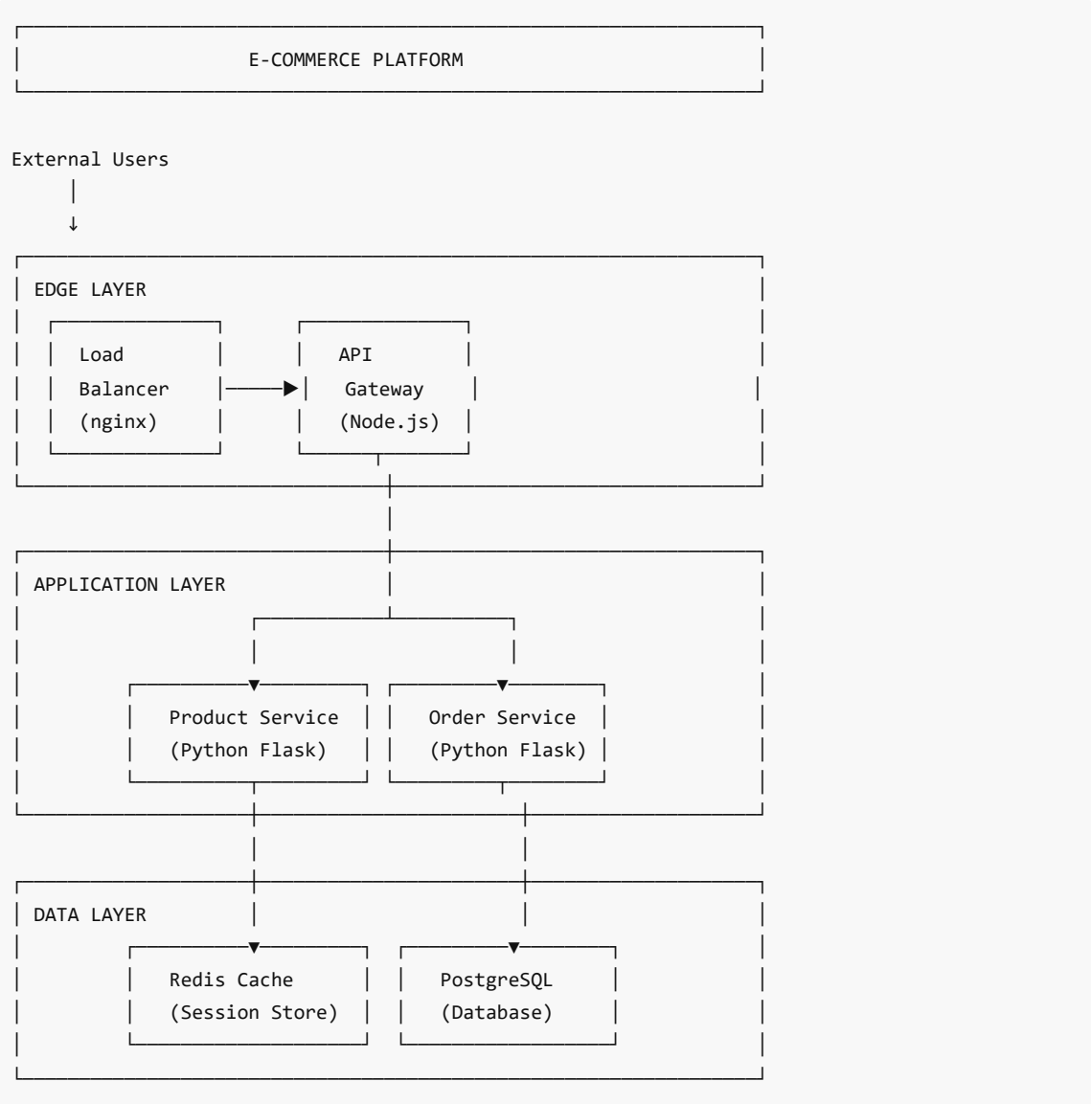
Container Networking Project

Project Overview

This project involves building a complete containerized microservices application infrastructure using only Linux primitives (network namespaces, veth pairs, bridges, iptables) to understand the low-level workings of container networking. The infrastructure simulates a real-world e-commerce platform with multiple services.

System Architecture



Prerequisites

- Linux environment (WSL2 or Linux VM)
- Root/Sudo privileges
- Python 3
- `iproute2` (for `ip` command)

- `iptables`
- `curl`

1: Foundation - Linux Primitives

Goals

- Set up isolated network namespaces
- Create virtual network interfaces
- Implement basic inter-namespace communication

Tasks

Task 1.1: Create Network Namespaces

Create six network namespaces representing my services:

```
# Create namespaces
sudo ip netns add nginx-lb
sudo ip netns add api-gateway
sudo ip netns add product-service
sudo ip netns add order-service
sudo ip netns add redis-cache
sudo ip netns add postgres-db
```

Deliverable: Screenshot showing all namespaces created

Run: `ip netns list`

Task 1.2: Build a Virtual Bridge Network

Create a bridge to connect all services:

```
# Create bridge
sudo ip link add br-app type bridge
```

```
sudo ip addr add 10.0.0.1/16 dev br-app
sudo ip link set br-app up
```

Connect each namespace to the bridge using veth pairs:

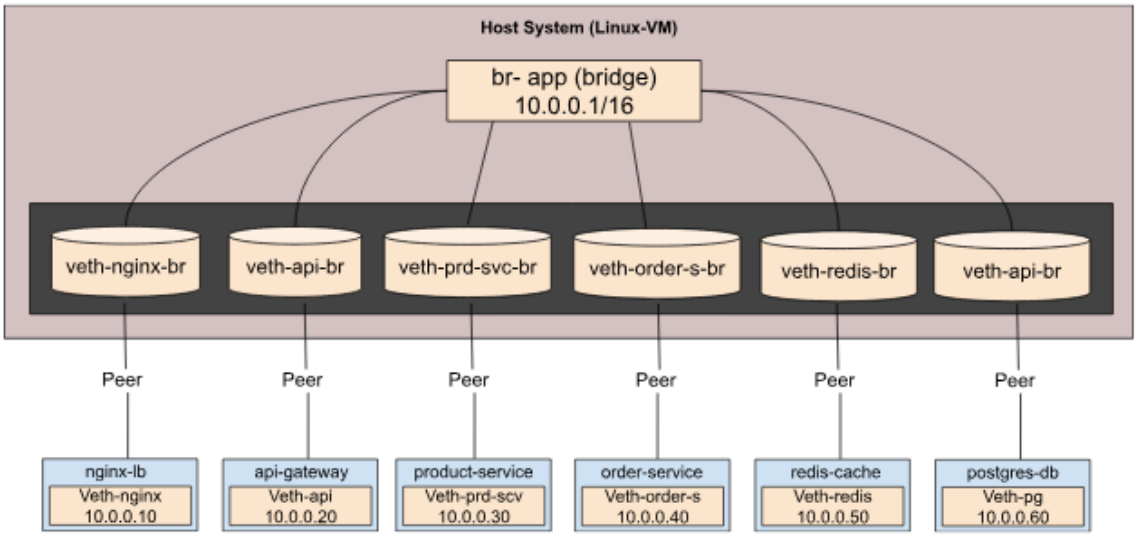
```
# Example for nginx-lb (repeat for all services)
sudo ip link add veth-nginx type veth peer name veth-nginx-br
sudo ip link set veth-nginx netns nginx-lb
sudo ip link set veth-nginx-br master br-app
sudo ip link set veth-nginx-br up
```

Configure inside namespace:

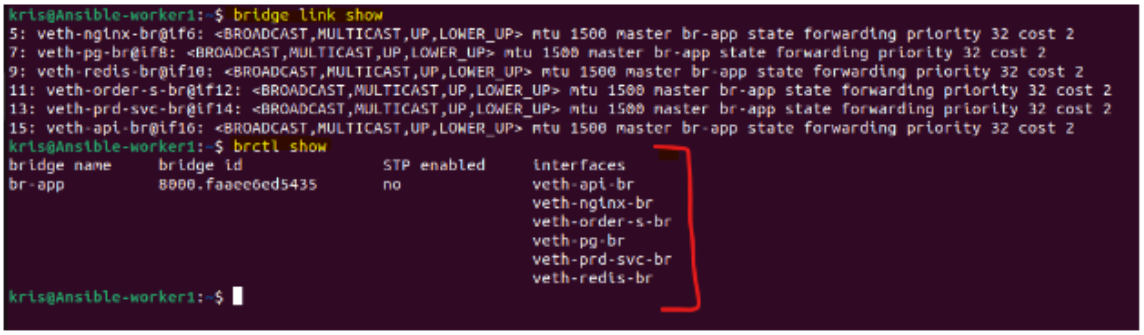
```
sudo ip netns exec nginx-lb ip addr add 10.0.0.10/16 dev veth-nginx
sudo ip netns exec nginx-lb ip link set veth-nginx up
sudo ip netns exec nginx-lb ip link set lo up
sudo ip netns exec nginx-lb ip route add default via 10.0.0.1

IP:
- nginx-lb: 10.0.0.10
- api-gateway: 10.0.0.20
- product-service: 10.0.0.30
- order-service: 10.0.0.40
- redis-cache: 10.0.0.50
- postgres-db: 10.0.0.60
```

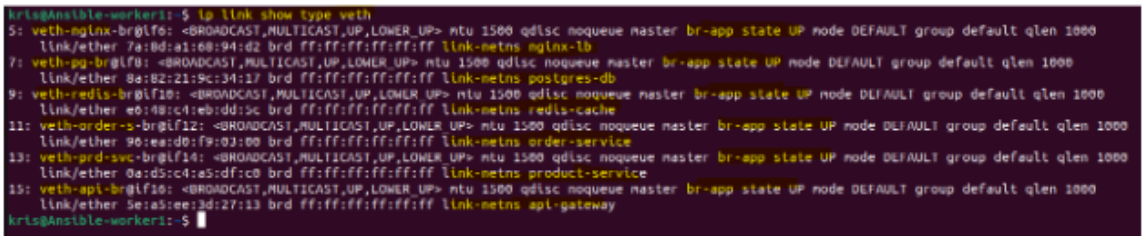
Deliverable:

