Docker Lab 7 — Service Mesh Concepts with Docker (Envoy / Traefik)

Author: Dr. Sandeep Kumar Sharma

Lab Description

Lab 7 introduces **service mesh concepts in a Docker environment**, using lightweight proxies like **Envoy** or **Traefik**. While service mesh is widely adopted in Kubernetes, the same ideas can be demonstrated in Docker to understand traffic shaping, routing, observability, and mTLS.

This lab walks through deploying two microservices (`sandeep-app1` and `sandeep-app2`) and placing Envoy/Traefik as a sidecar or gateway.

Topics Covered

- What is a service mesh?
- Sidecar proxy pattern
- Traffic routing between microservices
- L7 load balancing and retries
- Distributed tracing and logging
- Deploying Envoy/Traefik in Docker Compose

Learning Objectives

- Understand how service mesh improves microservice communication
- Deploy a reverse proxy/service mesh gateway in Docker
- Configure routing, retries, and load balancing rules
- Perform observability and logging checks

Learning Outcomes

- Ability to model service mesh concepts without Kubernetes
- Ability to set up distributed microservice traffic routing
- Understand production-grade networking patterns

Section 1 — Create Project Structure

```
mkdir ~/docker-service-mesh-lab
cd ~/docker-service-mesh-lab
```

Create directories:

```
mkdir app1 app2 envoy
```

Section 2 — Create Two Simple Nginx-Based Services

app1 content:

```
echo "App 1 Response from Sandeep" > app1/index.html
```

app2 content:

```
echo "App 2 Response from Sandeep" > app2/index.html
```

Section 3 — Create Envoy Configuration

```
nano envoy/envoy.yaml
```

Paste:

```
static_resources:
  listeners:
  - name: listener_0
    address:
      socket_address: { address: 0.0.0.0, port_value: 8080 }
    filter_chains:
    - filters:
      - name: envoy.filters.network.http_connection_manager
```

```
          typed_config:
            "@type": type.googleapis.com/
envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
            stat_prefix: ingress_http
            route_config:
              name: local_route
              virtual_hosts:
              - name: app
                domains: ["*"]
                routes:
                - match: { prefix: "/app1" }
                  route: { cluster: app1 }
                - match: { prefix: "/app2" }
                  route: { cluster: app2 }
            http_filters:
            - name: envoy.filters.http.router
  clusters:
  - name: app1
    connect_timeout: 0.25s
    type: logical_dns
    lb_policy: round_robin
    load_assignment:
      cluster_name: app1
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address: { address: sandeep-app1, port_value: 80 }

  - name: app2
    connect_timeout: 0.25s
    type: logical_dns
    lb_policy: round_robin
    load_assignment:
      cluster_name: app2
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address: { address: sandeep-app2, port_value: 80 }
```

Section 4 — Create Docker Compose File

```
nano docker-compose.yml
```

Paste:

```
version: '3.8'
services:
  sandeep-app1:
    image: nginx
    volumes:
      - ./app1:/usr/share/nginx/html
    networks:
      - sandeep-mesh

  sandeep-app2:
    image: nginx
    volumes:
      - ./app2:/usr/share/nginx/html
    networks:
      - sandeep-mesh

  envoy:
    image: envoyproxy/envoy:v1.28-latest
    volumes:
      - ./envoy/envoy.yaml:/etc/envoy/envoy.yaml
    networks:
      - sandeep-mesh
    ports:
      - "8080:8080"
    depends_on:
      - sandeep-app1
      - sandeep-app2

networks:
  sandeep-mesh:
    driver: bridge
```

Section 5 — Deploy the Service Mesh

```
docker compose up -d
```

Check running services:

```
docker ps
```

Section 6 — Test Routing via Envoy Gateway

Access service 1:

```
curl http://localhost:8080/app1
```

Access service 2:

```
curl http://localhost:8080/app2
```

Responses should come from Nginx containers.

Section 7 — Add Load Balancing & Retry Policies

Modify `envoy.yaml` to include:

```
retry_policy:
  retry_on: "5xx"
  num_retries: 3
```

Reload Envoy:

```
docker compose restart envoy
```

Section 8 — Cleanup

```
docker compose down
```

Summary

Lab 7 demonstrated service mesh concepts such as routing, load balancing, and sidecar proxying using Envoy inside a Docker environment. This gives a strong foundation before moving toward Kubernetes service mesh implementations.