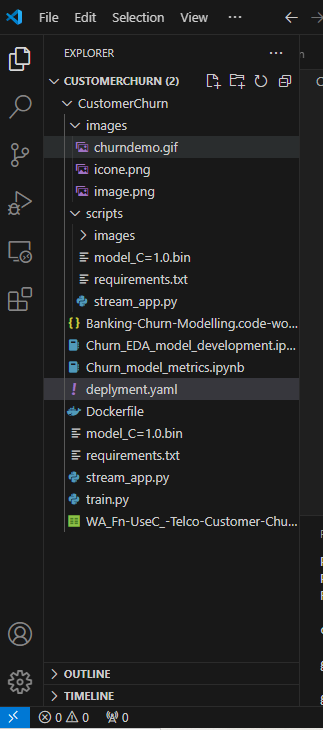
To implement the concepts discussed in the IEEE paper *"Containerization: Cloud Computing-based Inspiration Technology for Adoption through Docker and Kubernetes"*,We followed these steps to set up and deploy a containerized application of Customer Churn using both Docker and Kubernetes. The goal is to understand containerization for cloud computing and orchestration technologies, particularly through Docker and Kubernetes.

This is how the code structure looks like:

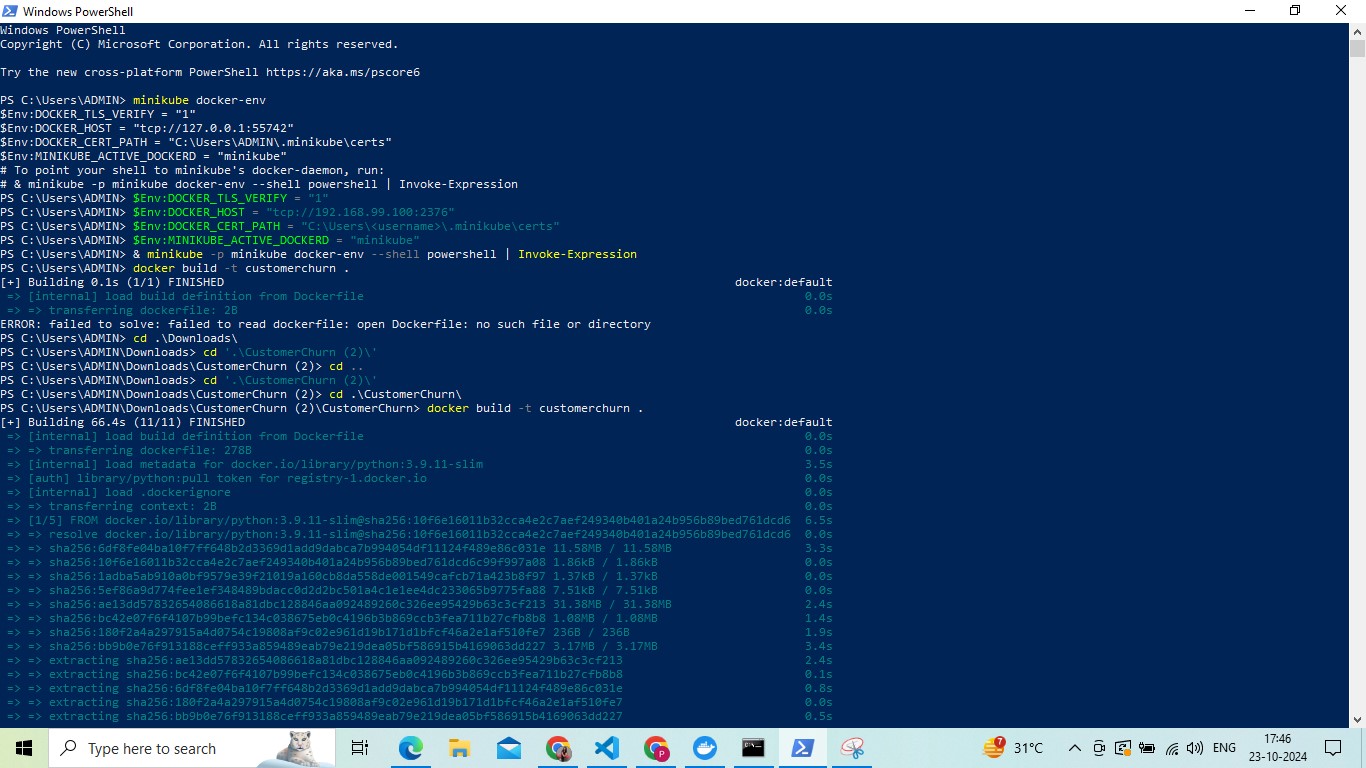


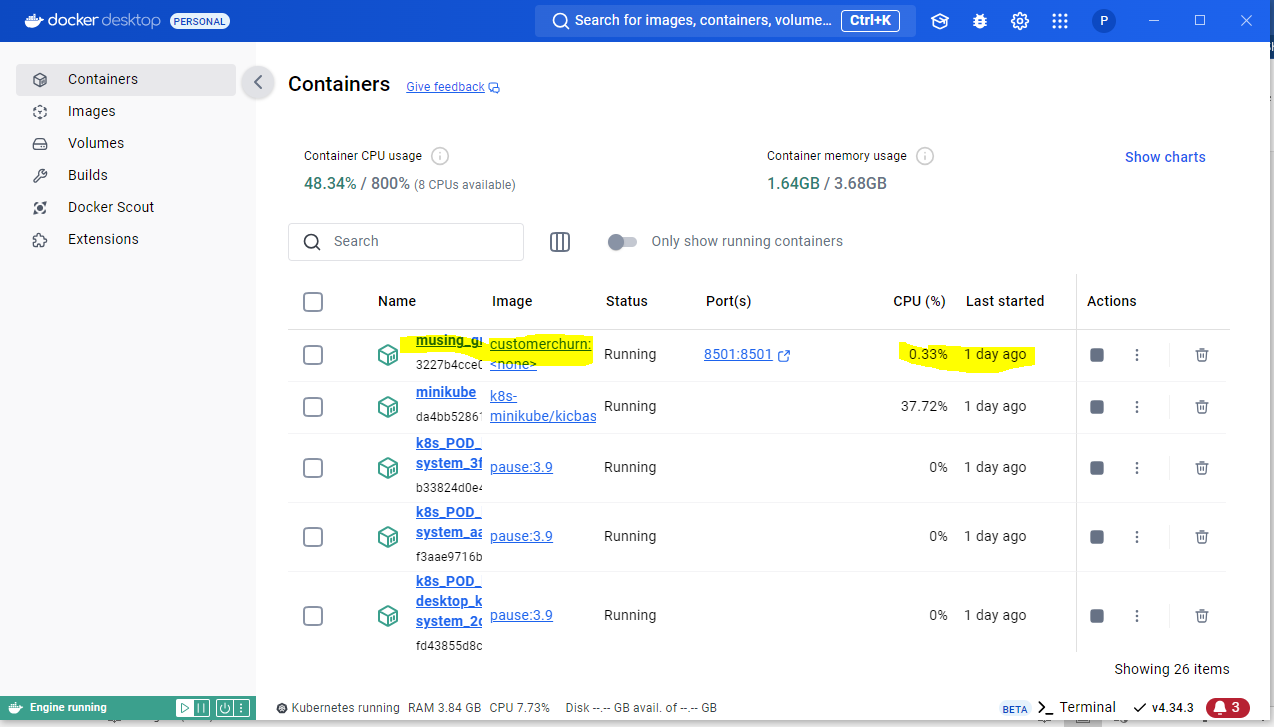
**Step 1: Set Up Docker Environment**

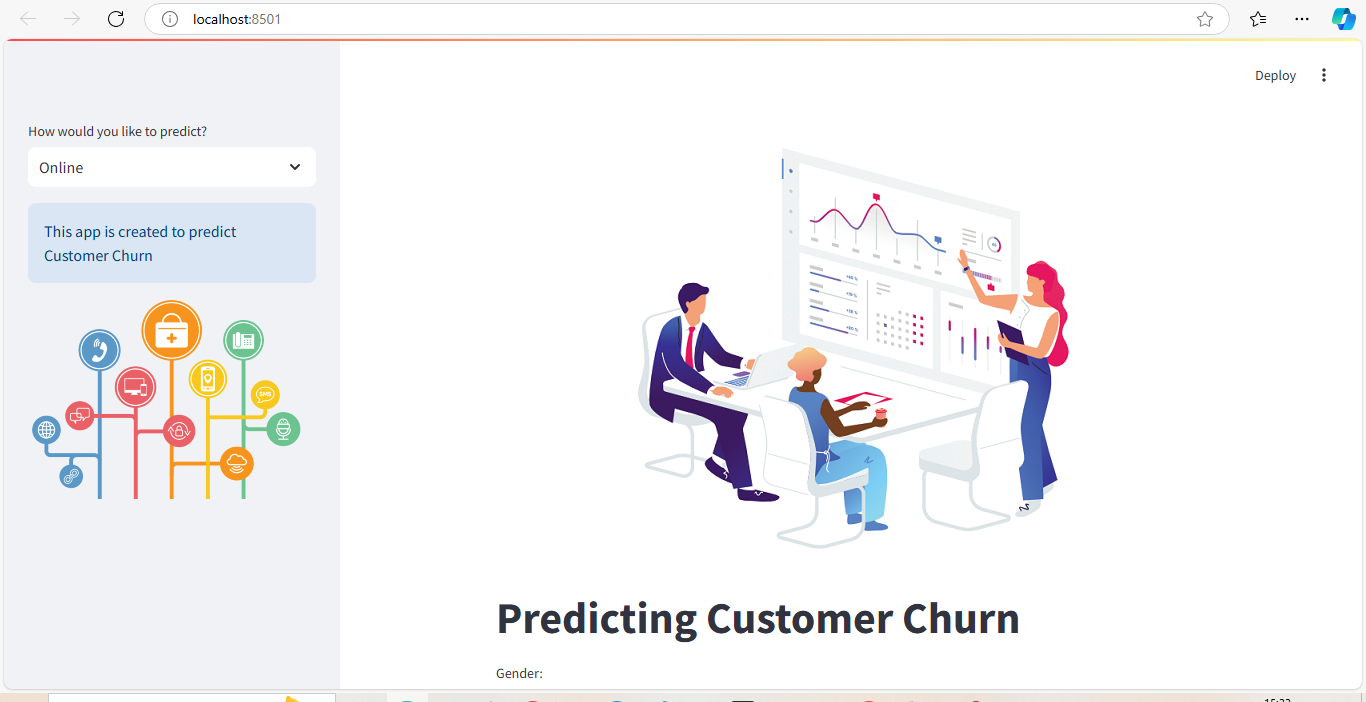
* **Installed Docker** on our local machine.
* **Create Docker file**: Developed a simple application (i.e Python app). Create a Docker file to containerize the application.
  + Example Docker file:
* FROM python:3.9.11-slim
* RUN /usr/local/bin/python -m pip install --upgrade pip
* WORKDIR /app
* COPY /scripts .
* RUN pip install -r requirements.txt
* EXPOSE 8501
* ENTRYPOINT ["streamlit", "run"]
* CMD ["stream\_app.py"]
* **Build and Run Docker Container**:

docker build –t customerchurn.

