E-Commerce App Deployment Using Kubernetes

This guide provides step-by-step instructions for setting up a simple e-commerce application using Flask (Python) for the backend and Nginx for the frontend, and deploying it in Kubernetes with Minikube.

1. Setup Directory Structure

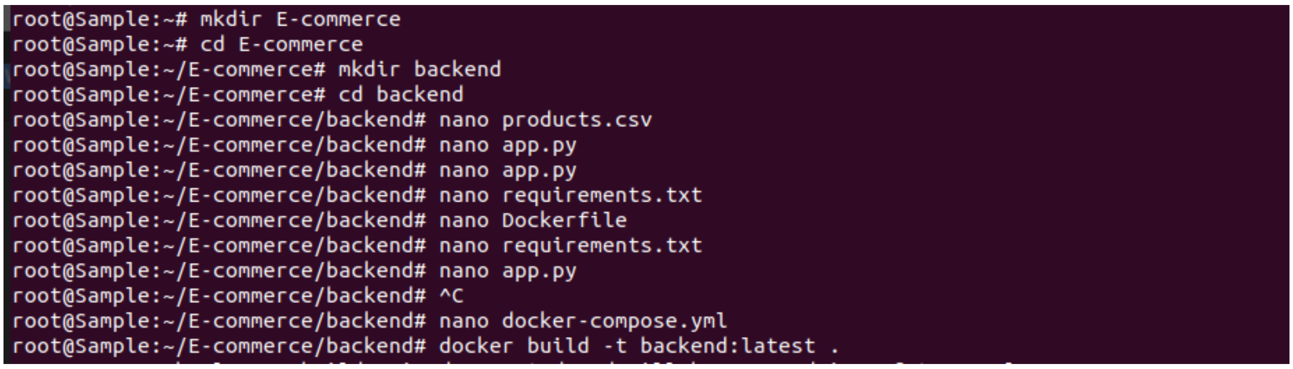
First, create a project directory to keep all files organized.

```
mkdir E-commerce && cd E-commerce
```

Backend Setup

Create a backend directory and navigate into it.

```
mkdir backend && cd backend
```



```
root@Sample:~# mkdir E-commerce
root@Sample:~# cd E-commerce
root@Sample:~/E-commerce# mkdir backend
root@Sample:~/E-commerce# cd backend
root@Sample:~/E-commerce/backend# nano products.csv
root@Sample:~/E-commerce/backend# nano app.py
root@Sample:~/E-commerce/backend# nano app.py
root@Sample:~/E-commerce/backend# nano requirements.txt
root@Sample:~/E-commerce/backend# nano Dockerfile
root@Sample:~/E-commerce/backend# nano requirements.txt
root@Sample:~/E-commerce/backend# nano app.py
root@Sample:~/E-commerce/backend# ^C
root@Sample:~/E-commerce/backend# nano docker-compose.yml
root@Sample:~/E-commerce/backend# docker build -t backend:latest .
```

Create `products.csv`

This file will store product details in CSV format.

```
nano products.csv
```

Paste the following sample data:

```
id,name,price,quantity
1,Smartphone,15000,25
2,Laptop,45000,15
3,Headphones,1500,50
4,Smartwatch,8000,30
5,Tablet,20000,20
6,Wireless Mouse,700,100
7,Bluetooth Speaker,1200,60
8,External Hard Drive,4000,40
9,USB Flash Drive,500,150
10,Monitor,10000,10
```

Create `app.py`

This script sets up a Flask server to read the CSV file and return product data as JSON.

```
nano app.py
```

Paste the following Python script:

```
from flask import Flask
import pandas as pd

app = Flask(__name__)

@app.route("/products", methods=['GET'])
def read_data():
    df = pd.read_csv("products.csv")  # Ensure products.csv exists
    json_data = df.to_json()
    return json_data

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5050)
```

Create `requirements.txt`

This file lists the dependencies required for the backend.

```
nano requirements.txt
```

Add dependencies:

```
flask
pandas
```

Create `Dockerfile`

This Dockerfile defines how to package the backend application into a container.

```
nano Dockerfile
```