- **Network diagram showing my setup**



- **Showing bridge with connected interfaces**



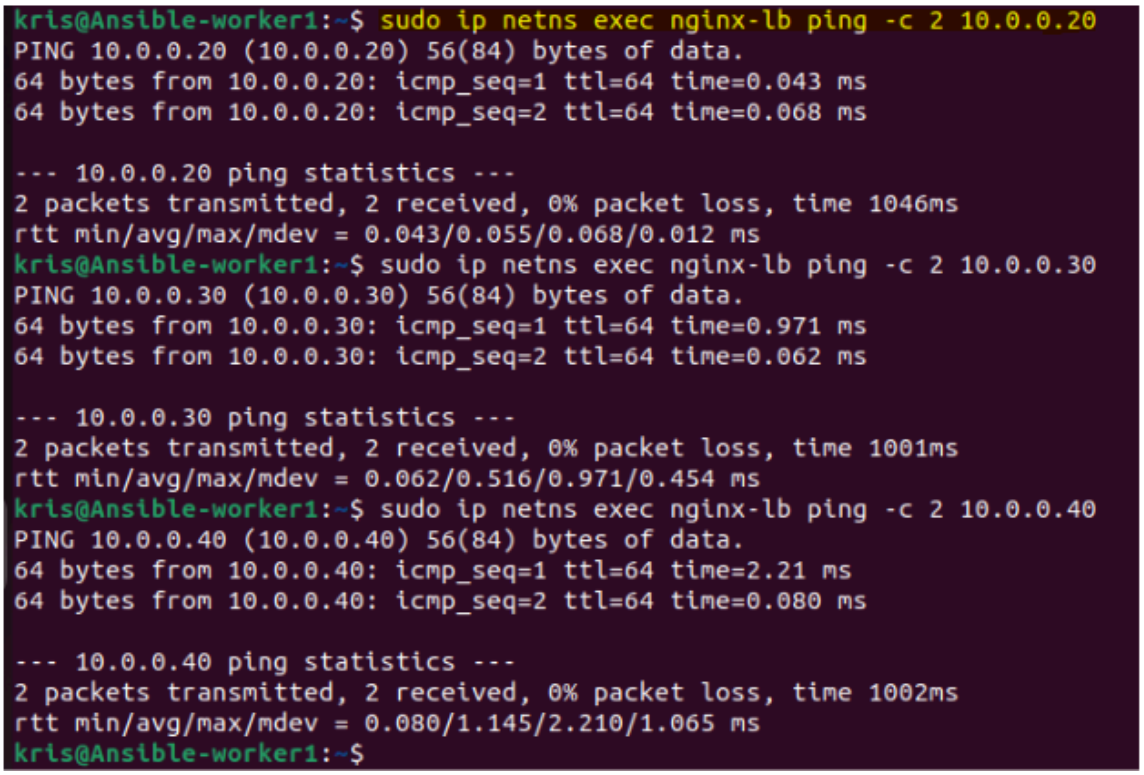


- **Proof of connectivity (ping tests between all namespaces)**

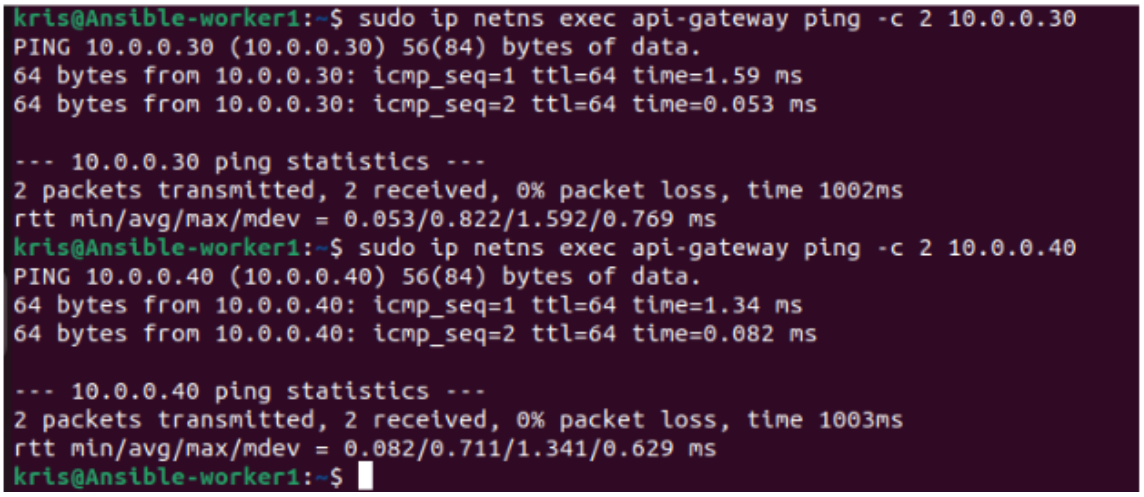
Run : `sudo ip netns exec nginx-lb ping -c 2 10.0.0.20`

nginx-lb –-> api-gateway (10.0.0.20)

nginx-lb –-> product-service(10.0.0.30)



api-gateway –-> product-service (10.0.0.30)



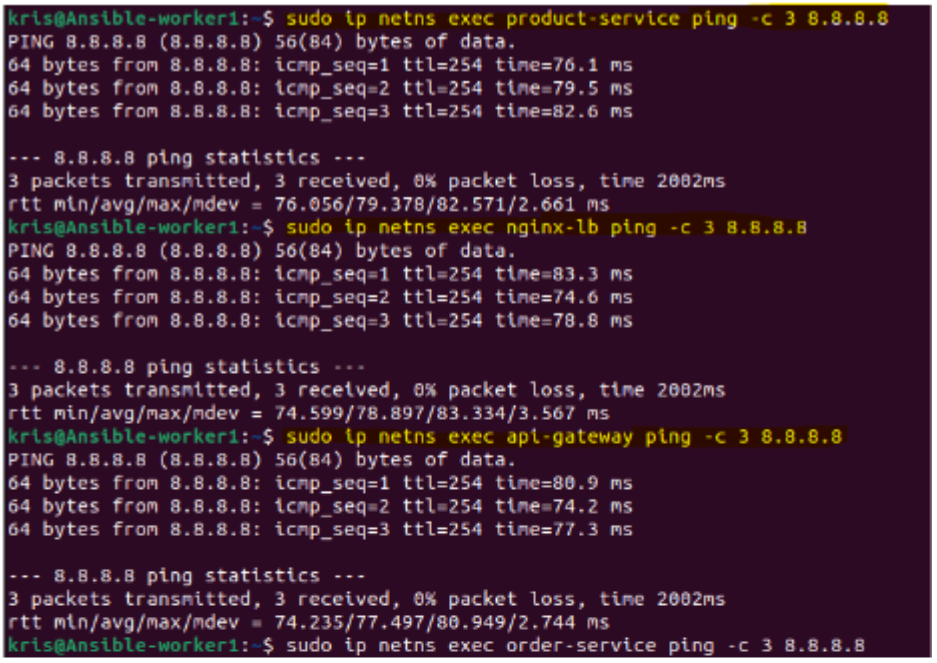
Task 1.3: Implement NAT for Internet Access

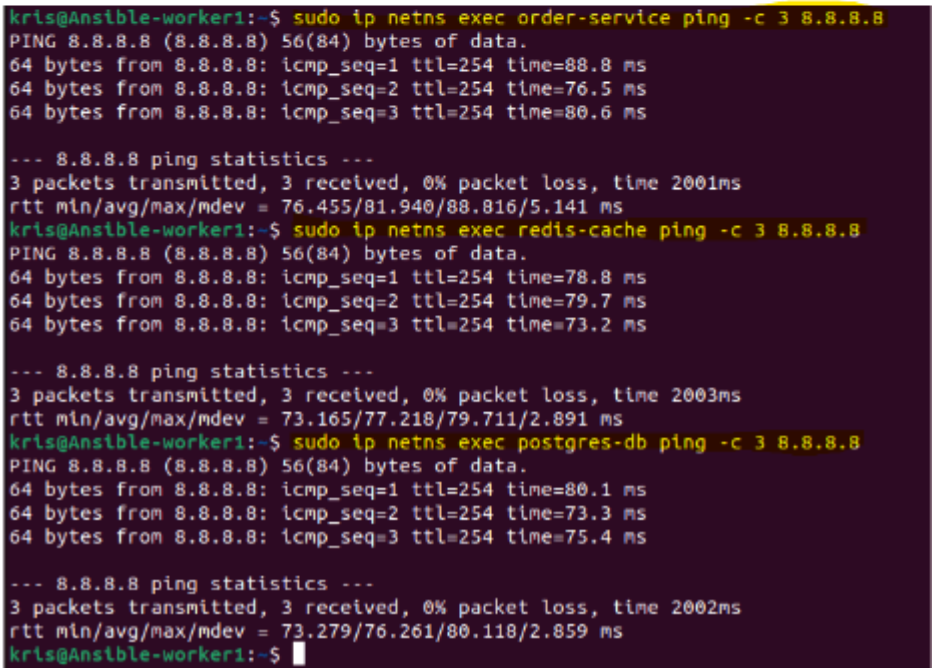
Enable internet access for all namespaces:

```
# Enable IP forwarding
sudo sysctl -w net.ipv4.ip_forward=1

# Add MASQUERADE rule
sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/16 ! -o br-app -j MASQUERADE
```

Deliverable: Test internet connectivity from each namespace

RUN: `sudo ip netns exec product-service ping -c 3 8.8.8.8`





Task 1.4: Setup Port Forwarding

Forward host port 8080 to nginx-lb:

```
# Add DNAT rule (PREROUTING - before routing decision)
sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to-destination 10.0.0.10:80

# Allow forwarding in FILTER table
sudo iptables -A FORWARD -p tcp -d 10.0.0.10 --dport 80 -j ACCEPT
```

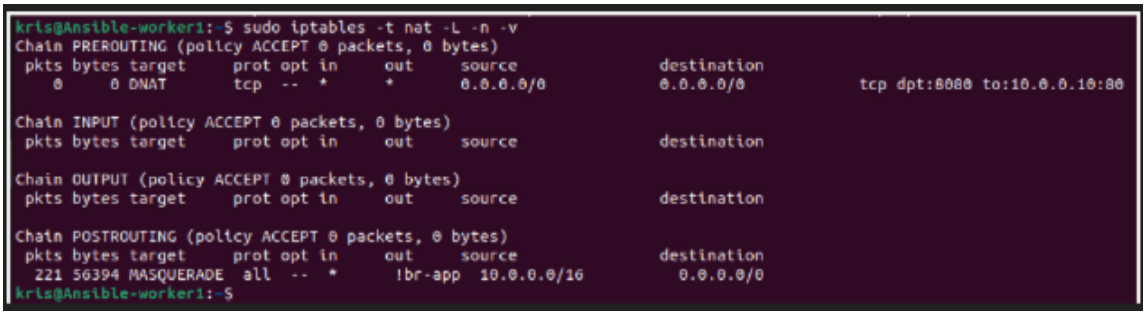
Now external clients can access via host IP on port 8080

Test from host (simulating external client) to confirm request is forwarded to nginx-lb

RUN: `curl http://localhost:8080`

View the NAT rule:

RUN: `sudo iptables -t nat -L -v -n`



Deliverable: Document all iptables rules with explanations

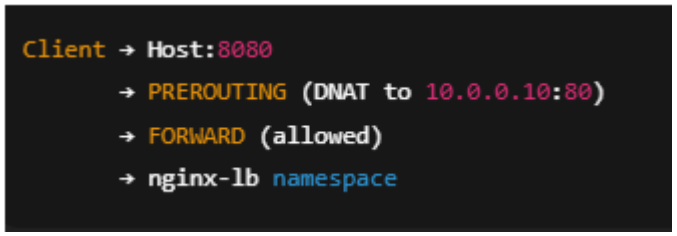
Explanation of the rule:

```
Rule:

1. sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/16 ! -o br-app -j MASQUERADE

What this rule does:
  - Allows containers/namespaces to access the internet
  - Hides internal IP addresses behind the host IP
  - Required for outbound connectivity

2. sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to-destination
```

```
10.0.0.10:80

What this rule does:
  - Redirects traffic from host port 8080
  - Sends it to nginx running inside the namespace on port 80
  - Enables external access to internal services

3. sudo iptables -A FORWARD -p tcp -d 10.0.0.10 --dport 80 -j ACCEPT

What this rule does:
  - Explicitly allows forwarded traffic to reach nginx-lb
  - Prevents Linux from dropping forwarded packets
  - Required for DNAT port forwarding to function
```

