DAY 4:DEVOPS

1. Install Docker Desktop

For Windows:

1. Go to the official Docker website: https://www.docker.com/products/docker-desktop/
2. Click on "Download for Windows (WSL2)"
3. Run the installer after downloading
4. Follow the installation steps (enable WSL2 if prompted)
5. Restart your system if required
6. Open Docker Desktop from the Start Menu
7. Ensure Docker is running (you will see "Docker is running" status)

2. Create Folder Structure and Files:

Open Command Prompt, Terminal, or any shell:

```
mkdir microservices-project

cd microservices-project

mkdir order-service

mkdir user-service


cd order-service

touch Dockerfile app.py requirements.txt

cd ..


cd user-service

touch Dockerfile app.py requirements.txt

cd ..
```

3. Add Content to Files

order-service/app.py

```
from flask import Flask

app = Flask(__name__)
```

```
@app.route('/')
def home():
    return "Hello from Order Service"


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

order-service/requirements.txt

```
flask
```

order-service/Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

user-service/app.py

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def home():
    return "Hello from User Service"


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)
```

user-service/Dockerfile

```

FROM python:3.9-slim
WORKDIR /app
RUN pip install -r requirements.txt
```

EXPOSE 5001

CMD ["python", "app.py"]

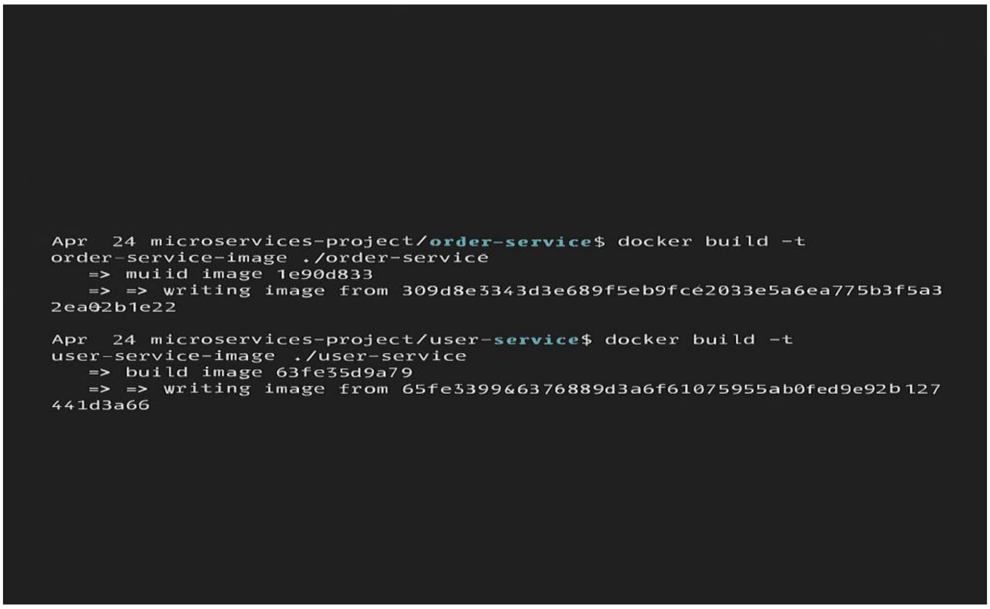
4. Build Docker Images

docker run -d -p 5000:5000 --name order-service-container order-service-image

docker run -d -p 5001:5001 --name user-service-container user-service-image

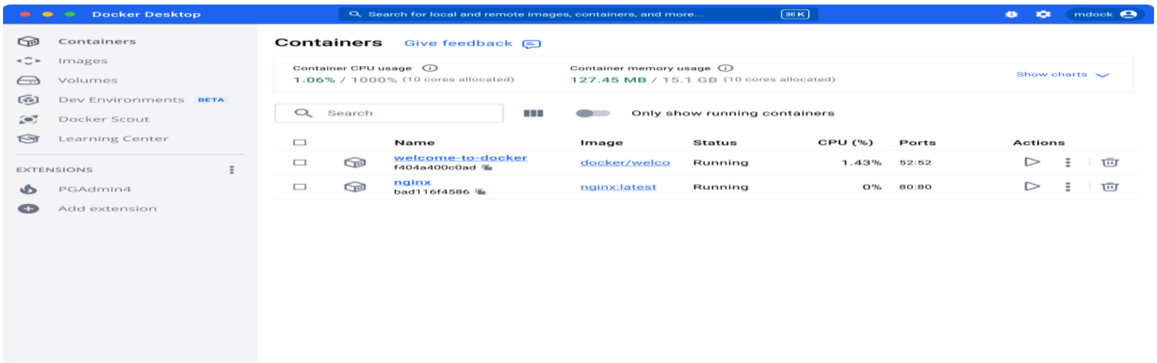
6. Verify in Docker Desktop

1. Open Docker Desktop
2. Go to the "Containers" tab
3. You will see both order-service-container and user-service-container listed and running



```
Apr  24 microservices-project/order-service$ docker build -t
order-service-image ./order-service
    => muiid image 1e90d833
    => => writing image from 309d8e3343d3e689f5eb9fcé2033e5a6ea775b3f5a3
2ea02b1e22

Apr  24 microservices-project/user-service$ docker build -t
user-service-image ./user-service
    => build image 63fe35d9a79
    => => writing image from 65fe3399&6376889d3a6f61075955ab0fed9e92b127
441d3a66
```



