📝 Advanced Lab Task: Multi-Service E-Commerce Simulation

🎯 Objective

Build a **multi-tier e-commerce system** inside Kubernetes with **ClusterIP** and **NodePort** services.

- **Database (MySQL / MongoDB)** – internal only.
- **Product Service (backend)** – ClusterIP, talks to DB.
- **Order Service (backend)** – ClusterIP, talks to DB + Product Service.
- **Frontend (Nginx or React app)** – NodePort, talks to Product + Order services.

⚡ Bonus Challenge: Use **NetworkPolicies** to secure communication.

🔹 Architecture Diagram

```
[ Browser ] → (NodePort) → [ Frontend ] → (ClusterIP) → [ Product Service ]
                                    ↘ (ClusterIP) → [ Order Service ] → (ClusterIP) → [ DB ]
```

🔹 Requirements for Each Component

1. Database

- Use `mysql:5.7` or `mongo:latest` image.

- Set environment variables (`MYSQL_ROOT_PASSWORD`, `MYSQL_DATABASE`, or `MONGO_INITDB_DATABASE`).

- Expose it internally using a **ClusterIP** service.

- Should not be directly reachable from the outside.

- Resource limits: **1 CPU and 1Gi memory**.

K8S > ! db-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: root123
        - name: MYSQL_DATABASE
          value: shopdb
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"

```

K8S > ! db-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: db-service
spec:
  selector:
    app: db
  ports:
    - port: 3306
      targetPort: 3306
  type: ClusterIP

```

---

2. Product Service

- Use a lightweight backend container such as:

  - `hashicorp/http-echo`

  - or a small custom API image if available.

- Add `args` to display a custom message like:

- `"-text=Product Service Connected to DB"`

- Expose it internally using a **ClusterIP** service.

- Must connect to the Database via `db-service`.

- Resource limits: **0.5 CPU and 500Mi memory**.

K8S > ! product-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
  template:
    metadata:
      labels:
        app: product
    spec:
      containers:
      - name: product
        image: hashicorp/http-echo
        args:
          - "-text=Product Service Connected to DB"
        resources:
          limits:
            cpu: "500m"
            memory: "500Mi"
```

K8S > ! product-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: product-service
spec:
  selector:
    app: product
  ports:
    - port: 80
      targetPort: 5678
  type: ClusterIP
```

---

3. Order Service

- Use `hashicorp/http-echo` or another small API image.

- Add `args` to display:

- `"-text=Order Service Connected to Product + DB"`

- Expose it internally using a **ClusterIP** service.

- Must connect to both `db-service` and `product-service`.

- Resource limits: **0.5 CPU and 500Mi memory**.

K8S > ! order-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order
  template:
    metadata:
      labels:
        app: order
    spec:
      containers:
      - name: order
        image: hashicorp/http-echo
        args:
          - "-text=Order Service Connected to Product + DB"
        resources:
          limits:
            cpu: "500m"
            memory: "500Mi"

```

K8S > ! order-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
  selector:
    app: order
  ports:
    - port: 80
      targetPort: 5678
  type: ClusterIP

```

---

4. Frontend

- Use `nginx:latest` or another frontend image.

- Optionally mount a ConfigMap with a custom `index.html` that calls Product and Order services.

- Expose it externally using a **NodePort** service.

- Resource limits: **0.5 CPU and 500Mi memory**.

K8S > ! frontend-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-config
data:
  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
      <title>E-Commerce Frontend</title>
    </head>
    <body>
      <h1>Welcome to E-Commerce Frontend</h1>
      <p><a href="http://product-service">Go to Product Service</a></p>
      <p><a href="http://order-service">Go to Order Service</a></p>
    </body>
    </html>

```

K8S > ! frontend-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        volumeMounts:
        - name: frontend-content
          mountPath: /usr/share/nginx/html/index.html
          subPath: index.html
        resources:
          limits:
            cpu: "500m"
            memory: "500Mi"
      volumes:
      - name: frontend-content
        configMap:
          name: frontend-config