Traffic Flow Summary:

External Client (Port 8080) -> Host (Port 8080) -> DNAT -> nginx-lb (Port 80)



Option / Flag	Meaning	Explanation
`-t nat`	NAT table	Specifies that the rule applies to the NAT table
`-A POSTROUTING`	POSTROUTING chain	Appends rule to POSTROUTING chain (applied **after** routing decision)
`-A PREROUTING`	PREROUTING chain	Appends rule to PREROUTING chain (applied **before** routing decision)
`-A FORWARD`	FORWARD chain	Appends rule to FORWARD chain (controls forwarded traffic)
`-s 10.0.0.0/16`	Source network	Matches packets originating from the container network
`! -o br-app`	Not bridge interface	Matches packets **not** leaving via the bridge interface (i.e., traffic going to the internet)
`-p tcp`	TCP protocol	Matches TCP traffic
`--dport 8080`	Destination port	Matches packets destined to port **8080** on the host
`--dport 80`	Destination port	Matches packets destined to port **80**
`-d 10.0.0.10`	Destination IP	Matches destination IP address of the nginx-lb namespace
`-j MASQUERADE`	SNAT action	Applies MASQUERADE (dynamic Source NAT), rewriting source IP to host's external IP

`-j DNAT`	DNAT action	Applies Destination NAT, rewriting destination IP and/or port
`--to-destination 10.0.0.10:80`	DNAT target	Forwards traffic to nginx-lb namespace at IP `10.0.0.10` on port `80`
`-j ACCEPT`	Accept action	Allows the packet to be forwarded

2: Application Services

Goals

- Deploy actual services in namespaces
- Implement service-to-service communication
- Test the complete application flow

Tasks

Task 2.1: Deploy Nginx Load Balancer

Create a simple nginx configuration that load balances to the API gateway:

Install and run nginx in the namespace:

```
# Create nginx config

sudo ip netns exec nginx-lb bash -c 'cat <<EOF > /tmp/nginx/nginx.conf
events {
    worker_connections 1024;
}

http {
    upstream api_gateway {
        server 10.0.0.20:3000;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://api_gateway;
            proxy_set_header Host \$host;
            proxy_set_header X-Real-IP \$remote_addr;
        }

        location /health {
            return 200 "OK\n";
            add_header Content-Type text/plain;
        }
    }
}
EOF'
```

Verify the nginx config file has been created in nginx-lb namespace

RUN: `sudo ip netns exec nginx-lb cat /tmp/nginx/nginx.conf`

Start nginx inside nginx-lb namespace

RUN: `sudo ip netns exec nginx-lb nginx -c /tmp/nginx/nginx.conf`

Deliverable: Working load balancer responding to HTTP requests

Test from host:

RUN: `curl http://192.168.56.104:8080/health`

```
kris@Ansible-worker1:~$ curl http://192.168.56.104:8080/health
OK
kris@Ansible-worker1:~$
```

Test from another namespace:

RUN: `sudo ip netns exec api-gateway curl http://10.0.0.10/health`

```
kris@Ansible-worker1:~$ sudo ip netns exec api-gateway curl http://10.0.0.10/health
OK
kris@Ansible-worker1:~$ sudo ip netns exec redis-cache curl http://10.0.0.10/health
OK
kris@Ansible-worker1:~$ sudo ip netns exec postgres-db curl http://10.0.0.10/health
OK
kris@Ansible-worker1:~$ sudo ip netns exec product-service curl http://10.0.0.10/health
OK
kris@Ansible-worker1:~$ sudo ip netns exec order-service curl http://10.0.0.10/health
OK
kris@Ansible-worker1:~$
```

Task 2.2: Create API Gateway

Build a Node.js or Python API gateway that routes to backend services.

create api-gateway.py:

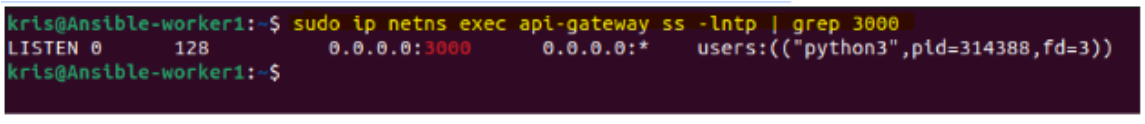
```
from flask import Flask, jsonify, request
import requests

app = Flask(__name__)

PRODUCT_SERVICE = "http://10.0.0.30:5000"
ORDER_SERVICE = "http://10.0.0.40:5000"

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "service": "api-gateway"})

@app.route('/api/products', methods=['GET'])
def get_products():
    try:
        response = requests.get(f"{PRODUCT_SERVICE}/products")
```

```
        return jsonify(response.json()), response.status_code
    except Exception as e:
        return jsonify({"error": str(e)}), 503

@app.route('/api/products/<id>', methods=['GET'])
def get_product(id):
    try:
        response = requests.get(f"{PRODUCT_SERVICE}/products/{id}")
        return jsonify(response.json()), response.status_code
    except Exception as e:
        return jsonify({"error": str(e)}), 503

@app.route('/api/orders', methods=['POST'])
def create_order():
    try:
        response = requests.post(
            f"{ORDER_SERVICE}/orders",
            json=request.json
        )
        return jsonify(response.json()), response.status_code
    except Exception as e:
        return jsonify({"error": str(e)}), 503

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=3000)
```

```
Copy file to accessible location

sudo ip netns exec api-gateway bash -c 'cat <<EOF > /tmp/api-gateway.py
EOF'

# Install dependencies in namespace or use a Python virtual environment
sudo ip netns exec api-gateway pip install flask requests

# Start api-gateway
sudo ip netns exec api-gateway python3 /tmp/api-gateway.py &
```

Flask running inside the api-gateway namespace on port 3000:

RUN: `sudo ip netns exec api-gateway ss -lntp | grep 3000`



Testing the Api health from inside the namespace:

RUN: `sudo ip netns exec api-gateway curl http://127:3000/health`

```
kris@Ansible-worker1:~$ sudo ip netns exec api-gateway curl http://127.0.0.1:3000/health
127.0.0.1 - - [26/Dec/2025 04:47:01] "GET /health HTTP/1.1" 200 -
{"service":"api-gateway","status":"healthy"}
kris@Ansible-worker1:~$
```

Test from other namespaces (e.g., nginx-lb)

RUN: `sudo ip netns exec nginx-lb curl http://10.0.0.20:3000/health`

```
kris@Ansible-worker1:~$ sudo ip netns exec nginx-lb curl http://10.0.0.20:3000/health
10.0.0.10 - - [26/Dec/2025 05:13:41] "GET /health HTTP/1.1" 200 -
{"service":"api-gateway","status":"healthy"}
kris@Ansible-worker1:~$
```

Deliverable: API Gateway responding to requests and routing correctly

- **Test API Gateway Routing to product-service from api-gateway namespace:**

  RUN: `sudo ip netns exec api-gateway curl http://localhost:3000/api/products`

```
kris@Ansible-worker1:~$ sudo ip netns exec api-gateway curl http://localhost:3000/api/products
[{"id":"1","name":"Laptop","price":999.99,"stock":50},{"id":"2","name":"Mouse","price":29.99,"stock":200},{"id":"3","name":"Keyboard","price":
79.99,"stock":150}]
kris@Ansible-worker1:~$
```

- **Then test API Gateway Routing to product service via my host port-forwarding to confirm everything works end-to-end:**

  RUN: `curl http://192.168.56.104:8080/api/products`

```
kris@Ansible-worker1:~$ curl http://192.168.56.104:8080/api/products
[{"id":"1","name":"Laptop","price":999.99,"stock":50},{"id":"2","name":"Mouse","price":29.99,"stock":200},{"id":"3","name":"Keyboard","price":
79.99,"stock":150}]
kris@Ansible-worker1:~$
```

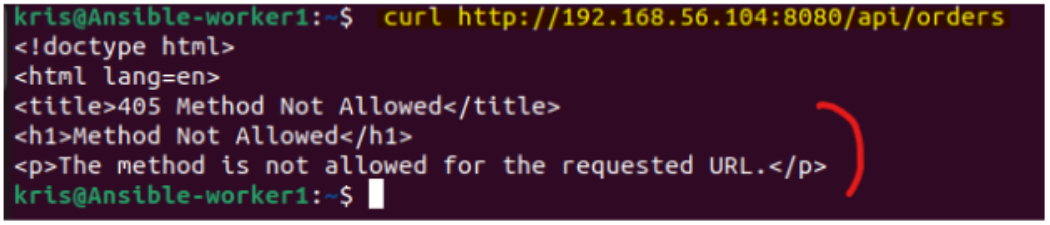
- **Test API Gateway Routing to order-service from api-gateway namespace:**

  RUN: `sudo ip netns exec api-gateway curl http://localhost:3000/api/orders`

```
kris@Ansible-worker1:~$ sudo ip netns exec api-gateway curl http://localhost:3000/api/orders
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
kris@Ansible-worker1:~$
```

- **Then test API Gateway Routing to order service via my host port-forwarding to confirm everything works end-to-end:**