docker run -d -p 8501:8501 customerchurn







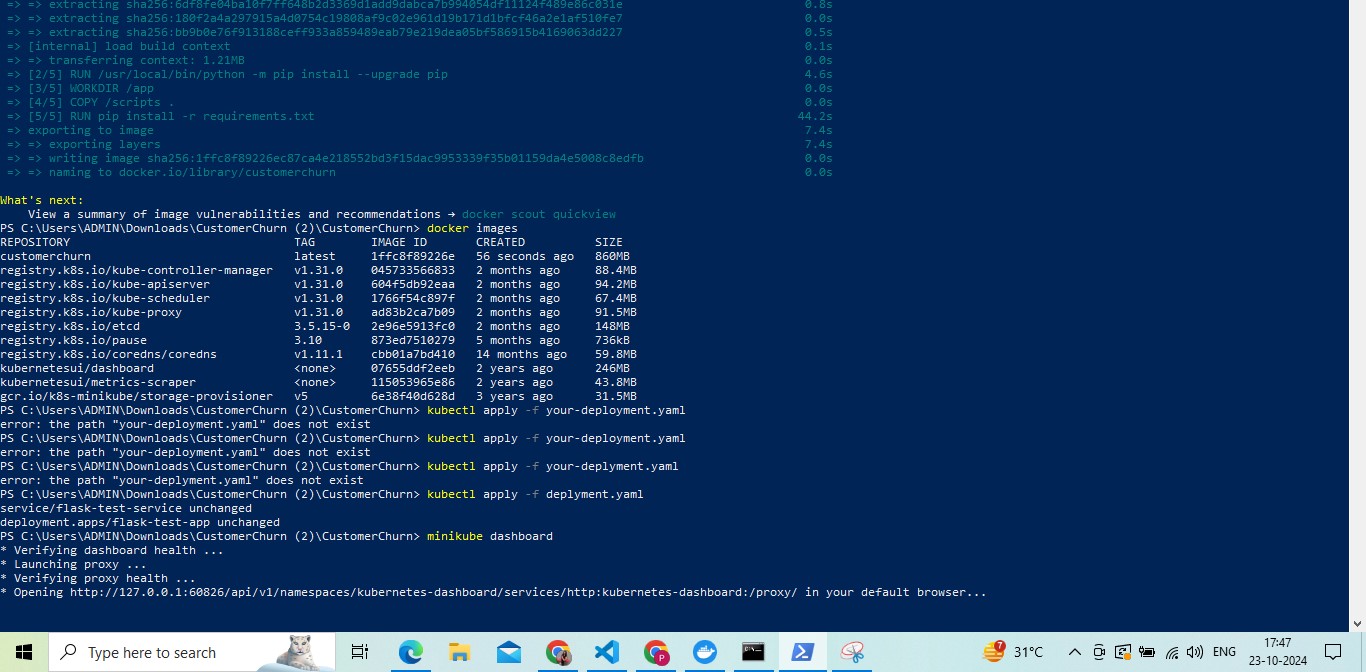
**Step 2: Set Up Kubernetes Environment**

* **Install Kubernetes** on our machine
  + For local testing, installed **Minikube**
  + **Install kubectl** to manage Kubernetes clusters

**Step 3: Create Kubernetes Deployment and Service**

* **Create Deployment YAML**: Created a deplyment.yaml file to define app’s deployment in Kubernetes. This file contained both service & deployment .
* apiVersion: v1
* kind: Service
* metadata:
* name: flask-test-service
* spec:
* selector:
* app: flask-test-app
* ports:
* - protocol: "TCP"
* port: 6000
* targetPort: 5000
* type: LoadBalancer
* ---
* apiVersion: apps/v1
* kind: Deployment
* metadata:
* name: flask-test-app
* spec:
* selector:
* matchLabels:
* app: flask-test-app
* replicas: 5
* template:
* metadata:
* labels:
* app: flask-test-app
* spec:
* containers:
* - name: flask-test-app
* image: customerchurn
* imagePullPolicy: Never
* ports:
* - containerPort: 5000
* **Deploy to Kubernetes**:
  + Deploy the application to Kubernetes by applying the YAML files:

kubectl apply -f deployment.yaml



**Step 4: Monitor and Scale Kubernetes Pods**

* **Scaling**: Use Kubernetes to scale your application.