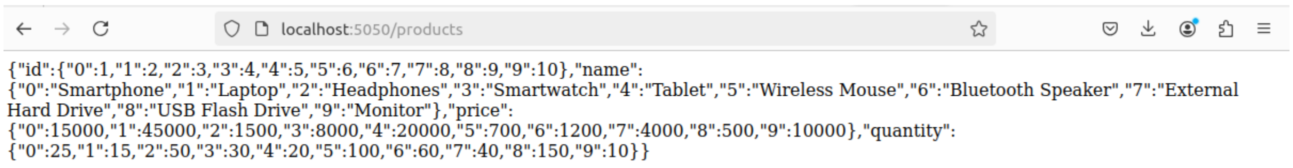
Paste the following:

```
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5050
CMD ["python", "app.py"]
```

Build & Run Backend Container

Build and run the backend container.

```
docker build -t backend:latest .
docker run -itd -p 5050:5050 backend
docker logs $(docker ps -q --filter "ancestor=backend")
```



{"id":{"0":1,"1":2,"2":3,"3":4,"4":5,"5":6,"6":7,"7":8,"8":9,"9":10},"name":
{"0":"Smartphone","1":"Laptop","2":"Headphones","3":"Smartwatch","4":"Tablet","5":"Wireless Mouse","6":"Bluetooth Speaker","7":"External
Hard Drive","8":"USB Flash Drive","9":"Monitor"},"price":
{"0":15000,"1":45000,"2":1500,"3":8000,"4":20000,"5":700,"6":1200,"7":4000,"8":500,"9":10000},"quantity":
{"0":25,"1":15,"2":50,"3":30,"4":20,"5":100,"6":60,"7":40,"8":150,"9":10}}

Frontend Setup

Create a frontend directory and navigate into it.

```
cd ..
mkdir frontend && cd frontend
```

Create `index.html`

This HTML file loads the product list from the backend.

```
nano index.html
```

Paste the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Commerce Store</title>
    <script>
        async function fetchProducts() {
            const response = await fetch("http://localhost:5050/products");
            const products = await response.json();
            let output = "<h2>Product List</h2><ul>";
            for (const id in products.name) {
                output += `<li>${products.name[id]} -
$${products.price[id]}</li>`;
            }
            output += "</ul>";
            document.getElementById("product-list").innerHTML = output;
        }
    </script>
</head>
<body onload="fetchProducts()">
    <h1>Welcome to Our Store</h1>
    <div id="product-list">Loading...</div>
</body>
</html>
```

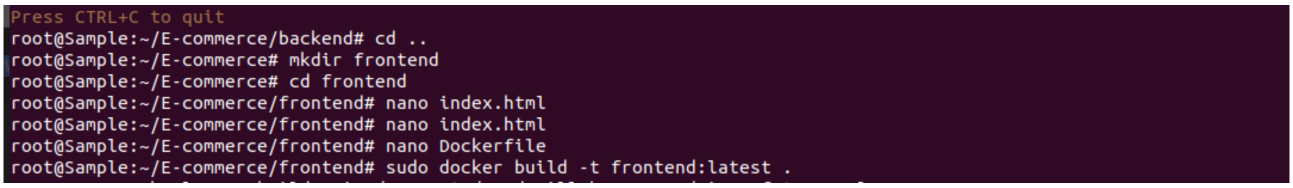
Create `Dockerfile`

This Dockerfile packages the frontend as an Nginx container.

```
nano Dockerfile
```

Paste:

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
```



```
Press CTRL+C to quit
root@Sample:~/E-commerce/backend# cd ..
root@Sample:~/E-commerce# mkdir frontend
root@Sample:~/E-commerce# cd frontend
root@Sample:~/E-commerce/frontend# nano index.html
root@Sample:~/E-commerce/frontend# nano index.html
root@Sample:~/E-commerce/frontend# nano Dockerfile
root@Sample:~/E-commerce/frontend# sudo docker build -t frontend:latest .
```

Build & Run Frontend Container

```
docker build -t frontend:latest .
```

2. Kubernetes Deployment

Create a `k8s` directory for Kubernetes configuration files.

```
cd ..
mkdir k8s && cd k8s
```

Backend Deployment (`backend-deployment.yaml`)

Defines a backend pod in Kubernetes.

```
nano backend-deployment.yaml
```

Paste:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: backend:latest
        ports:
        - containerPort: 5050
```

Frontend Deployment (`frontend-deployment.yaml`)

Defines a frontend pod in Kubernetes.

```
nano frontend-deployment.yaml
```

Paste:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: frontend:latest
        ports:
        - containerPort: 3000
```

Connecting Frontend & Backend (`service.yaml`)

Defines services for communication between frontend and backend.

```
nano service.yaml
```

Paste:

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
```

```
      port: 5050
      targetPort: 5050
  type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: NodePort
```

ConfigMap (`configmap.yaml`)

Stores backend configuration values.