  RUN: `curl http://192.168.56.104:8080/api/orders`



Task 2.3: Build Product Service

Create product-service.py:

```
from flask import Flask, jsonify
import redis
import json

app = Flask(__name__)

# Connect to Redis cache
try:
    cache = redis.Redis(host='10.0.0.50', port=6379, decode_responses=True)
except:
    cache = None

# Mock product database
PRODUCTS = {
    "1": {"id": "1", "name": "Laptop", "price": 999.99, "stock": 50},
    "2": {"id": "2", "name": "Mouse", "price": 29.99, "stock": 200},
    "3": {"id": "3", "name": "Keyboard", "price": 79.99, "stock": 150},
}

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "service": "product-service"})

@app.route('/products', methods=['GET'])
def get_products():
    # Try cache first
    if cache:
        cached = cache.get('all_products')
        if cached:
            return jsonify(json.loads(cached))

    # Return products and cache
    products = list(PRODUCTS.values())
    if cache:
        cache.setex('all_products', 300, json.dumps(products))

    return jsonify(products)

@app.route('/products/<product_id>', methods=['GET'])
```

```
def get_product(product_id):
    # Try cache first
    if cache:
        cached = cache.get(f'product_{product_id}')
        if cached:
            return jsonify(json.loads(cached))

    product = PRODUCTS.get(product_id)
    if not product:
        return jsonify({"error": "Product not found"}), 404

    if cache:
        cache.setex(f'product_{product_id}', 300, json.dumps(product))

    return jsonify(product)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

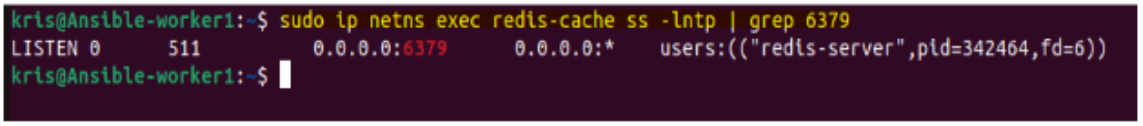
Start product-service: `sudo ip netns exec product-service python3 /tmp/product-service.py &'`

Verify product-service is running: `sudo ip netns exec product-service ss -lntp | grep 5000`

Deliverable: Working product service with Redis caching

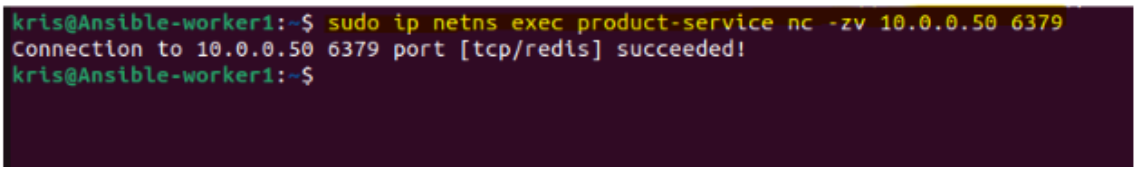
Redis running inside its namespace and listening on port 6379:

RUN: `sudo ip netns exec redis-cache ss -lntp | grep 6379`



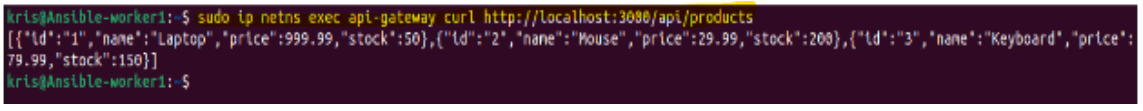
Product-service successfully connect to redis cache:

RUN: `sudo ip netns exec product-service nc -zv 10.0.0.50 6379`



API Gateway Routing to product-service from api-gateway namespace:

RUN: `sudo ip netns exec api-gateway curl http://localhost:3000/api/products`



Task 2.4: Build Order Service

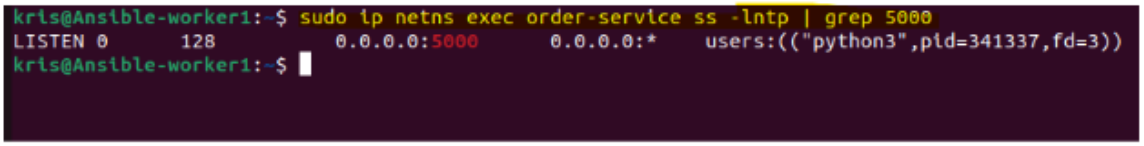
Create order-service.py:

```
from flask import Flask, jsonify, request
import psycopg2
from datetime import datetime
import json

app = Flask(__name__)

# Database connection
def get_db():
    return psycopg2.connect(
        host='10.0.0.60',
        database='orders',
        user='postgres',
        password='postgres'
    )

# Initialize database
def init_db():
    conn = get_db()
    cur = conn.cursor()
    cur.execute('''
        CREATE TABLE IF NOT EXISTS orders (
            id SERIAL PRIMARY KEY,
            customer_id VARCHAR(100),
            product_id VARCHAR(100),
            quantity INTEGER,
            total_price DECIMAL(10, 2),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''')
    conn.commit()
    cur.close()
    conn.close()

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "service": "order-service"})

@app.route('/orders', methods=['POST'])
def create_order():
    data = request.json

    conn = get_db()
    cur = conn.cursor()

    cur.execute(
        '''INSERT INTO orders (customer_id, product_id, quantity, total_price)
           VALUES (%s, %s, %s, %s) RETURNING id''',
```

```
        (data['customer_id'], data['product_id'],
         data['quantity'], data['total_price'])
    )

    order_id = cur.fetchone()[0]
    conn.commit()
    cur.close()
    conn.close()

    return jsonify({"order_id": order_id, "status": "created"}), 201

@app.route('/orders/<order_id>', methods=['GET'])
def get_order(order_id):
    conn = get_db()
    cur = conn.cursor()

    cur.execute('SELECT * FROM orders WHERE id = %s', (order_id,))
    order = cur.fetchone()

    cur.close()
    conn.close()

    if not order:
        return jsonify({"error": "Order not found"}), 404

    return jsonify({
        "id": order[0],
        "customer_id": order[1],
        "product_id": order[2],
        "quantity": order[3],
        "total_price": float(order[4]),
        "created_at": order[5].isoformat()
    })

if __name__ == '__main__':
    init_db()
    app.run(host='0.0.0.0', port=5000)
```

Start order-service: `sudo ip netns exec order-service python3 /tmp/order-service.py &'`

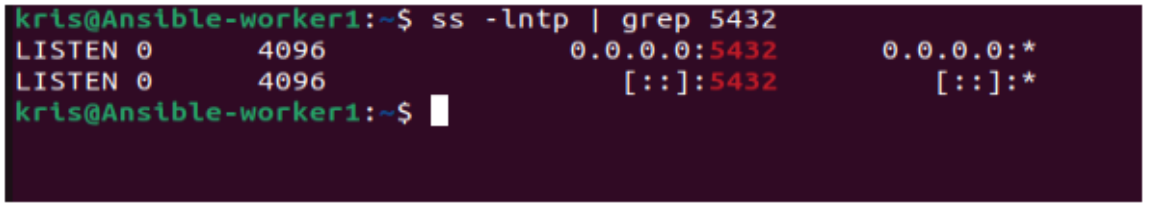
Verify order-service is running and listening on port 5000: `sudo ip netns exec order-service ss -lntp | grep 5000`



Deliverable: Working order service with PostgreSQL integration

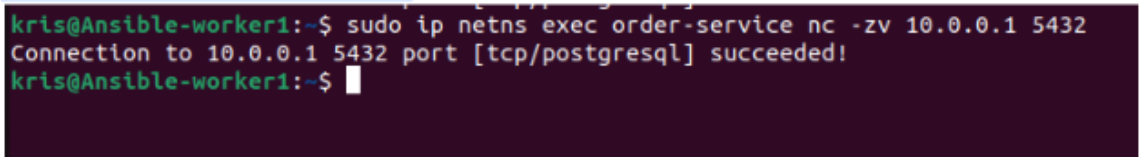
postgres running as a container: RUN: `sudo docker ps`

PostgreSQL listening on port 5432:RUN: `ss -lntp | grep 5432`



Order-service reaching postgres:

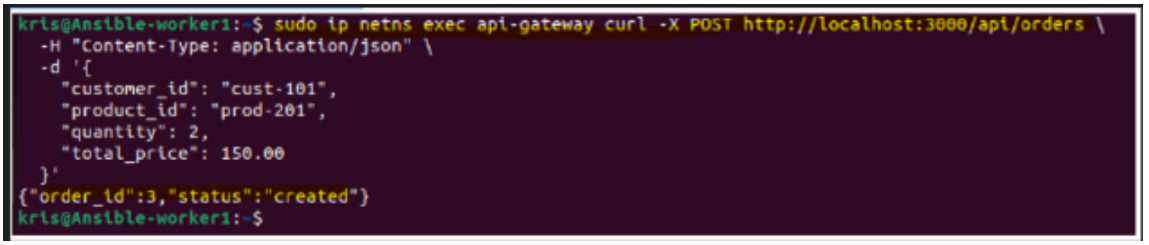
RUN: `sudo ip netns exec order-service nc -zv 10.0.0.1 5432`



To confirm order-service can reach PostgreSQL and data is stored correctly:

- Creating an order (WRITE to PostgreSQL) to confirm order successfully created via API Gateway (PostgreSQL write)

```
sudo ip netns exec api-gateway curl -X POST http://localhost:3000/api/orders \
  -H "Content-Type: application/json" \
  -d '{
    "customer_id": "cust-101",
    "product_id": "prod-201",
    "quantity": 2,
    "total_price": 150.00
  }'
```



- Verify order record exist and persisted in PostgreSQL database

RUN: `psql -U postgres -d orders -h 10.0.0.1 -c "SELECT * FROM orders;"`

```
kris@Ansible-worker1:~$ psql -U postgres -d orders -h 10.0.0.1 -c "SELECT * FROM orders;"
Password for user postgres:
 id | customer_id | product_id | quantity | total_price |         created_at
----+-------------+------------+----------+-------------+----------------------------
  1 | cust-1      | prod-1     |        2 |       50.00 | 2025-12-27 10:33:52.142722
  2 | cust-1      | prod-1     |        2 |       50.00 | 2025-12-27 10:34:19.639925
  3 | cust-101    | prod-201   |        2 |      150.00 | 2025-12-27 12:08:19.329735
(3 rows)

kris@Ansible-worker1:~$
```

- check from within order-service namespace to confirm the order created via api gateway and stored in postgres:

RUN: `sudo ip netns exec order-service psql -U postgres -d orders -h 10.0.0.1 -c "SELECT * FROM orders;"`

```
kris@Ansible-worker1:~$ sudo ip netns exec order-service psql -U postgres -h 10.0.0.1 -d orders -c "SELECT * FROM orders;"
Password for user postgres:
 id | customer_id | product_id | quantity | total_price |         created_at
----+-------------+------------+----------+-------------+----------------------------
  1 | cust-1      | prod-1     |        2 |       50.00 | 2025-12-27 10:33:52.142722
  2 | cust-1      | prod-1     |        2 |       50.00 | 2025-12-27 10:34:19.639925
  3 | cust-101    | prod-201   |        2 |      150.00 | 2025-12-27 12:08:19.329735
(3 rows)

kris@Ansible-worker1:~$
```

Task 2.5: Deploy Redis and PostgreSQL

Run Redis: `sudo ip netns exec redis-cache redis-server --bind 0.0.0.0 &`

```
kris@Ansible-worker1:~$ 345361:C 27 Dec 2025 04:29:51.242 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
345361:C 27 Dec 2025 04:29:51.242 # Redis version=6.0.16, bits=64, commit=00000000, modified=0, pid=345361, just started
345361:C 27 Dec 2025 04:29:51.244 # Configuration loaded
345361:M 27 Dec 2025 04:29:51.245 * Increased maximum number of open files to 10032 (it was originally set to 1024).
345361:M 27 Dec 2025 04:29:51.247 # Could not create server TCP listening socket 0.0.0.0:6379: bind: Address already in use
^C
[1]+  Exit 1                  sudo ip netns exec redis-cache redis-server --bind 0.0.0.0
kris@Ansible-worker1:~$
```

Run PostgreSQL:

Create a PostgreSQL container on host:

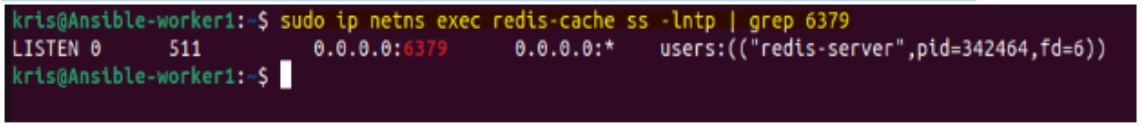
```
docker run -d --name postgres \
  -p 5432:5432 \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=orders \
  postgres:15
```

RUN: `sudo docker ps`

