**DevOps Implementation – Abhishek Veeramalla**

**Containerization of the project**

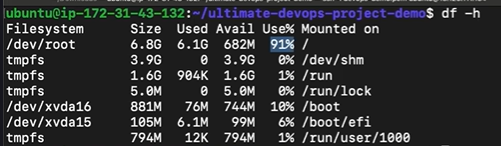
**Docker, Docker Compose**Connect to ec2 with key pair

* ssh -i devops-demo.pem ubuntu@ip-address

**docker compose**

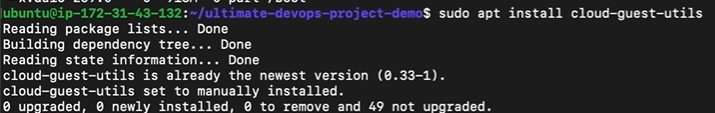
* In the market docker-compose is used to run the complex application locally (local machine) which has multiple microservices.
* Docker compose is used to run multiple containers and also it establishes communication between multiple containers.
* Some times docker compose is also used in dev CI/CD pipelines.

**Resizing EC2 volume and file system**

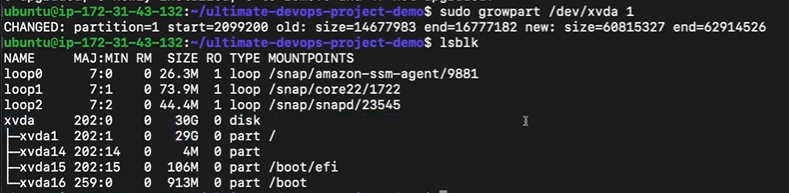
****

**Step 1: Modify and increase the size of the EC2 volume**

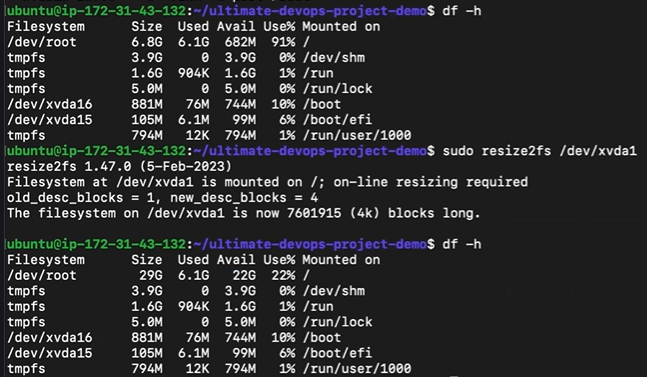
**Step 2: Grow the partition of the file system.**

****

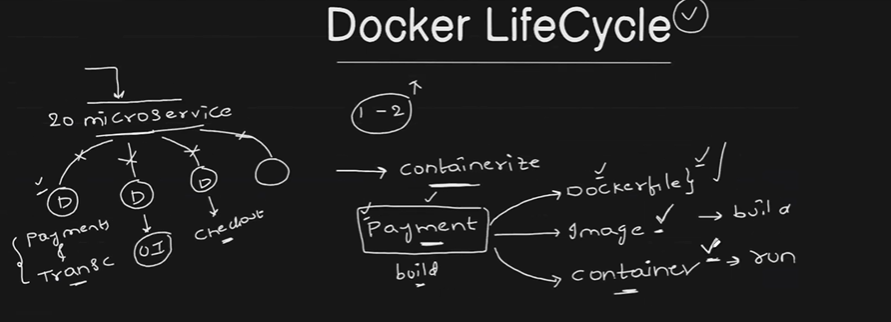
**Sudo apt install cloud-guest-utils**

****

**sudo growpart /dev/xvda 1**

****

**sudo resize2fs /dev/xvda1**

****

* In real time scenario projects you may work with 3-4 microservices with 1-2 developer Teams.
* First in order to run the application in containers or to implement containerization or to implement docker life cycle, you need to know how to build or run the application in local machine or locally in the EC2.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Containerization of first microservice - Product Catalog - Golang**

**# open vim editor in ubuntu linux**

* vim Dockerfile

# Dockerfile

-------------------------------------------------------------------------------

**FROM golang:1.22-alpine AS builder**

**WORKDIR /usr/src/app**

**# Copy the rest of the source code**

**COPY . .**

**# Download**

**RUN go mod download**

**RUN go build -o product-catalog ./**

**FROM alpine AS release**

**WORKDIR /usr/src/app**

**COPY --from=builder /usr/src/app/product-catalog ./**

**ENV PRODUCT\_CATALOG\_PORT=8088**

**ENTRYPOINT ["./product-catalog"]**

---------------------------------------------------------------------------

# to build image

* docker build -t knsharsha/product-catalog:v1

# to run container

* docker run knsharsha/product-catalog:v1

# check the output accordingly with developers.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Building and running locally in EC2, Ad microservice - Java**