```
nano configmap.yaml
```

Paste:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-config
data:
  DATABASE_FILE: "/backend/products.csv"
```

3. Installing Kubernetes

Instructions to install Minikube and `kubectl`.

Step 1: Install Docker

```
sudo apt update
sudo apt install -y docker.io
```

Step 2: Verify Docker Installation

```
docker --version
```

You should see output similar to: `Docker version 20.10.12, build e91ed57`

Step 3: Enable and Start Docker

```
sudo systemctl enable docker
sudo systemctl start docker
```

Check Docker status:

```
sudo systemctl status docker
```

Kubectl is the command-line tool used to interact with a Kubernetes cluster.

Step 1: Download Kubectl

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

Step 2: Install Kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

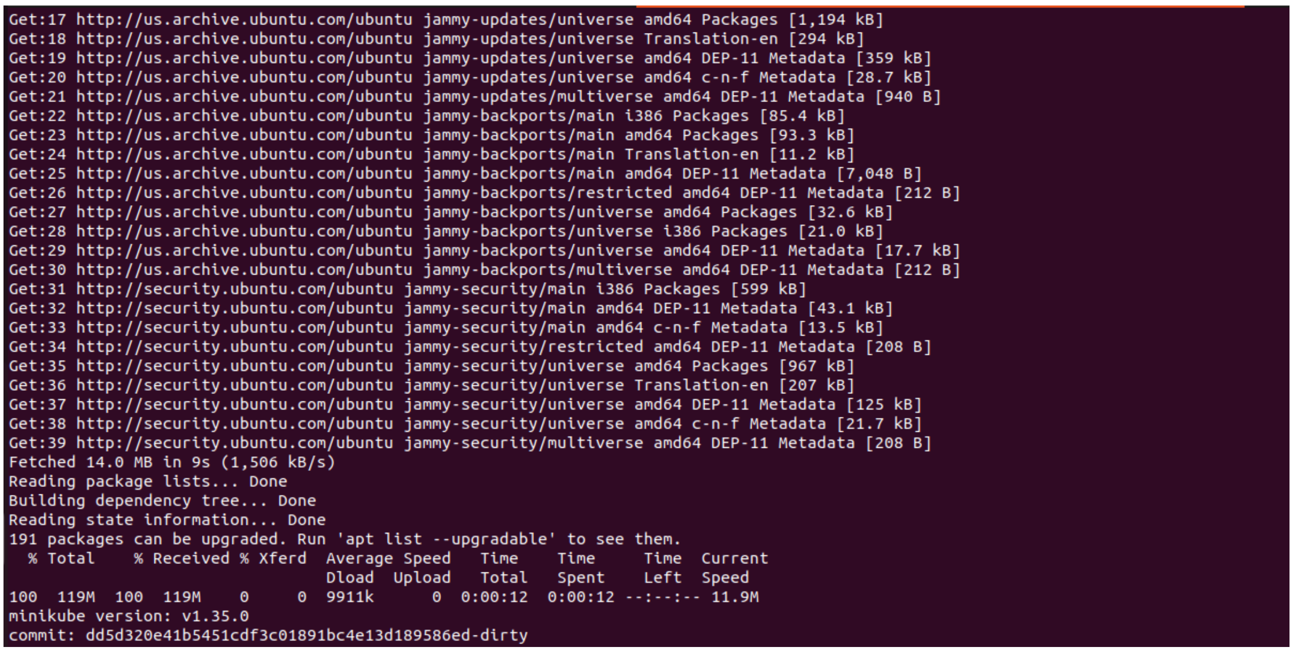
Step 3: Verify Installation

```
kubectl version --client
```

If the installation is successful, it will display the version information.