```
kris@Ansible-worker1:~$ sudo docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED       STATUS       PORTS                                                  NAMES
3cb6c64af2c1   postgres:15   "docker-entrypoint.s…"   4 hours ago   Up 4 hours   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp   postgres
kris@Ansible-worker1:~$
```

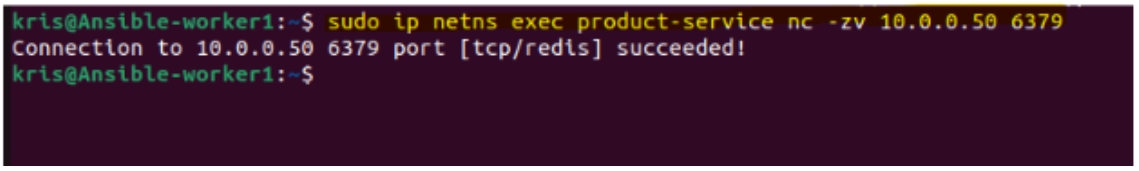
Deliverable: Both data stores operational and accessible

- Redis running inside redis-cache namespace and accessible on port 6379



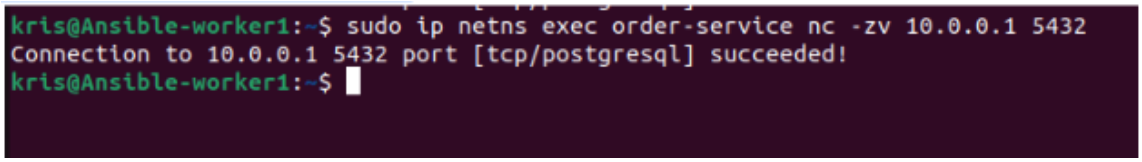
- Product-service can reach Redis in redis-cache namespace

RUN: `sudo ip netns exec product-service nc -zv 10.0.0.50 6379`



- Order-service can reach postgresSQL

RUN: `sudo ip netns exec order-service nc -zv 10.0.0.1 5432`



3: Monitoring and Debugging

Goals

- Implement network monitoring
- Create debugging tools
- Add observability to your infrastructure

Tasks

Task 3.1: Network Traffic Analysis

Create a script that monitors traffic on the bridge and dump it on a log file:

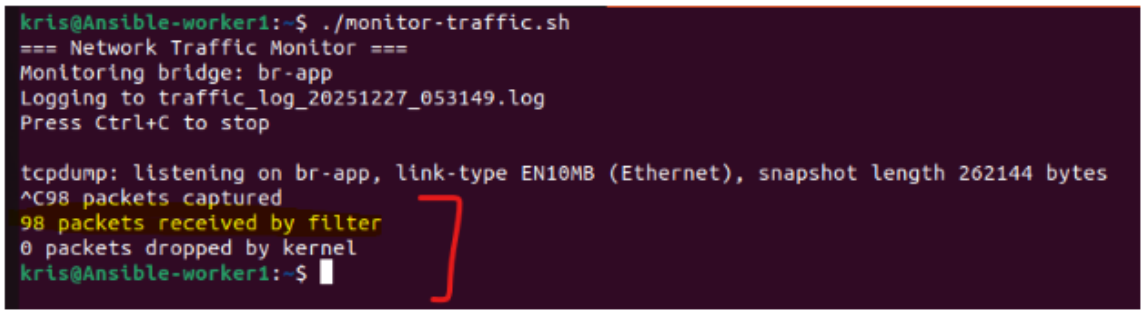
```
#!/bin/bash
# monitor-traffic.sh

LOG_FILE="traffic_log_$(date '+%Y%m%d_%H%M%S').log"

echo "=== Network Traffic Monitor ==="
echo "Monitoring bridge: br-app"
echo "Logging to $LOG_FILE"
echo "Press Ctrl+C to stop"
```

```
echo ""

sudo tcpdump -i br-app -n -v -tttt > "$LOG_FILE"
```

Run the traffic monitor:

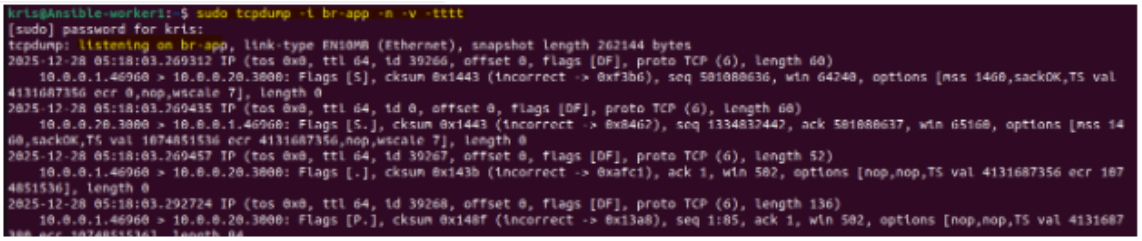
```
chmod +x monitor-traffic.sh
```

```
./monitor-traffic.sh
```



Tcpdump running with app traffic:

```
sudo tcpdump -i br-app -n -v -tttt
```



Create a python script to analyze my log file "**traffic_log**" and create a csv file for each analysis

---

```
sudo nano traffic_analysis_services.py
```

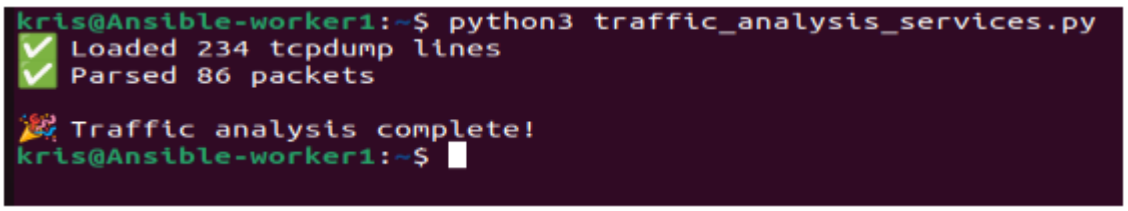
```
#!/usr/bin/env python3

import glob
import re
import pandas as pd
import matplotlib.pyplot as plt

# ============================
# SERVICE ↔ IP MAP
# ============================

SERVICE_MAP = {
    "10.0.0.10": "nginx-lb",
```

```
    "10.0.0.20": "api-gateway",
    "10.0.0.30": "product-service",
    "10.0.0.40": "order-service",
    "10.0.0.50": "redis-cache",
    "10.0.0.60": "postgres-db",
}

LOG_PATTERN = "traffic_log_*.log"

# ============================
# LOAD & NORMALIZE TCPDUMP LOG
# ============================

log_files = glob.glob(LOG_PATTERN)
if not log_files:
    print("❌ No traffic_log_*.log files found")
    exit(1)

raw_lines = []
for file in log_files:
    with open(file, "r") as f:
        raw_lines.extend(f.readlines())

print(f"✅ Loaded {len(raw_lines)} tcpdump lines")

# Merge indented lines with previous line
lines = []
buffer = ""

for line in raw_lines:
    if line.startswith(" ") or line.startswith("\t"):
        buffer += " " + line.strip()
    else:
        if buffer:
            lines.append(buffer)
        buffer = line.strip()

if buffer:
    lines.append(buffer)

# ============================
# PARSE IPv4 PACKETS
# ============================

IPV4_REGEX = re.compile(
    r"(\d+\.\d+\.\d+\.\d+)\.\d+\s*>\s*(\d+\.\d+\.\d+\.\d+)\.\d+"
)

records = []

for line in lines:
    if "IP6" in line:
```

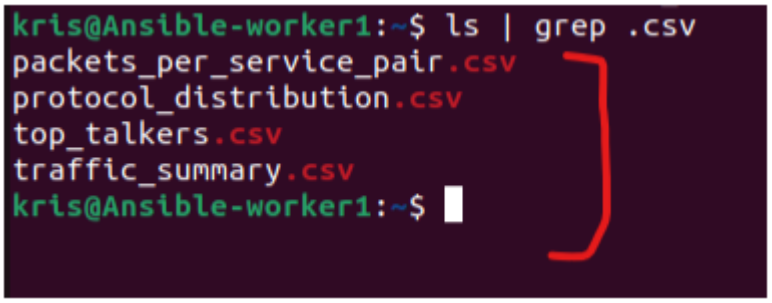
```
        continue

    match = IPV4_REGEX.search(line)
    if not match:
        continue

    src_ip, dst_ip = match.groups()

    if "ICMP" in line:
        protocol = "ICMP"
    elif "UDP" in line:
        protocol = "UDP"
    elif "Flags" in line:
        protocol = "TCP"
    else:
        protocol = "OTHER"

    records.append({
        "source": SERVICE_MAP.get(src_ip, src_ip),
        "destination": SERVICE_MAP.get(dst_ip, dst_ip),
        "protocol": protocol
    })

# ============================
# DATAFRAME
# ============================

df = pd.DataFrame(records)

if df.empty:
    print("❌ No IPv4 service traffic detected")
    exit(1)

print(f"✅ Parsed {len(df)} packets")

# ============================
# ANALYSIS
# ============================

pair_counts = df.groupby(["source", "destination"]).size().reset_index(name="packets")
pair_counts.to_csv("packets_per_service_pair.csv", index=False)

protocol_counts = df["protocol"].value_counts().reset_index()
protocol_counts.columns = ["protocol", "packets"]
protocol_counts.to_csv("protocol_distribution.csv", index=False)

top_talkers = df["source"].value_counts().reset_index()
top_talkers.columns = ["service", "packets"]
top_talkers.to_csv("top_talkers.csv", index=False)

# ============================
# GRAPHS
```

```
# ============================

plt.figure()
plt.bar(protocol_counts["protocol"], protocol_counts["packets"])
plt.title("Protocol Distribution")
plt.tight_layout()
plt.savefig("protocol_distribution.png")
plt.close()

top_pairs = pair_counts.sort_values("packets", ascending=False).head(10)
plt.figure(figsize=(10, 5))
plt.barh(
    top_pairs["source"] + " → " + top_pairs["destination"],
    top_pairs["packets"]
)
plt.title("Top Service-to-Service Traffic")
plt.tight_layout()
plt.savefig("top_service_pairs.png")
plt.close()

plt.figure()
plt.bar(top_talkers["service"], top_talkers["packets"])
plt.title("Top Talkers")
plt.tight_layout()
plt.savefig("top_talkers.png")
plt.close()