* sudo apt install openjdk-21-jre-headless

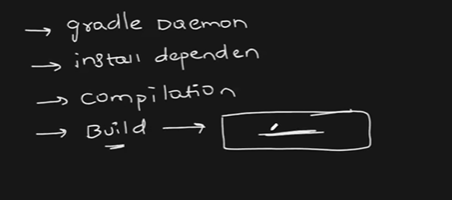
incase in order to give execute permission for the gradlew

* chmod +x ./gradlew

to build the grade Java application

* ./gradlew installDist

This command will do the following.



To run the Ad Service:

* export AD\_PORT=8080
* export FEATURE\_FLAG\_GRPC\_SERVICE\_ADDR=featureflagservice:50053
* ./build/install/opentelemetry-demo-ad/bin/Ad

**Containerization of Ad microservice - Java**

#Dockerfile

**FROM eclipse-temurin:21-jdk AS Builder**

**WORKDIR /usr/src/app/**

**COPY gradlew\* settings.gradle\* build.gradle .**

**COPY ./gradle ./gradle**

**RUN chmod +x ./gradlew**

**RUN ./gradlew**

**RUN ./gradlew downloadrepos**

**COPY**

**COPY ./pb ./proto**

**RUN chmod +x ./gradlew**

**RUN ./gradlew installDist -PprotoSourceDir=./proto**

**########################################################**

**FROM eclispe-temurin:21-jre**

**WORKDIR /usr/src/app/**

**COPY --from=builder /usr/src/app/ ./**

**ENV AD\_PORT=9099**

**ENTRYPOINT ["./build/install/opentelemetry-demo-ad/bin/Ad"]**

**Containerization of Recommendations microservice – Python**

# Dockerfile

**FROM python:3.12-slim-bookworm AS base**

**WORKDIR /usr/src/app**

**COPY requirements.txt ./**

**RUN pip install --upgrade pip**

**RUN pip install -r requirements.txt**

**COPY . .**

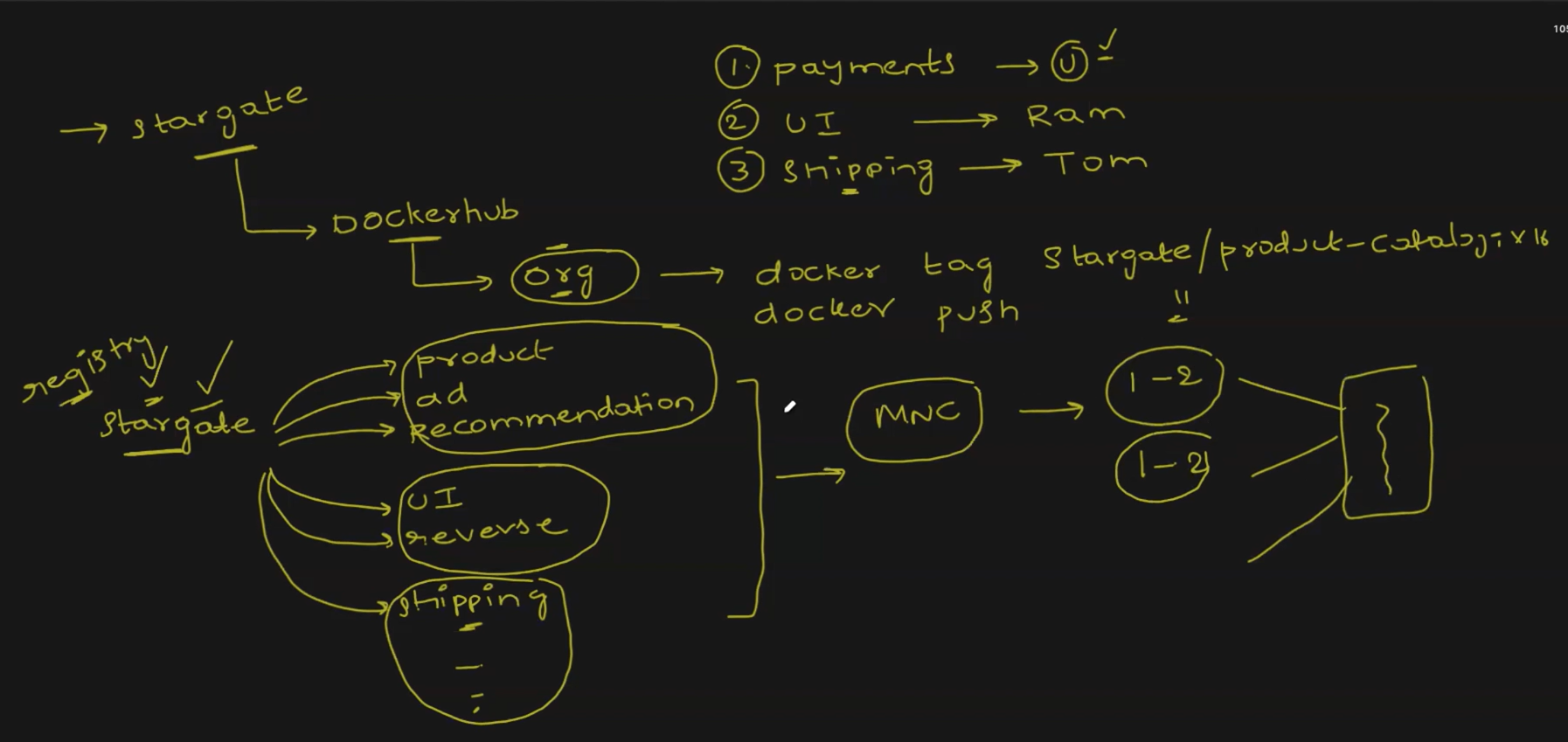
**ENTRYPOINT ["python", "recommendation\_server.py"]**