4. Installing Minikube

Minikube provides a local Kubernetes cluster, making it ideal for development and testing.



```
Get:17 http://us.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1,194 kB]
Get:18 http://us.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [294 kB]
Get:19 http://us.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 DEP-11 Metadata [359 kB]
Get:20 http://us.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [28.7 kB]
Get:21 http://us.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 DEP-11 Metadata [940 B]
Get:22 http://us.archive.ubuntu.com/ubuntu jammy-backports/main i386 Packages [85.4 kB]
Get:23 http://us.archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [93.3 kB]
Get:24 http://us.archive.ubuntu.com/ubuntu jammy-backports/main Translation-en [11.2 kB]
Get:25 http://us.archive.ubuntu.com/ubuntu jammy-backports/main amd64 DEP-11 Metadata [7,048 B]
Get:26 http://us.archive.ubuntu.com/ubuntu jammy-backports/restricted amd64 DEP-11 Metadata [212 B]
Get:27 http://us.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [32.6 kB]
Get:28 http://us.archive.ubuntu.com/ubuntu jammy-backports/universe i386 Packages [21.0 kB]
Get:29 http://us.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 DEP-11 Metadata [17.7 kB]
Get:30 http://us.archive.ubuntu.com/ubuntu jammy-backports/multiverse amd64 DEP-11 Metadata [212 B]
Get:31 http://security.ubuntu.com/ubuntu jammy-security/main i386 Packages [599 kB]
Get:32 http://security.ubuntu.com/ubuntu jammy-security/main amd64 DEP-11 Metadata [43.1 kB]
Get:33 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.5 kB]
Get:34 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 DEP-11 Metadata [208 B]
Get:35 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [967 kB]
Get:36 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [207 kB]
Get:37 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 DEP-11 Metadata [125 kB]
Get:38 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [21.7 kB]
Get:39 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 DEP-11 Metadata [208 B]
Fetched 14.0 MB in 9s (1,506 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
191 packages can be upgraded. Run 'apt list --upgradable' to see them.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  119M  100  119M    0     0  9911k      0  0:00:12  0:00:12 --:--:-- 11.9M
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
```

Step 1: Download Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
```

Step 2: Install Minikube

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Step 3: Verify Minikube Installation

```
minikube version
```

This should return the installed Minikube version.

5. Starting Minikube

Once Minikube is installed, you can start it using the Docker driver.

```
minikube start --driver=docker
```

This command will:

- Download the necessary Kubernetes images.
- Start a single-node Kubernetes cluster.
- Configure kubectl to interact with the cluster.

Check the status of the Minikube cluster:

```
minikube status
```

Verify that Kubernetes is running:

```
kubectl get nodes
```

Expected output:

```
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   3m24s   v1.32.0
```

6. Enabling the Kubernetes Dashboard (Optional)

Minikube includes a web-based Kubernetes dashboard. To enable it, run:

```
minikube dashboard
```

This will open a web browser with the Kubernetes dashboard.

7. Managing Minikube

Stopping Minikube

To stop the Minikube cluster without deleting it:

```
minikube stop
```

Deleting Minikube Cluster

To remove Minikube completely:

```
minikube delete
```

Checking Running Services

To list all Kubernetes services:

```
kubectl get services
```

8. Troubleshooting Tips

1. If Minikube Fails to Start

Try deleting and restarting Minikube:

```
minikube delete
minikube start --driver=docker
```

2. If Kubectl Cannot Connect to Minikube

Check if Minikube is running:

```
minikube status
```

If it’s stopped, restart it:

```
minikube start
```

3. If Kubernetes Services Are Not Accessible

Use port forwarding to access a service:

```
kubectl port-forward svc/<service-name> <local-port>:<service-port>
```

Example:

```
kubectl port-forward svc/backend-service 5000:5000
```

Then access the service at:

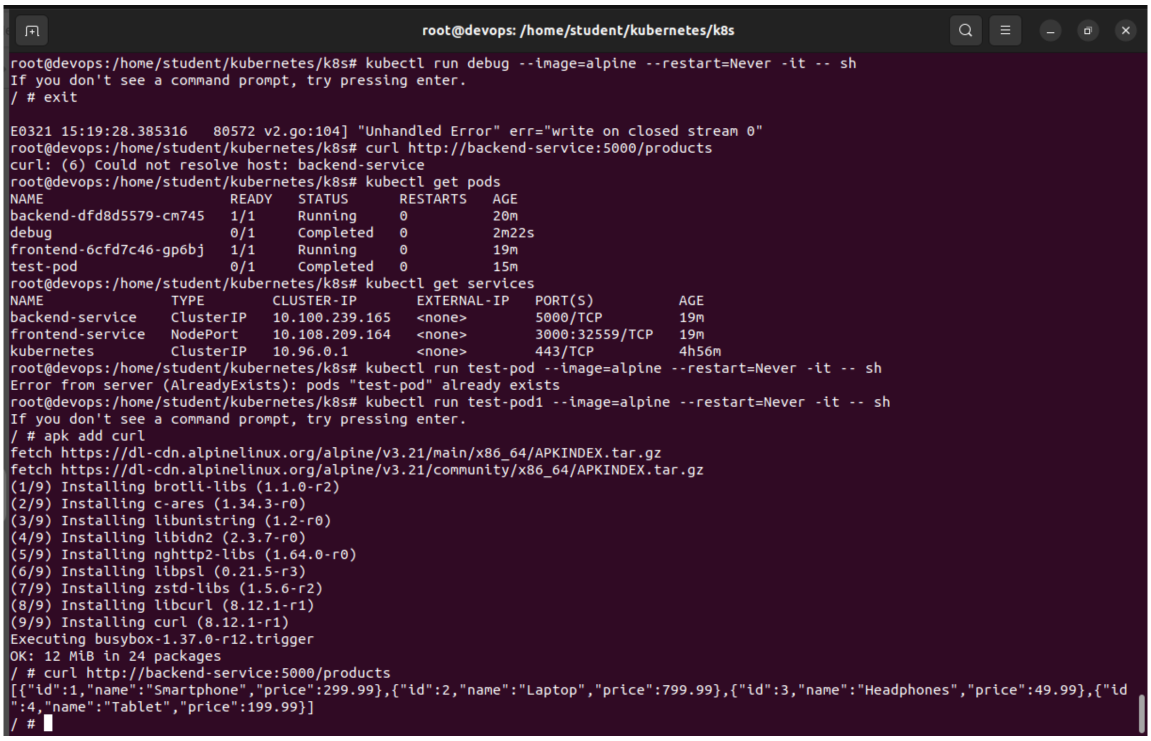
```
http://localhost:5000
```

9. Deploying in Minikube

Deploying the application using Kubernetes and Minikube.

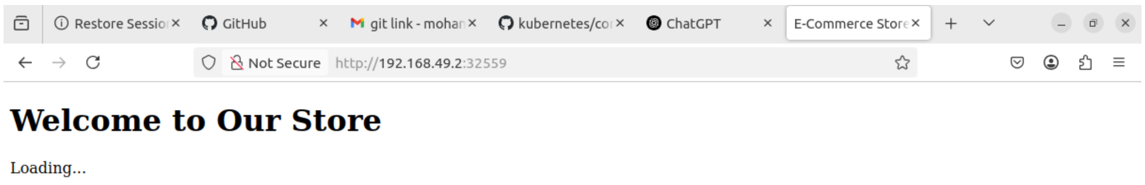
```
minikube start
eval $(minikube docker-env)
kubectl apply -f k8s/
kubectl get pods
kubectl get services
minikube service frontend-service --url
```

Open the displayed URL in a browser to view the application.



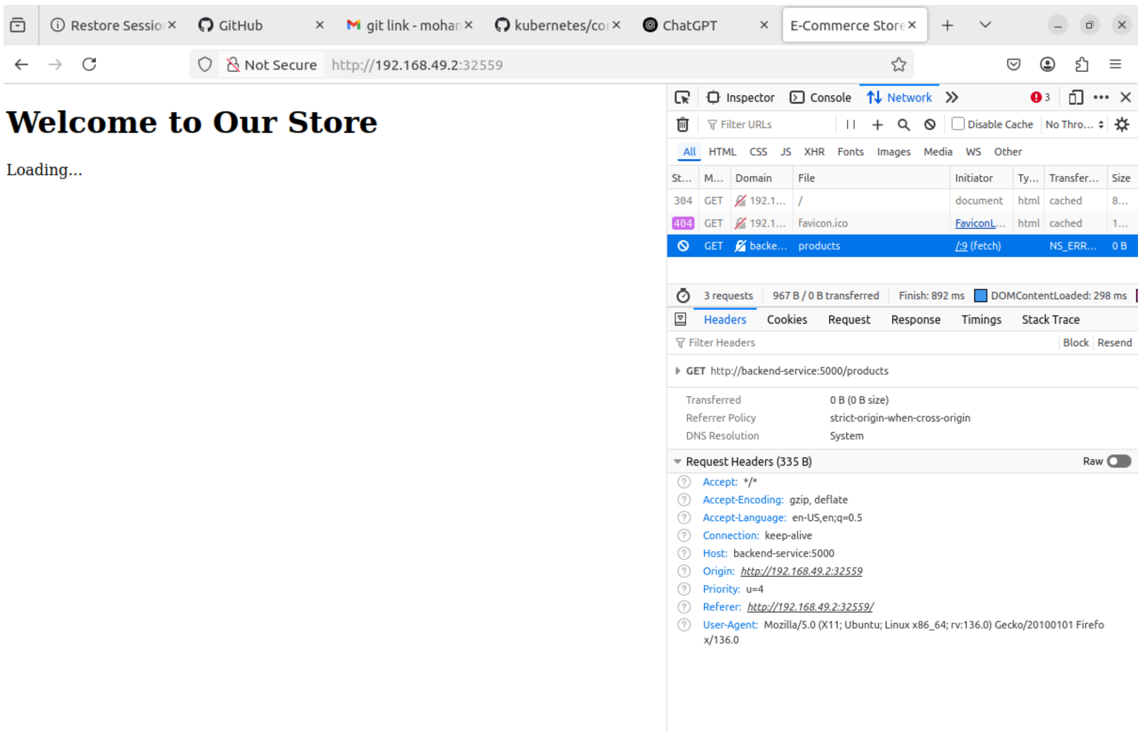
```
root@devops:/home/student/kubernetes/k8s# kubectl run debug --image=alpine --restart=Never -it -- sh
If you don't see a command prompt, try pressing enter.
/ # exit