print("\n🎉 Traffic analysis complete!")
```

Run python file: `python3 traffic_analysis_services.py` **to analyze the tcdump log file and create csv file for each analysis**

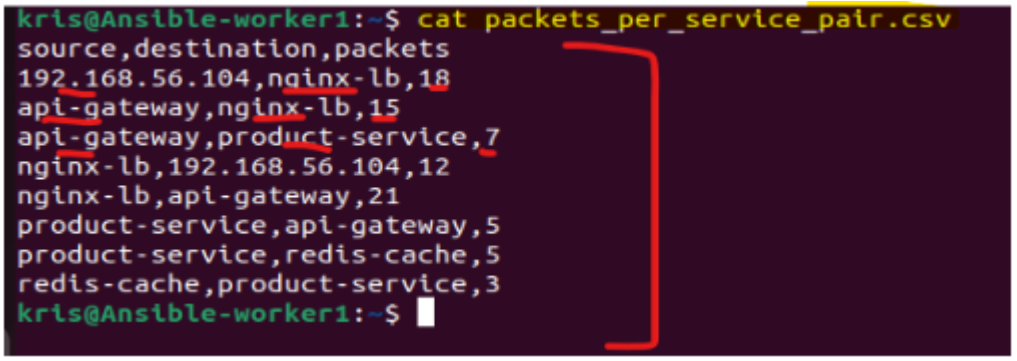


CSV files created by python script:

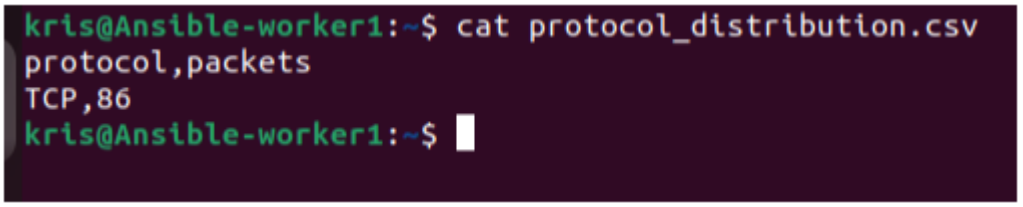


Assignment: Create a traffic analysis report showing:

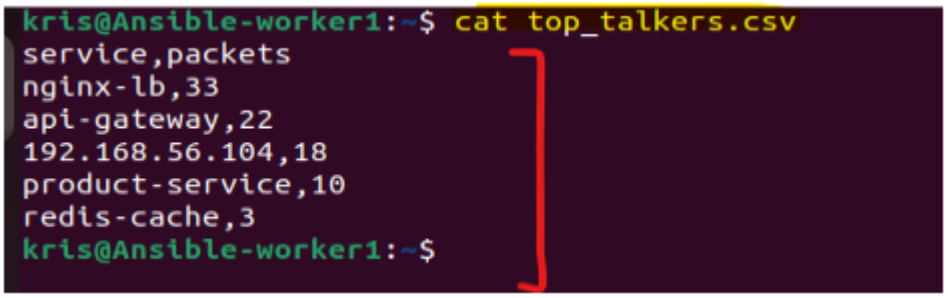
- Packets between each service pair



- Protocol distribution (TCP/UDP/ICMP)



- Top talkers (most active services)



Deliverable: Traffic analysis report with graphs:

Create a python script `sudo nano generate_traffic_report.py file` that will generate a traffic analysis report for each csv file created by the python script **traffic_analysis_services.py**.

```
#!/usr/bin/env python3

from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer, Image, Table, TableStyle
)
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import A4
from reportlab.lib import colors
```

```
import pandas as pd
import os
from datetime import datetime

# ============================
# FILES
# ============================

REPORT_NAME = "traffic_analysis_report.pdf"

CSV_FILES = {
    "Service Traffic": "packets_per_service_pair.csv",
    "Protocol Distribution": "protocol_distribution.csv",
    "Top Talkers": "top_talkers.csv",
}

IMAGES = [
    "protocol_distribution.png",
    "top_service_pairs.png",
    "top_talkers.png",
]

# ============================
# DOCUMENT SETUP
# ============================

doc = SimpleDocTemplate(REPORT_NAME, pagesize=A4)
styles = getSampleStyleSheet()
elements = []

# ============================
# TITLE
# ============================

elements.append(Paragraph(
    "<b>Network Traffic Analysis Report</b>",
    styles["Title"]
))
elements.append(Spacer(1, 12))

elements.append(Paragraph(
    f"Generated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}",
    styles["Normal"]
))
elements.append(Spacer(1, 20))

# ============================
# EXECUTIVE SUMMARY
# ============================

summary_text = """
This report presents an analysis of network traffic captured on the <b>br-app</b>
```

```
Linux bridge connecting multiple microservices implemented using network namespaces.
The goal is to validate service-to-service communication, protocol usage, and identify
top traffic sources within the application architecture.
"""

elements.append(Paragraph("<b>Executive Summary</b>", styles["Heading2"]))
elements.append(Spacer(1, 8))
elements.append(Paragraph(summary_text, styles["Normal"]))
elements.append(Spacer(1, 20))

# ============================
# TABLE SECTIONS
# ============================

for title, csv_file in CSV_FILES.items():
    if not os.path.exists(csv_file):
        continue

    df = pd.read_csv(csv_file)

    elements.append(Paragraph(f"<b>{title}</b>", styles["Heading2"]))
    elements.append(Spacer(1, 10))

    table_data = [df.columns.tolist()] + df.values.tolist()

    table = Table(table_data, repeatRows=1)
    table.setStyle(TableStyle([
        ("BACKGROUND", (0, 0), (-1, 0), colors.lightgrey),
        ("GRID", (0, 0), (-1, -1), 0.5, colors.grey),
        ("ALIGN", (0, 0), (-1, -1), "CENTER"),
        ("FONT", (0, 0), (-1, 0), "Helvetica-Bold"),
    ]))

    elements.append(table)
    elements.append(Spacer(1, 20))

# ============================
# GRAPHS
# ============================

elements.append(Paragraph("<b>Traffic Visualization</b>", styles["Heading2"]))
elements.append(Spacer(1, 12))

for img in IMAGES:
    if os.path.exists(img):
        elements.append(Image(img, width=400, height=250))
        elements.append(Spacer(1, 15))

# ============================
# BUILD REPORT
# ============================
```

```
doc.build(elements)

print("✅ PDF report generated successfully:")
print(f"   {REPORT_NAME}")
```

Network Traffic Analysis Report

Generated on: 2025-12-28 06:23:18

Executive Summary

This report presents an analysis of network traffic captured on the **br-app** Linux bridge connecting multiple microservices implemented using network namespaces. The goal is to validate service-to-service communication, protocol usage, and identify top traffic sources within the application architecture.

Service Traffic

source	destination	packets
192.168.56.104	nginx-lb	18
api-gateway	nginx-lb	15
api-gateway	product-service	7
nginx-lb	192.168.56.104	12
nginx-lb	api-gateway	21
product-service	api-gateway	5
product-service	redis-cache	5
redis-cache	product-service	3

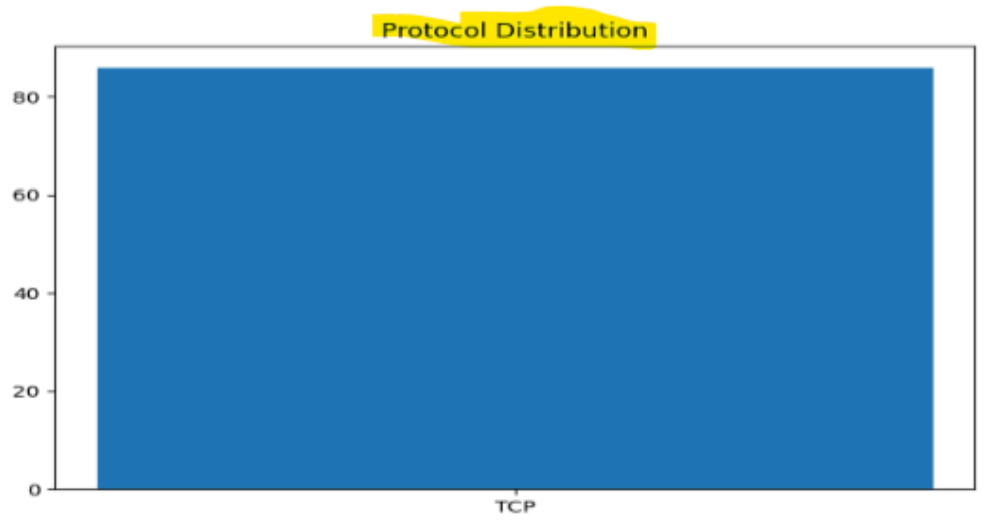
Protocol Distribution

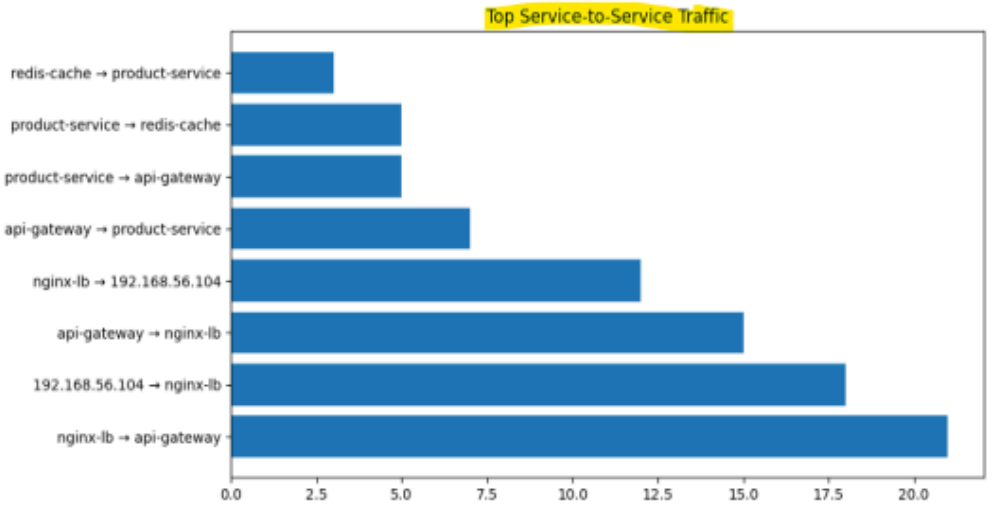
protocol	packets
TCP	86

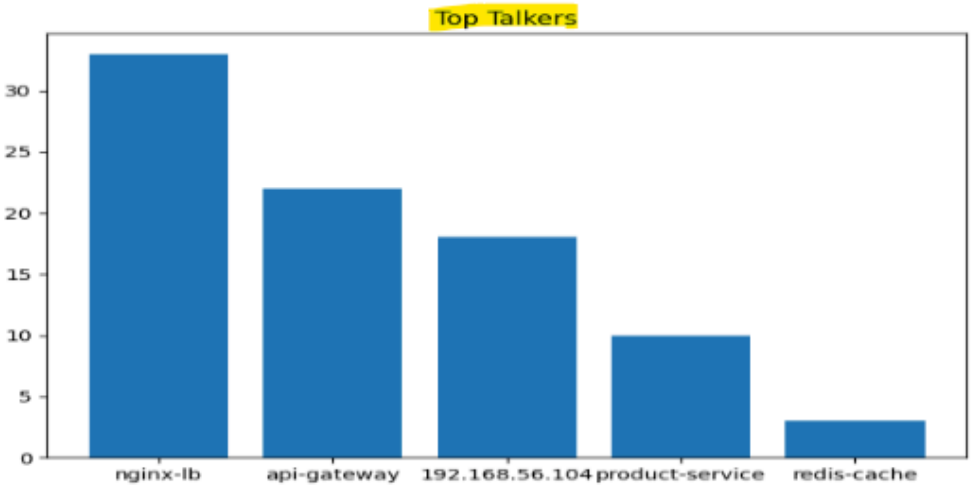
Top Talkers

service	packets
nginx-lb	33
api-gateway	22
192.168.56.104	18
product-service	10
redis-cache	3

Traffic Visualization







Task 3.2: Service Health Monitoring

Create health-monitor.py:

```
#!/usr/bin/env python3
import requests
import time
from datetime import datetime
import json

