**Objective: Understand migration strategies to move applications to EKS.**

**Tasks:**

1. Containerize a legacy application public app in the docker hub.

2. Deploy it on a local Kubernetes cluster.

3. Migrate the application to EKS.

**Documentation:**

- Benefits of containerization.

- Strategies for migration to Kubernetes.

- AWS tools to facilitate migration.

**Tasks: 1**

**1. Containerize a legacy application:**

**Benefits of Containerization:**

1. Isolation: Each container runs in isolation, ensuring that the app has all dependencies packaged with it.
2. Consistency: Containers ensure your app runs the same regardless of where the container is run.
3. Scalability: Easily scale apps by just running more instances of the container.
4. Portability: Easily move apps across different cloud or OS environments.
5. Resource Efficiency: Containers share the same OS kernel, making them lightweight compared to VMs.

Steps:

For this example, let's assume we have a simple Python Flask application.

step-1. Create a simple Flask application **(app.py)**:

| from flask import Flask  app = Flask(\_\_name\_\_)  @app.route('/')  def hello():  return "Hello, World!"  if \_\_name\_\_ == "\_\_main\_\_":  app.run(host="0.0.0.0", port=5000) |
| --- |

2. Create a Dockerfile for the Flask application:

| FROM python:3.8-slim  WORKDIR /app  COPY requirements.txt requirements.txt  RUN pip install -r requirements.txt  COPY . .  CMD ["python", "app.py"] |
| --- |

3. You would also need a **requirements.txt** file:

| flask==2.0.1 |
| --- |

Build the Docker image:

| docker build -t my-flask-app:latest . |
| --- |

image reload it into Minikube:

| docker save my-flask-app:latest | (eval $(minikube docker-env) && docker load) |
| --- |

Optional:

| minikube image load image-name:tag |
| --- |

**Tasks: 2**

**2. Deploy it on a local Kubernetes cluster:**

Strategies for Migration to Kubernetes:

1. Rehost: Lift and shift your application directly to Kubernetes. This is the quickest approach but might not be optimized.
2. Refactor: Modify the application to leverage Kubernetes features like ConfigMaps, Secrets, etc.
3. Re-architect: This involves changing the architecture of the application to fit a microservices pattern or similar.
4. Rebuild: Rewrite the application from scratch to be cloud-native.
5. Replace: Replace the existing application with a SaaS or similar solution.

For this lab, we will use the rehost strategy.

* **Set up a local Kubernetes cluster using tools like** [**Minikube**](https://minikube.sigs.k8s.io/docs/start/) **or kind.**
* **Create a Kubernetes deployment file (deployment.yaml):**

| apiVersion: apps/v1 kind: Deployment metadata:  name: flask-app-deployment spec:  replicas: 2  selector:  matchLabels:  app: flask-app  template:  metadata:  labels:  app: flask-app  spec:  containers:  - name: flask-app-container  image: my-flask-app:latest  imagePullPolicy: Never  ports:  - containerPort: 5000  ---  apiVersion: v1 kind: Service metadata:  name: flask-app-service spec:  selector:  app: flask-app  ports:  - protocol: TCP  port: 80  targetPort: 5000  type: NodePort |
| --- |

**Tasks: 3**

**3. Migrate the application to EKS:**

**AWS Tools to Facilitate Migration:**

* EKS: Managed Kubernetes Service by AWS.
* ECR: AWS's Docker container registry. Useful to store your Docker images.
* AWS CLI: Command-line tool for AWS services.

Requirements Tools:  
1. [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

2. eksctl

Create CLuster:

| eksctl create cluster \  --name=eks-labs \  --version=1.27 \  --region=us-east-2 \  --spot \  --node-type=t2.medium \  --nodes=1 \  --nodes-min=1 \  --nodes-max=2 \  --nodegroup-name=muzammil-node-grp \  --managed |
| --- |

**Steps:**

Push the Docker image to ECR:

| # Create a repository in ECR aws ecr create-repository --repository-name my-flask-app  # Authenticate Docker to the ECR registry aws ecr get-login-password --region <REGION> | docker login --username AWS --password-stdin <YOUR\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com  # Tag the image for ECR docker tag my-flask-app:latest <YOUR\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com/my-flask-app:latest  # Push the image docker push <YOUR\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com/my-flask-app:latest |
| --- |

**Update the deployment.yaml to use the ECR image:**

| ...  image: <YOUR\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com/my-flask-app:latest ... |
| --- |

Set up an EKS cluster using the AWS Management Console or AWS CLI.

Deploy your application to EKS:

| kubectl apply -f deployment.yaml |
| --- |