**Push Image**

* docker login -u <username>
* Enter password

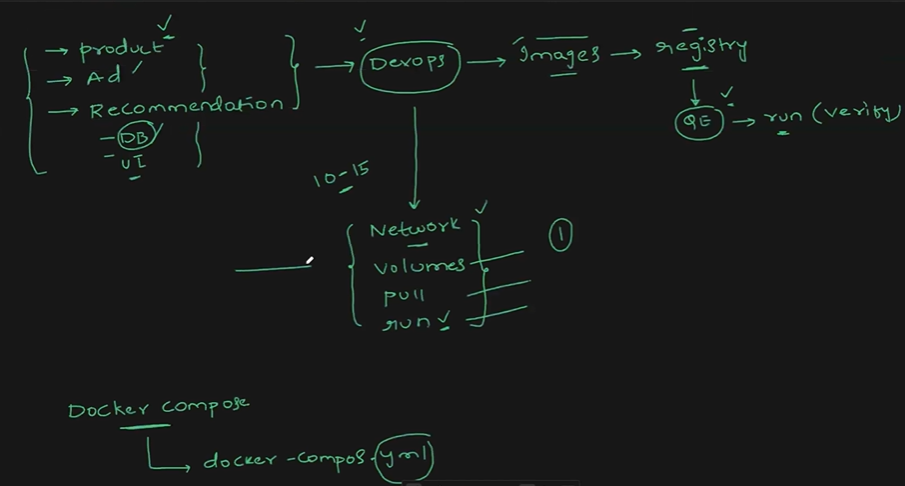
Here Username is “knsharsha”

* docker push docker.io/knsharsha/product-catalog:v1
* docker push docker.io/knsharsha/ad-service:v1
* docker push docker.io/knsharsha/recommendation-service:v1