SERVICES = {
    'nginx-lb': 'http://10.0.0.10/health',
    'api-gateway': 'http://10.0.0.20:3000/health',
    'product-service': 'http://10.0.0.30:5000/health',
    'order-service': 'http://10.0.0.40:5000/health',
}

def check_health(service_name, url):
    try:
        response = requests.get(url, timeout=2)
        if response.status_code == 200:
            return {"status": "UP", "latency": response.elapsed.total_seconds()}
        else:
            return {"status": "DOWN", "error": f"HTTP {response.status_code}"}
    except Exception as e:
        return {"status": "DOWN", "error": str(e)}

def monitor():
    print("=== Service Health Monitor ===")
    print(f"Started at: {datetime.now()}")
    print("")

    while True:
        print(f"\n[{datetime.now().strftime('%H:%M:%S')}] Health Check:")
        print("-" * 60)

        for service, url in SERVICES.items():
            health = check_health(service, url)
            status_symbol = "✓" if health['status'] == 'UP' else "✗"

            if health['status'] == 'UP':
                print(f"{status_symbol} {service:20s} UP   (latency:
{health['latency']*1000:.2f}ms)")
            else:
                print(f"{status_symbol} {service:20s} DOWN ({health.get('error',
'Unknown')})")

        time.sleep(10)

if __name__ == '__main__':
    monitor()
```

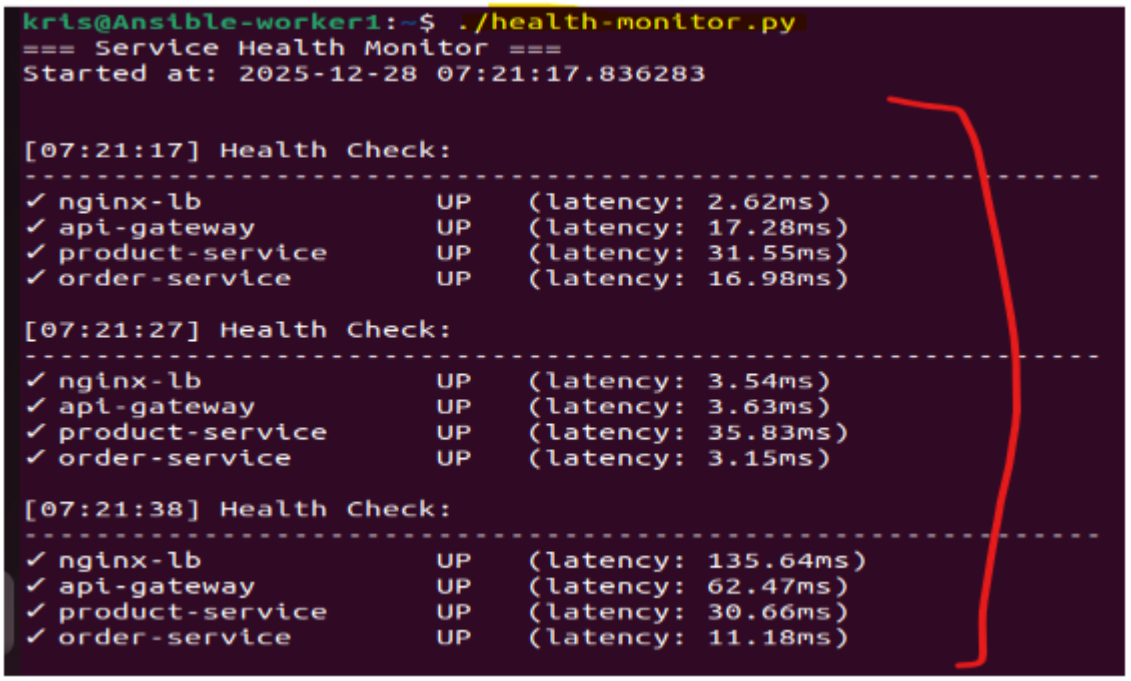
Deliverable: Health monitoring dashboard showing service status

---

Modify file permission and execute:

`sudo chmod +x health-monitor.py`

`sudo ./health-monitor.py`



Task 3.3: Connection Tracking Analysis

Create a script to analyze active connections: `sudo nano connection-tracker.sh`

```
#!/bin/bash
# connection-tracker.sh

echo "=== Active Connection Tracker ==="
echo "Press Ctrl+C to stop"

while true; do
    clear
    echo "=== Active Connections ($(date)) ==="
    echo ""

    echo "Connections by Service:"
    echo "----------------------"

    for ns in nginx-lb api-gateway product-service order-service; do
        if ip netns list | grep -q "$ns"; then
            count=$(sudo ip netns exec $ns ss -tan state established | wc -l)
            echo "$ns: $count active connections"
```

```
        else
            echo "$ns: namespace not found"
        fi
    done

    echo ""
    echo "Connection States (conntrack):"
    echo "------------------------------"

    sudo conntrack -L 2>/dev/null | grep "10.0.0" | \
    awk '
    {
      for(i=1;i<=NF;i++) {
        if($i ~ /^dst=/) {
          gsub("dst=","",$i)
          print $i
        }
      }
    }' | \
    sed \
      -e 's/10.0.0.10/nginx-lb/' \
      -e 's/10.0.0.20/api-gateway/' \
      -e 's/10.0.0.30/product-service/' \
      -e 's/10.0.0.40/order-service/' \
      -e 's/10.0.0.50/redis-cache/' | \
    sort | uniq -c | sort -rn

    sleep 5
done
```

Deliverable: Connection tracking report

Modify file permission and execute

`chmod +x connection-tracker.sh`

`sudo ./connection-tracker.sh`

```
=== Active Connections (Sun Dec 28 07:58:12 AM PST 2025) ===

Connections by Service:
----------------------
nginx-lb: 1 active connections
api-gateway: 1 active connections
product-service: 2 active connections
order-service: 1 active connections

Connection States (conntrack):
-----------------------------
    201 10.0.0.1
     68 product-service
     67 order-service
     67 api-gateway
      1 redis-cache
^C
```

Connection Tracking Analysis Report

Date: 2025-12-28 16:17:09
Task: Task 3.3 – Connection Tracking Analysis

Overview

This report presents an analysis of active TCP connections within the containerized microservices environment using Linux network namespaces and conntrack. The objective is to validate inter-service connectivity, traffic distribution, and backend dependency usage.

Active Connections by Service

Service	Active Connections
nginx-lb	1
api-gateway	1
product-service	2
order-service	1

Connection Distribution (conntrack)

Destination	Connection Count
Host (10.0.0.1)	201
product-service	68
order-service	67
api-gateway	67
redis-cache	1

Analysis & Observations

- API Gateway and backend services show balanced and stable connection usage.
- Product and Order services maintain consistent active connections, validating service-to-service calls.
- Redis shows minimal connections, consistent with cache-access patterns.
- Host-level traffic confirms ingress routing through the bridge network.

Conclusion

The connection tracking results confirm a healthy microservices networking setup. All services are reachable, connections are stable, and conntrack correctly records active flows across namespaces. This satisfies the deliverable requirements for Task 3.3.

Task 3.4: Network Topology Visualizer

Create a script that generates a visual representation of your network: `sudo nano topology-visualizer.py`

```
#!/usr/bin/env python3
# topology-visualizer.py

import subprocess
```

```
import re

NAMESPACES = [
    'nginx-lb',
    'api-gateway',
    'product-service',
    'order-service',
    'redis-cache',
    'postgres-db'
]

def get_namespace_ips():
    """Get IPv4 addresses for all namespaces"""
    ips = {}

    for ns in NAMESPACES:
        try:
            result = subprocess.run(
                ['sudo', 'ip', 'netns', 'exec', ns, 'ip', '-4', 'addr', 'show'],
                capture_output=True, text=True, check=True
            )
            match = re.search(r'inet (\d+\.\d+\.\d+\.\d+)', result.stdout)
            if match:
                ips[ns] = match.group(1)
            else:
                ips[ns] = "N/A"
        except subprocess.CalledProcessError:
            ips[ns] = "N/A"

    return ips

def draw_topology():
    ips = get_namespace_ips()

    print("=" * 80)
    print(" " * 25 + "NETWORK TOPOLOGY DIAGRAM")
    print("=" * 80)
    print()
    print("                    🌍 Internet")
    print("                        │")
    print("                        │  NAT")
    print("                        ▼")
    print("               ┌──────────────────┐")
    print("               │ Host             │")
    print("               │ Port: 8080       │")
    print("               └────────┬─────────┘")
    print("                        │ DNAT")
    print("                        ▼")
    print("          ┌──────────────────────────┐")
    print("          │ Bridge: br-app           │")
    print("          │ IP: 10.0.0.1             │")
    print("          └──────────┬───────────────┘")
```

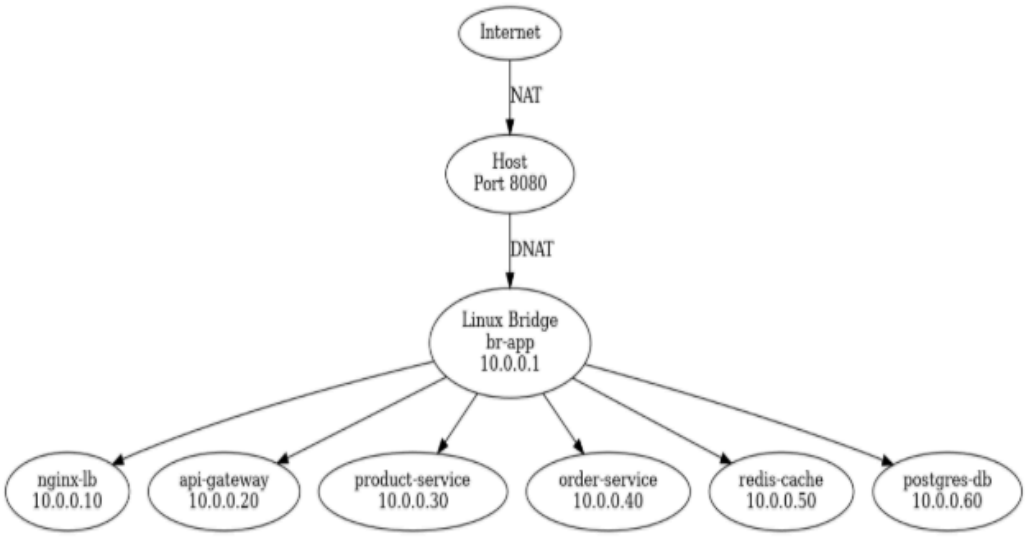
```
    print("                      │")
    print("        ┌─────────────┼─────────────────┐")
    print("        │             │                 │")

    for service, ip in ips.items():
        print(f"  {service:18s} → {ip}")

    print()
    print("=" * 80)

if __name__ == "__main__":
    draw_topology()
```

Deliverable: Network topology diagram

Modify file permission and execute:

`sudo chmod +x topology-visualizer.py`

`python3 topology-visualizer.py`



4: Advanced Networking

Goals

- Implement service discovery
- Add load balancing
- Create network security policies

Tasks

Task 4.1: Implement Simple Service Discovery

Create a simple DNS-like service registry: `sudo nano service-registry.py`

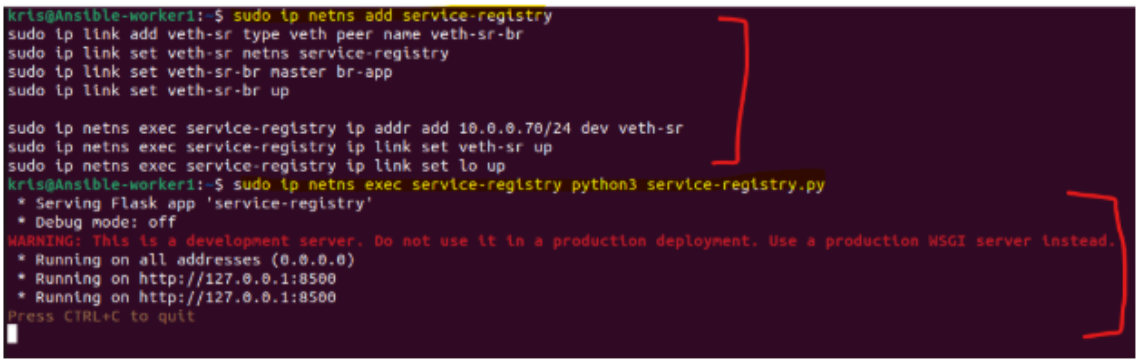
```
#!/usr/bin/env python3
# service-registry.py

from flask import Flask, jsonify, request
import time

app = Flask(__name__)