Containers

Container CPU usage 1.06% / 1000% (10 cores allocated)

Container memory usage 127.45 MB / 15.1 GB (10 cores allocated)

□		Name	Image	Status	CPU (%)	Ports	Actions
□		welcome-to-docker f404a400c0ad	docker/welco	Running	1.43%	52:52	
□		nginx bad116f4586	nginx:latest	Running	0%	80:80	

Creating an S3 Bucket in the AWS Console

1. Sign in to the AWS Management Console and open the S3 service.
2. Click “Create bucket.”
3. Enter a Bucket name (must be globally unique) and choose an AWS Region.
4. (Optional) Under Bucket settings for Block Public Access, leave defaults to block public access (recommended).

5. (Optional) Under Versioning, enable if you need object versioning.
6. (Optional) Under Encryption, choose AWS-managed key (SSE-S3) or your own KMS key.
7. (Optional) Add tags in the Tags section.
8. Review settings and click “Create bucket.”

2. Creating an S3 Bucket with Terraform

Prerequisites:

- Terraform installed (v1.0+).
- AWS credentials configured (e.g. via ~/.aws/credentials or environment variables).

In an empty project folder, create a file main.tf with:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
  required_version = ">= 1.0.0"
}

provider "aws" {
  region = "us-east-1"          # change to your region
}

resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-123" # change to a globally unique name
  acl    = "private"
  tags = {
    Environment = "dev"
    CreatedBy   = "Terraform"
  }
}
```

Initialize Terraform:

```
terraform init
```

Review the plan:

```
terraform plan
```

Apply the plan:

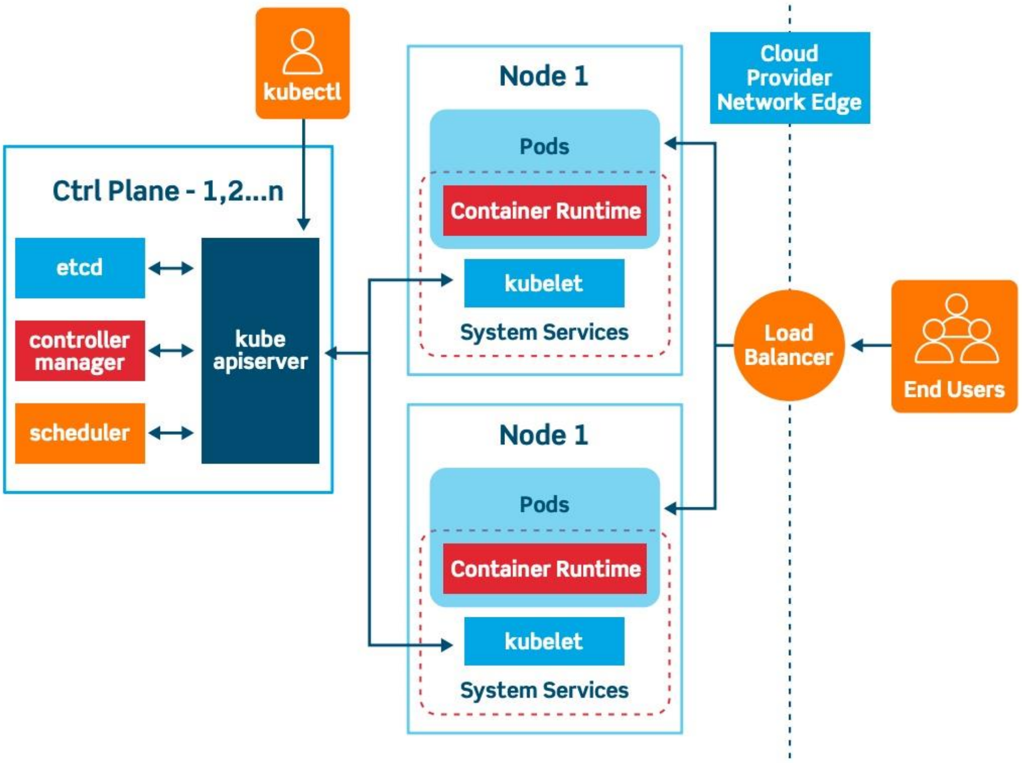
```
terraform apply
```

(When done) To destroy:

```
terraform destroy
```

Introduction to Kubernetes

Kubernetes is an orchestration platform—a system or tool that automates deployment, scaling, management, and operation of containerized applications. Note: Kubernetes uses Docker (or another container runtime) internally to run containers. With Kubernetes (often abbreviated "K8s"), you manage your Docker containers declaratively.



Docker Swarm vs. Kubernetes

- Auto-scaling
  - Docker Swarm: scaling must be triggered manually (docker service scale …).
  - Kubernetes: supports Horizontal Pod Autoscaler (HPA) to scale pods based on CPU, memory, or custom metrics.
- Production readiness
  - Docker Swarm: simpler, but fewer enterprise-grade features.
  - Kubernetes: richer feature set (self-healing, rolling updates, auto-scaling, namespaces) and is the industry standard for production.
- Recommendation
  For production deployments, Kubernetes is highly recommended. It is effectively the successor to Docker Swarm for orchestrating containers at scale.

Auto-Scaling

Auto-scaling automatically increases or decreases the number of running containers (pods) based on observed load (e.g., CPU usage, request rate). In Kubernetes, this is implemented via the Horizontal Pod Autoscaler.

What Is a Kubernetes Cluster?

- Cluster: a group of servers (nodes) managed together.
  - Master Node(s): control plane components (API server, scheduler, controller manager) that accept user/developer instructions.
  - Worker Node(s): run the containerized applications as pods.
- Workflow:
  1. You (DevOps engineer or developer) submit a manifest (deployment, service, etc.) to the Kubernetes API on the master node.
  2. The master node schedules pods onto worker nodes.
  3. Worker nodes run your containers inside pods.
- High Availability: by distributing pods across multiple nodes, Kubernetes ensures your application stays available even if individual nodes fail.

Installing kubectl on Linux

```
# Download the binary

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
# Make it executable

chmod +x kubectl
```

```
# Move to PATH
```

```
sudo mv kubectl /usr/local/bin/


# Verify
kubectl version –client
```

Installing AWS CLI

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws –version
```

Installing Git

```
sudo yum install git -y   # on Amazon Linux / CentOS
git --version             # e.g., git version 2.47.1
```

Clone the Voting App Repository

```
git clone https://github.com/N4si/K8s-voting-app.git
cd K8s-voting-app
ls -l
cd manifests
ls
cd ..
```

aws configureConnect to an EKS Clusteraws configure

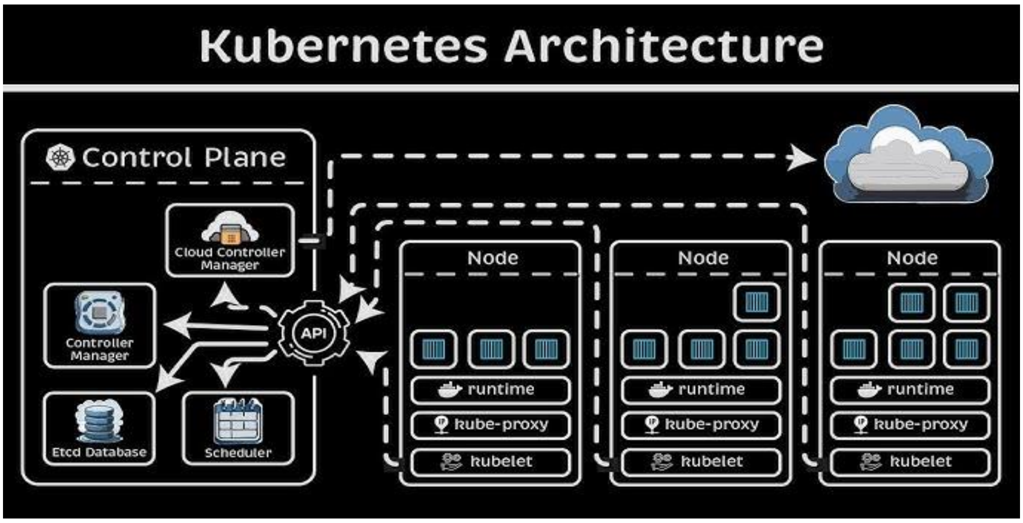
```
aws configure
```

Update kubeconfig for your EKS cluster

```
aws eks update-kubeconfig --name anvitha-cluster --region ap-south-1
```

Verify nodes

```
kubectl get nodes
```


Kubernetes Architecture
Control Plane
Cloud Controller Manager
Controller Manager
API
Etcd Database
Scheduler
Node
runtime
kube-proxy
kubelet
Node
runtime
kube-proxy
kubelet
Node
runtime
kube-proxy
kubelet