kubectl scale deployment myapp-deployment --replicas=5

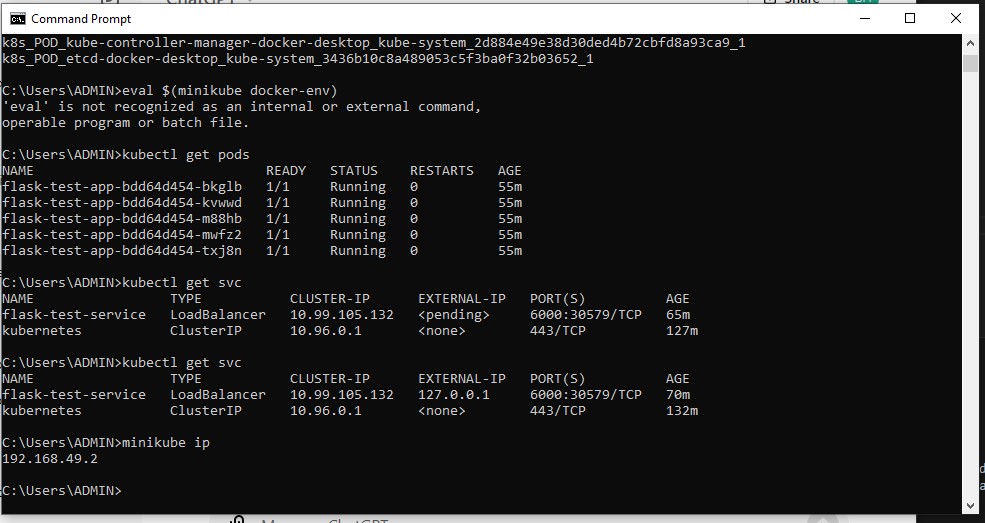


* **Monitoring**: View the status of your pods and services using commands like:

kubectl get pods

kubectl get services

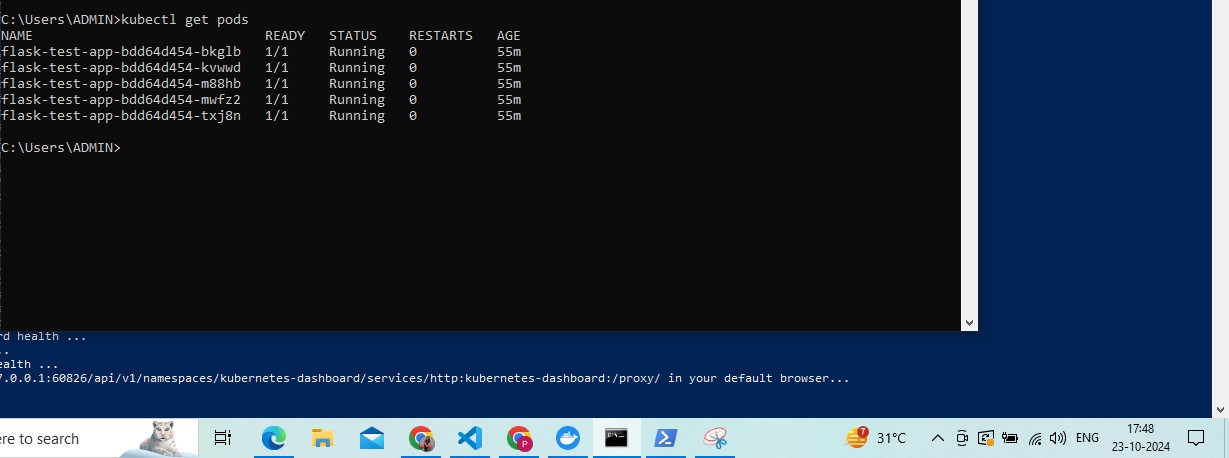
kubectl logs <pod\_name>



**Step 5: Kubernetes Features Exploration**

* **Horizontal Pod Autoscaler (HPA)**: Set up HPA to automatically scale the number of pods based on CPU utilization.

kubectl autoscale deployment myapp-deployment --cpu-percent=50 --min=2 --max=10



**Step 6: Evaluate Performance and Analyse Results**

* **Containerization Benefits**: After deploying, the scalability, resource usage, and efficiency improved provided by containerization.
* **Kubernetes Benefits**: Kubernetes features such as auto-scaling, load balancing, and failover handling enhanced the performance. Minikube dashboard eased the process of monitoring & failovering.