```

K8S > ! frontend-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30080
  type: NodePort

```

---

5. NetworkPolicies (Optional Challenge)

- Allow only Product and Order services to talk to the Database.

- Deny direct DB access from the Frontend.

- Allow the Frontend to reach only Product and Order services.

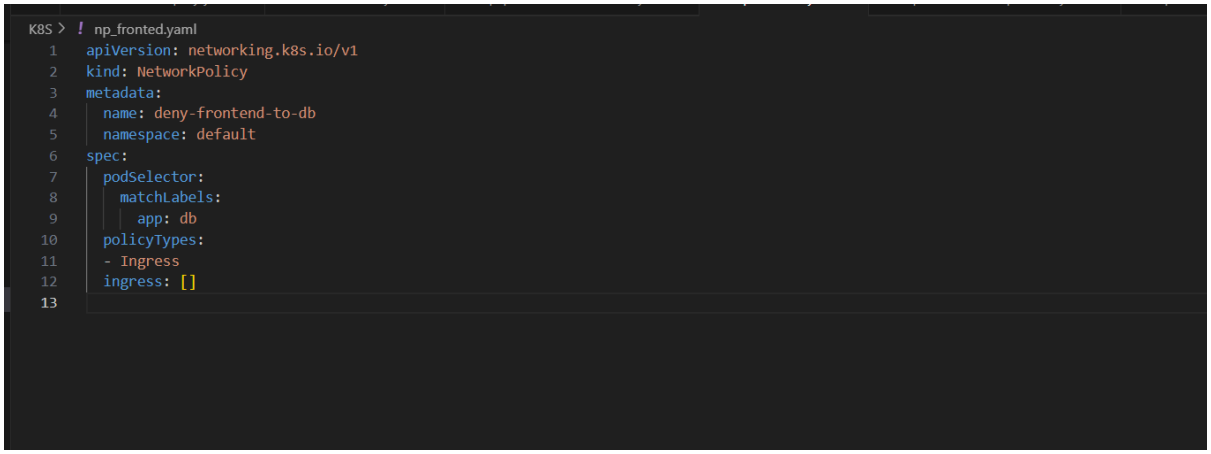
K8S > ! np-product-order-to-db.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-product-order-to-db
spec:
  podSelector:
    matchLabels:
      app: database
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: product
    - podSelector:
        matchLabels:
          app: order
```

K8S > ! np-frontend-to-product.yaml

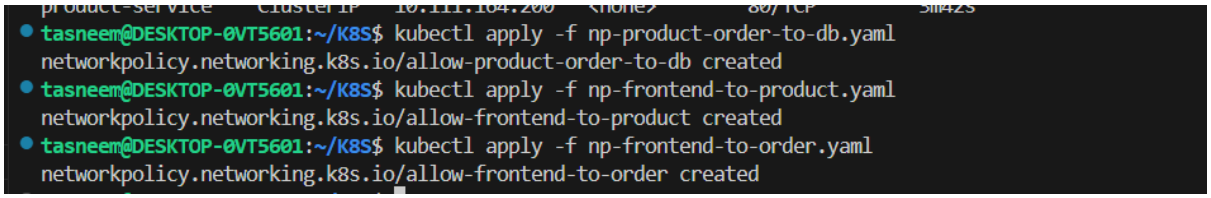
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-product
spec:
  podSelector:
    matchLabels:
      app: product
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-product-order-to-db
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: product
    - podSelector:
        matchLabels:
          app: order

```



K8S > ! np_fronted.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-frontend-to-db
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
  - Ingress
  ingress: []

```



```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl apply -f np-product-order-to-db.yaml
networkpolicy.networking.k8s.io/allow-product-order-to-db created
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl apply -f np-frontend-to-product.yaml
networkpolicy.networking.k8s.io/allow-frontend-to-product created
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl apply -f np-frontend-to-order.yaml
networkpolicy.networking.k8s.io/allow-frontend-to-order created
```

🔹 Tasks for Trainee

Arch communication:

- Product → DB ✅

- Order → DB + Product ✅

- Frontend → Product + Order ✅

- Frontend → DB ❌ (should fail after NetworkPolicy).

1. Write the YAML manifests for Database, Product, Order, and Frontend (with correct **image** and **args**).

2. Deploy them into the cluster.

3. Verify that all pods and services are running correctly.

4. Verify that the **Product service can connect to the Database**.

5. Verify that the **Order service can connect to both the Database and Product service**.

6. Verify that the **Frontend can connect to the Product and Order services**.

7. Verify that the **Frontend cannot connect to the Database** once you apply NetworkPolicies.

8. Verify that the **Frontend is reachable externally through NodePort** using the Minikube IP.

9. Document your results (which checks succeeded and which failed after applying policies).

🔹 trainee Instructions

1. Write the YAML manifests for Database, Product, Order, and Frontend (with correct **image** and **args**).

2. Deploy them into the cluster.

3. Verify that all pods and services are running correctly.

- **Provide a screenshot** of the output of `kubectl get pods` and `kubectl get svc`.