E0321 15:19:28.385316   80572 v2.go:104] "Unhandled Error" err="write on closed stream 0"
root@devops:/home/student/kubernetes/k8s# curl http://backend-service:5000/products
curl: (6) Could not resolve host: backend-service
root@devops:/home/student/kubernetes/k8s# kubectl get pods
NAME                       READY   STATUS      RESTARTS   AGE
backend-dfd8d5579-cm745    1/1     Running     0          20m
debug                      0/1     Completed   0          2m22s
frontend-6cfd7c46-gp6bj    1/1     Running     0          19m
test-pod                   0/1     Completed   0          15m
root@devops:/home/student/kubernetes/k8s# kubectl get services
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
backend-service    ClusterIP   10.100.239.165   <none>        5000/TCP         19m
frontend-service   NodePort    10.108.209.164   <none>        3000:32559/TCP   19m
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP          4h56m
root@devops:/home/student/kubernetes/k8s# kubectl run test-pod --image=alpine --restart=Never -it -- sh
Error from server (AlreadyExists): pods "test-pod" already exists
root@devops:/home/student/kubernetes/k8s# kubectl run test-pod1 --image=alpine --restart=Never -it -- sh
If you don't see a command prompt, try pressing enter.
/ # apk add curl
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/community/x86_64/APKINDEX.tar.gz
(1/9) Installing brotli-libs (1.1.0-r2)
(2/9) Installing c-ares (1.34.3-r0)
(3/9) Installing libunistring (1.2-r0)
(4/9) Installing libidn2 (2.3.7-r0)
(5/9) Installing nghttp2-libs (1.64.0-r0)
(6/9) Installing libpsl (0.21.5-r3)
(7/9) Installing zstd-libs (1.5.6-r2)
(8/9) Installing libcurl (8.12.1-r1)
(9/9) Installing curl (8.12.1-r1)
Executing busybox-1.37.0-r12.trigger
OK: 12 MiB in 24 packages
/ # curl http://backend-service:5000/products
[{"id":1,"name":"Smartphone","price":299.99},{"id":2,"name":"Laptop","price":799.99},{"id":3,"name":"Headphones","price":49.99},{"id
":4,"name":"Tablet","price":199.99}]
/ # 
```



Welcome to Our Store

Loading...



Welcome to Our Store

Loading...

St…	M…	Domain	File	Initiator	Ty…	Transfer…	Size
304	GET	192.1…	/	document	html	cached	8…
404	GET	192.1…	favicon.ico	FaviconL…	html	cached	1…
	GET	backe…	products	/:9 (fetch)		NS_ERR…	0 B

```
3 requests | 967 B / 0 B transferred | Finish: 892 ms | DOMContentLoaded: 298 ms

GET http://backend-service:5000/products

Transferred        0 B (0 B size)
Referrer Policy    strict-origin-when-cross-origin
DNS Resolution     System

Request Headers (335 B)
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Host: backend-service:5000
Origin: http://192.168.49.2:32559
Priority: u=4
Referer: http://192.168.49.2:32559/
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefo
x/136.0
```