpine
```



```
WORKDIR /src
```



```
COPY . .
```



```
RUN go mod download
```



```
RUN go build -o /bin/client ./cmd/client
```



```
RUN go build -o /bin/server ./cmd/server
```



```
ENTRYPOINT [ "/bin/server" ]
```

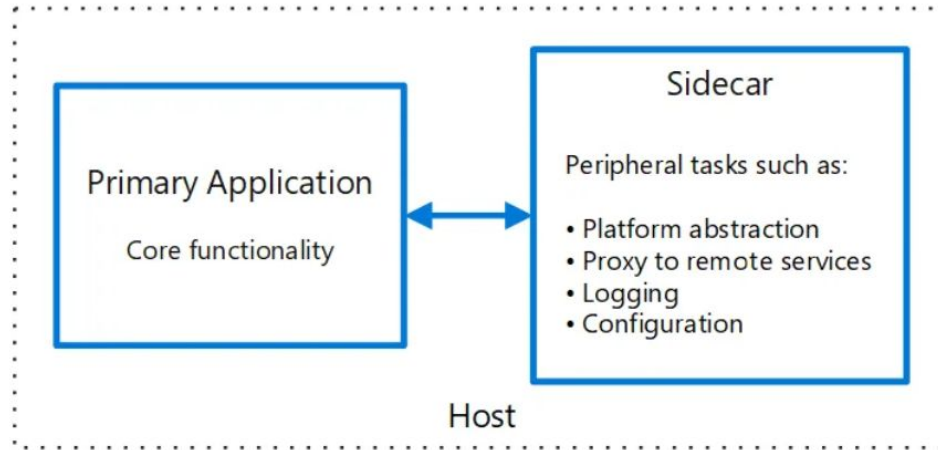


Read more: <https://docs.docker.com/build/guide/layers/>

Sidecar Design Pattern



Sidecar Design Pattern



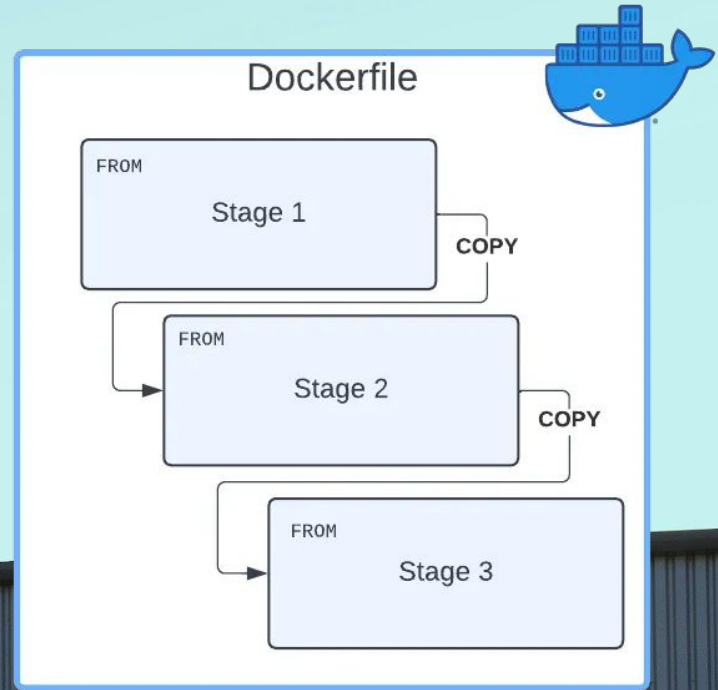
source: Microsoft docs

Multi Stage builds

- Use multiple FROM statements.
- Perform operation in one stage then copy stuff from that stage to another.

Use Case:

- Building, compiling
- Reduce size of image



Multi Stage builds

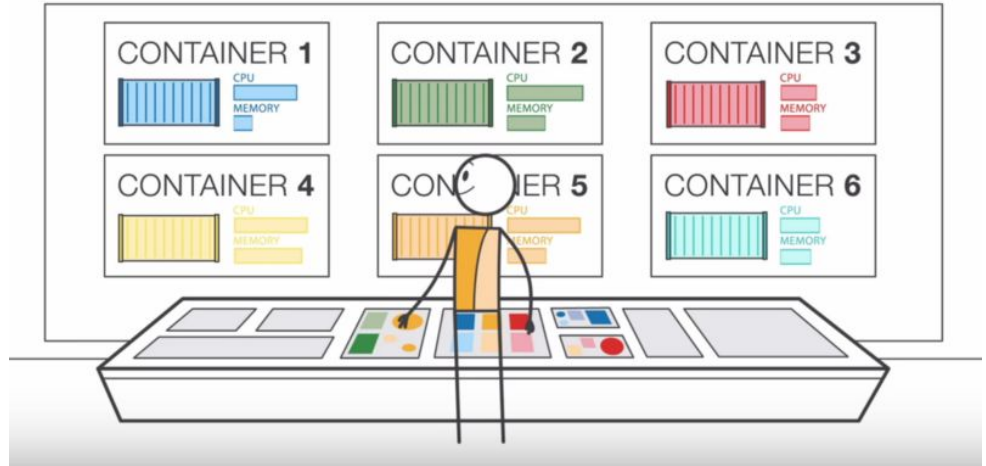
```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /source
# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore
# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /source/aspnetapp
RUN dotnet publish -c release -o /app --no-restore
# final stage/image

FROM mcr.microsoft.com/dotnet/aspnet:8.0
WORKDIR /app
COPY --from=build /app ./
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Running containers on cloud & at scale.

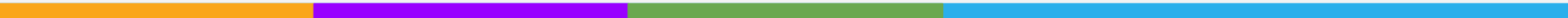
Containers as a service (CaaS) is a cloud-based service that allows DevOps Engineers to upload, organize, run, scale, and manage containers at scale.

- Amazon Elastic Container Service
- Google Kubernetes Engine



References

<https://docs.docker.com/get-started/overview/>



Assignment

Dockerize your project.

- Can be any project that you did in previous course.
- Try to incorporate multi-stage builds if possible.
- docker-compose if possible.
- Submit a fully commented **dockerfile**



End of Week 2

Q&A