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db-service        ClusterIP   10.99.102.8      <none>        3306/TCP       23m
frontend-service  NodePort    10.102.171.123   <none>        80:30080/TCP   18m
kubernetes        ClusterIP   10.96.0.1        <none>        443/TCP        3d23h
my-service        ClusterIP   10.106.6.237     <none>        80/TCP         3d8h
order-service     ClusterIP   10.98.54.13      <none>        80/TCP         20m
product-service   ClusterIP   10.111.164.200   <none>        80/TCP         21m
```

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl get pods
NAME                                READY   STATUS    RESTARTS        AGE
db-7b6b59699-7k9dj                  1/1     Running   1 (15m ago)     27m
deployment-1-858c547685-2p5vx       1/1     Running   12 (9m49s ago)  3d21h
deployment-1-858c547685-ckgxz       1/1     Running   12 (18m ago)    3d21h
frontend-76f668cb49-ggrll           1/1     Running   0               19m
httpd-frontend-65c89bbdc5-5r6jb     1/1     Running   12 (18m ago)    3d21h
httpd-frontend-65c89bbdc5-k4sxw     1/1     Running   12 (18m ago)    3d21h
httpd-frontend-65c89bbdc5-ms85p     1/1     Running   12 (18m ago)    3d21h
my-nginx                            1/1     Running   1 (76m ago)     3d22h
my-nginx-rs-d78qb                   1/1     Running   1 (76m ago)     3d22h
order-7dc5cd4dbc-2ptmp              1/1     Running   0               21m
product-9cbd4fdc4-5pj8m             1/1     Running   0               26m
web-deploy-7dcdb65bff-2xhmw         1/1     Running   1 (76m ago)     3d9h
web-deploy-7dcdb65bff-dqfct         1/1     Running   1 (76m ago)     3d9h
web-deploy-7dcdb65bff-k6nm4         1/1     Running   1 (76m ago)     3d9h
```

4. Verify that the **Product service can connect to the Database**.

- **Provide a screenshot** of the command line showing this test.

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl run test-pod -it --image=alpine -- sh
If you don't see a command prompt, try pressing enter.
/ # apk add --no-cache curl bind-tools
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/community/x86_64/APKINDEX.tar.gz
(1/23) Installing fstrm (0.6.1-r4)
(2/23) Installing krb5-conf (1.0-r2)
(3/23) Installing libcom_err (1.47.2-r2)
(4/23) Installing keyutils-libs (1.6.3-r4)
(5/23) Installing libverto (0.3.2-r2)
(6/23) Installing krb5-libs (1.21.3-r0)
(7/23) Installing json-c (0.18-r1)
(8/23) Installing nghttp2-libs (1.65.0-r0)
(9/23) Installing protobuf-c (1.5.2-r0)
(10/23) Installing userspace-rcu (0.15.2-r0)
(11/23) Installing libuv (1.51.0-r0)
(12/23) Installing xz-libs (5.8.1-r0)
(13/23) Installing libxml2 (2.13.8-r0)
(14/23) Installing bind-libs (9.20.12-r0)
(15/23) Installing bind-tools (9.20.12-r0)
```

```
/ # curl http://product-service:80
Product Service Connected to DB
/ #
```

5. Verify that the **Order service can connect to both the Database and Product service**.

- **Provide a screenshot** of the command line showing this test.

```
/ # apk add --no-cache mysql-client
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/community/x86_64/APKINDEX.tar.gz
(1/7) Installing mariadb-common (11.4.8-r0)
(2/7) Installing libgcc (14.2.0-r6)
(3/7) Installing ncurses-terminfo-base (6.5_p20250503-r0)
(4/7) Installing libncursesw (6.5_p20250503-r0)
(5/7) Installing libstdc++ (14.2.0-r6)
(6/7) Installing mariadb-client (11.4.8-r0)
(7/7) Installing mysql-client (11.4.8-r0)
Executing busybox-1.37.0-r18.trigger
OK: 49 MiB in 23 packages
```

```
Bye
/ # curl http://product-service:80

Product Service Connected to DB
/ #
/ # curl http://order-service:80
Order Service Connected to Product + DB
/ #
```


6. Verify that the **Frontend can connect to the Product and Order services**.

- **Provide a screenshot** of the command line showing this test.