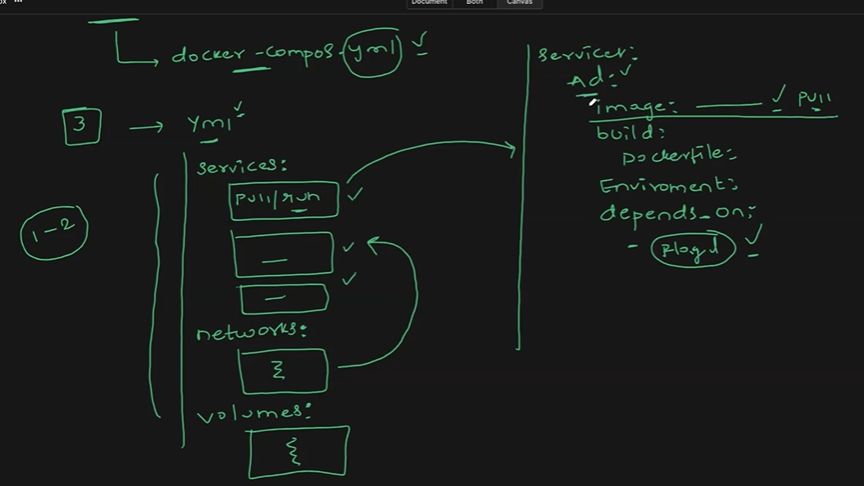
****

**Docker Compose**

Docker compose is used to run multiple containers at a time

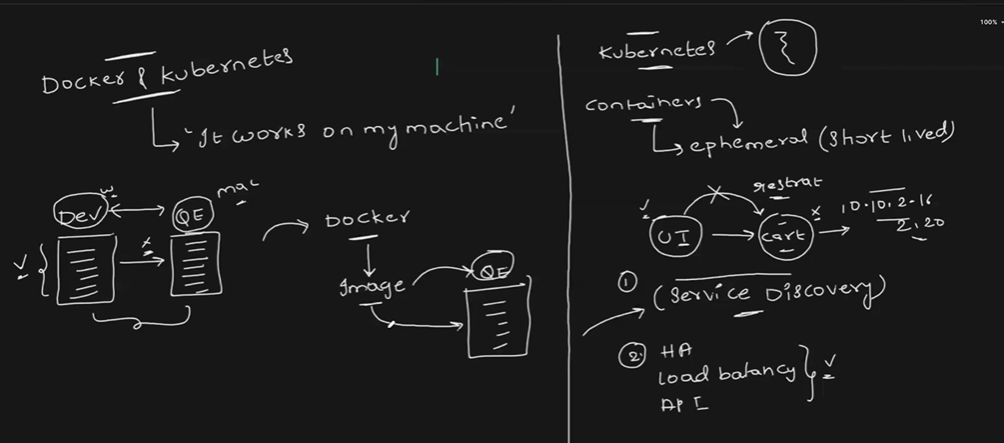


How to write a docker compose file



**Need for Container Orchestration**

Docker VS Kubernetes



**Service discovery problem:** Because of the “ephemeral” nature of the containers, when containers goes down and when it again goes up its ip address changes so one microservice and talk to another microservice.

This problem can be solved by “Kubernetes”

Mostly containers are not used in production, because in production there should be so many capabilities like,  
  
HA – High Availability

Load Balancing

API Gateway Integration.

Where these capabilities will be there in Kubernetes and docker is not having these capabilities to that extent.

* So this is where in order to have all these capabilities we need a Container Orchestration Engine called Kubernetes.  
    
  Docker Compose VS Kubernetes  
    
  A blackboard with white text

  AI-generated content may be incorrect.

why do we use kubernetes if docker compose is there ?

* **Copilot said:**
* **Copilot**
* Great question! While **Docker Compose** and **Kubernetes** both help manage containers, they serve different purposes and are suited for different scales and complexities of deployment. Here's a clear comparison to help you understand why Kubernetes is often preferred in production environments:
* **🐳 Docker Compose**
* **Best for**: Local development and small-scale deployments.
* **Functionality**:
  + Defines and runs multi-container Docker applications.
  + Uses a simple docker-compose.yml file.
  + Easy to set up and use.
* **Limitations**:
  + No built-in scaling or self-healing.
  + Limited networking and service discovery.
  + No native support for rolling updates or load balancing.
  + Not ideal for managing containers across multiple machines.
* **☸️ Kubernetes**
* **Best for**: Production-grade, large-scale, distributed systems.
* **Functionality**:
  + Orchestrates containers across a cluster of machines.
  + Handles **auto-scaling**, **self-healing**, **rolling updates**, and **load balancing**.
  + Provides **service discovery**, **networking**, and **persistent storage**.
  + Supports **declarative configuration** and **infrastructure as code**.
* **Complexity**:
  + Steeper learning curve.
  + Requires more setup and resources.
  + Often used with tools like Helm, ArgoCD, and Prometheus for full ecosystem support.
* **🔍 Summary Table**

| * **Feature** | * **Docker Compose** | * **Kubernetes** |
| --- | --- | --- |
| * Scale | * Small/local | * Large/distributed |
| * Setup | * Simple | * Complex |
| * Auto-scaling | * ❌ | * ✅ |
| * Self-healing | * ❌ | * ✅ |
| * Rolling updates | * ❌ | * ✅ |
| * Multi-host support | * ❌ | * ✅ |
| * Monitoring & logging | * Basic | * Advanced (via integrations) |
| * Use case | * Dev/test environments | * Production environments |

* **🚀 When to Use What?**
* **Use Docker Compose** for quick prototyping, local development, or small apps.
* **Use Kubernetes** when you need reliability, scalability, and automation in production.
* Would you like a visual diagram comparing their architectures or a sample migration guide from Docker Compose to Kubernetes?
* You said:
* yes I need a visual diagram
* **Copilot said:**
* **Copilot**
* Here's a visual diagram comparing **Docker Compose** and **Kubernetes** in terms of architecture and capabilities. Let me know if you'd like a version with annotations or a printable format!  
    
  