```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl exec -it frontend-76f668cb49-ggrll  -- /bin/sh
# apk add --no-cache curl
/bin/sh: 1: apk: not found
# curl http://product-service:80
Product Service Connected to DB
# curl http://order-service:80
Order Service Connected to Product + DB
#
```

7. Verify that the **Frontend cannot connect to the Database** after applying NetworkPolicies.

- **Provide a screenshot** of the failed test.

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl exec -it frontend-76f668cb49-ggrll  -- /bin/sh
# mysql -h db-service -u root -p --ssl=0
/bin/sh: 1: mysql: not found
# apt update && apt install -y mysql-client
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8793 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [6924 B]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [277 kB]
Fetched 9331 kB in 24s (392 kB/s)
Reading package lists... Done
```

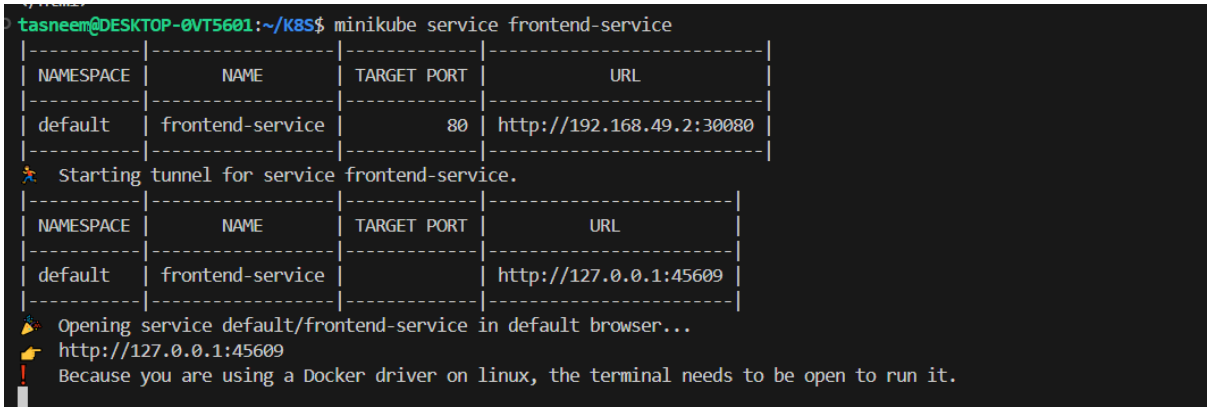
```
# apt install -y default-mysql-client
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libconfig-inifiles-perl libdbd-mariadb-perl libdbi-perl libgdbm-compat4 libgdbm6 libgpm2 libmariadb3 libn
  mariadb-client-core mariadb-common mysql-common netbase perl perl-modules-5.36
Suggested packages:
  libclone-perl libmldbm-perl libnet-daemon-perl libsql-statement-perl gdbm-l10n gpm sensible-utils perl-do
  libtap-harness-archive-perl
The following NEW packages will be installed:
  default-mysql-client libconfig-inifiles-perl libdbd-mariadb-perl libdbi-perl libgdbm-compat4 libgdbm6 libg
  mariadb-client mariadb-client-core mariadb-common mysql-common netbase perl perl-modules-5.36
0 upgraded, 18 newly installed, 0 to remove and 2 not upgraded.
Need to get 12.6 MB of archives.
After this operation, 133 MB of additional disk space will be used.
```

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl exec -it frontend-76f668cb49-ggrll  -- /bin/sh
# mysql -h db-service -u root -p --ssl=0
Enter password:
ERROR 2002 (HY000): Can't connect to server on 'db-service' (115)
# exit
command terminated with exit code 1
```

8. Verify that the **Frontend is reachable externally through NodePort** using the Minikube IP.

- **Provide a screenshot** of your browser or command line showing successful access.

```
tasneem@DESKTOP-0VT5601:~/K8S$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db-service        ClusterIP   10.99.102.8      <none>        3306/TCP       14h
frontend-service  NodePort    10.102.171.123   <none>        80:30080/TCP   13h
kubernetes        ClusterIP   10.96.0.1        <none>        443/TCP        4d12h
my-service        ClusterIP   10.106.6.237     <none>        80/TCP         3d22h
order-service     ClusterIP   10.98.54.13      <none>        80/TCP         13h
product-service   ClusterIP   10.111.164.200   <none>        80/TCP         13h
tasneem@DESKTOP-0VT5601:~/K8S$ minikube ip
!  Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 9.092302072s
💡  Restarting the docker service may improve performance.
192.168.49.2
tasneem@DESKTOP-0VT5601:~/K8S$ curl http://192.168.49.2:30080
<!DOCTYPE html>
<html>
<head>
  <title>E-Commerce Frontend</title>
</head>
<body>
  <h1>Welcome to E-Commerce Frontend</h1>
  <p><a href="http://product-service">Go to Product Service</a></p>
  <p><a href="http://order-service">Go to Order Service</a></p>
</body>
</html>
```



9. Submit all screenshots along with your YAML manifests as proof of your work.