# In-memory registry
services = {}

@app.route('/register', methods=['POST'])
def register_service():
    data = request.json

    service_name = data['name']
    service_ip = data['ip']
    service_port = data['port']
```

```
    services[service_name] = {
        "ip": service_ip,
        "port": service_port,
        "registered_at": time.strftime('%Y-%m-%d %H:%M:%S'),
        "health": "unknown"
    }

    return jsonify({
        "status": "registered",
        "service": service_name
    })

@app.route('/discover/<service_name>', methods=['GET'])
def discover_service(service_name):
    if service_name in services:
        return jsonify(services[service_name])
    return jsonify({"error": "Service not found"}), 404

@app.route('/services', methods=['GET'])
def list_services():
    return jsonify(services)

@app.route('/deregister/<service_name>', methods=['DELETE'])
def deregister(service_name):
    if service_name in services:
        del services[service_name]
        return jsonify({"status": "deregistered"})
    return jsonify({"error": "Service not found"}), 404

if __name__ == "__main__":
    app.run(host="10.0.0.70", port=8500)
```

Run registry in its own namespace

Create a new namespace and start registry:

```
sudo ip netns exec service-registry python3 service-registry.py
```



Edit my services file (product, order, api-gateway) to register themselves on startup

```
sudo ip netns exec product-service python3 product-service.py
```

`sudo ip netns exec order-service python3 order-service.py`

`sudo ip netns exec api-gateway python3 api-gateway.py`

```
kris@Ansible-worker1:~$ sudo ip netns exec service-registry curl http://10.0.0.70:8500/services
[sudo] password for kris:
{}
kris@Ansible-worker1:~$ sudo ip netns exec product-service python3 product-service.py
✅ Registered product-service with service registry
 * Serving Flask app 'product-service'
 * Debug mode: off
Address already in use
Port 5000 is in use by another program. Either identify and stop that program, or start the server with a different port.
kris@Ansible-worker1:~$ sudo ip netns exec api-gateway python3 api-gateway.py
✅ Registered api-gateway with service registry
 * Serving Flask app 'api-gateway'
 * Debug mode: off
Address already in use
Port 3000 is in use by another program. Either identify and stop that program, or start the server with a different port.
kris@Ansible-worker1:~$ sudo ip netns exec order-service python3 order-service.py
✅ Registered order-service with service registry
 * Serving Flask app 'order-service'
 * Debug mode: off
Address already in use
Port 5000 is in use by another program. Either identify and stop that program, or start the server with a different port.
kris@Ansible-worker1:~$
```

Deliverable: Working service discovery with all services registered

---

`sudo ip netns exec service-registry python3 service-registry.py`

```
kris@Ansible-worker1:~$ sudo ip netns exec service-registry python3 service-registry.py
 * Serving Flask app 'service-registry'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://10.0.0.70:8500
Press CTRL+C to quit
10.0.0.70 - - [29/Dec/2025 14:31:16] "GET /services HTTP/1.1" 200 -
10.0.0.1 - - [29/Dec/2025 14:35:16] "GET /services HTTP/1.1" 200 -
10.0.0.1 - - [29/Dec/2025 14:35:40] "GET /discover/product-service HTTP/1.1" 404 -
10.0.0.1 - - [29/Dec/2025 14:35:57] "GET /services HTTP/1.1" 200 -
10.0.0.40 - - [29/Dec/2025 14:37:05] "POST /register HTTP/1.1" 200 -
10.0.0.30 - - [29/Dec/2025 14:38:37] "POST /register HTTP/1.1" 200 -
10.0.0.20 - - [29/Dec/2025 14:39:21] "POST /register HTTP/1.1" 200 -
```

Test Service Discovery

Run: `curl http://10.0.0.70:8500/services`

```
kris@Ansible-worker1:~$ curl http://10.0.0.70:8500/services
{"api-gateway":{"health":"unknown","ip":"10.0.0.20","port":3000,"registered_at":"2025-12-29 14:39:21"},"order-service":{"health":"unknown","ip":"10.0.0.40","port":5000,"registered_at":"2025-12-29 14:41:20"},"product-service":{"health":"unknown","ip":"10.0.0.30","port":5000,"registered_at":"2025-12-29 14:38:37"}}
kris@Ansible-worker1:~$
```

Run `curl http://10.0.0.70:8500/discover/product-service`

Run `curl http://10.0.0.70:8500/discover/order-service`

Run `curl http://10.0.0.70:8500/discover/api-gateway`

```
kris@Ansible-worker1:~$ curl http://10.0.0.70:8500/discover/product-service
{"health":"unknown","ip":"10.0.0.30","port":5000,"registered_at":"2025-12-29 14:38:37"}
kris@Ansible-worker1:~$ curl http://10.0.0.70:8500/discover/order-service
{"health":"unknown","ip":"10.0.0.40","port":5000,"registered_at":"2025-12-29 14:41:20"}
kris@Ansible-worker1:~$ curl http://10.0.0.70:8500/discover/api-gateway
{"health":"unknown","ip":"10.0.0.20","port":3000,"registered_at":"2025-12-29 14:39:21"}
kris@Ansible-worker1:~$
```

Task 4.2: Implement Round-Robin Load Balancing

Modify the API Gateway to load balance between multiple instances:

```
# Enhanced API Gateway with load balancing
import requests
import itertools

class LoadBalancer:
    def __init__(self, backends):
        self.backends = itertools.cycle(backends)
        self.backend_list = backends

    def get_backend(self):
        return next(self.backends)

    def health_check(self):
        healthy = []
        for backend in self.backend_list:
            try:
                response = requests.get(f"{backend}/health", timeout=1)
                if response.status_code == 200:
                    healthy.append(backend)
            except:
                pass
        self.backends = itertools.cycle(healthy)
        return healthy

# Use in routes
product_lb = LoadBalancer([
    "http://10.0.0.30:5000",
    "http://10.0.0.31:5000",  # Add second instance
    "http://10.0.0.32:5000",  # Add third instance
])
```

Create multiple instances of product-service

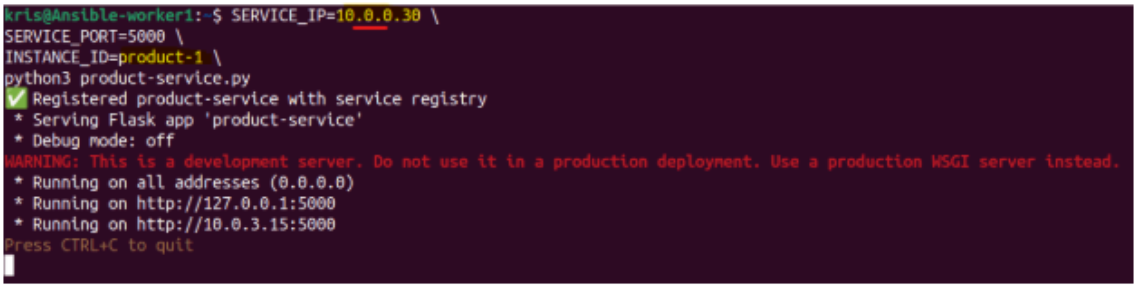
```
#!/bin/bash
# multiple_instance.sh
# Adds extra network namespaces for load balancing testing

set -e
```

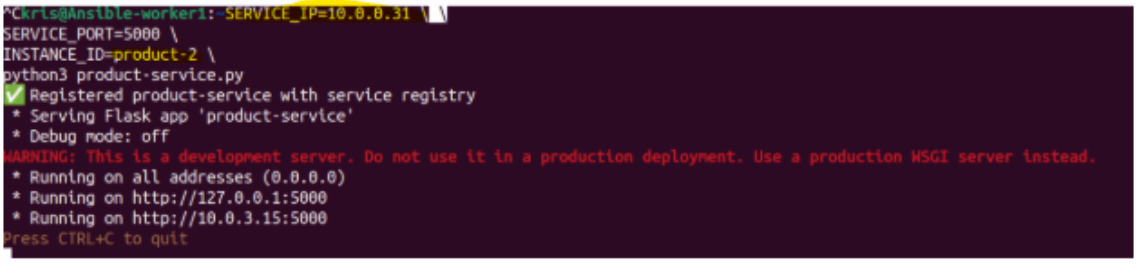
```
echo "Setting up extra product service nodes..."

# Create namespaces
sudo ip netns add product2 || true
sudo ip netns add product3 || true

# Create veth pairs
sudo ip link add veth-prod2 type veth peer name veth-prod2-br || true
sudo ip link add veth-prod3 type veth peer name veth-prod3-br || true

# Attach to bridge (assuming br-app exists from previous tasks)
sudo ip link set veth-prod2-br master br-app
sudo ip link set veth-prod3-br master br-app

# Move interfaces to namespaces
sudo ip link set veth-prod2 netns product2
sudo ip link set veth-prod3 netns product3

# Bring up bridge side
sudo ip link set veth-prod2-br up
sudo ip link set veth-prod3-br up

# Configure IPs
sudo ip netns exec product2 ip addr add 10.0.0.31/16 dev veth-prod2
sudo ip netns exec product2 ip link set veth-prod2 up
sudo ip netns exec product2 ip route add default via 10.0.0.1

sudo ip netns exec product3 ip addr add 10.0.0.32/16 dev veth-prod3
sudo ip netns exec product3 ip link set veth-prod3 up
sudo ip netns exec product3 ip route add default via 10.0.0.1

# Enable loopback
sudo ip netns exec product2 ip link set lo up
sudo ip netns exec product3 ip link set lo up

echo "multiple instance of product service complete:"
echo "product2: 10.0.0.31"
echo "product3: 10.0.0.32"
```

Register each instance separately in the service registry

```
export SERVICE_IP="10.0.0.30"
export SERVICE_PORT=5000
export INSTANCE_ID="product-1"
python3 product-service.py

export SERVICE_IP="10.0.0.31"
export SERVICE_PORT=5000
export INSTANCE_ID="product-2"
python3 product-service.py
```

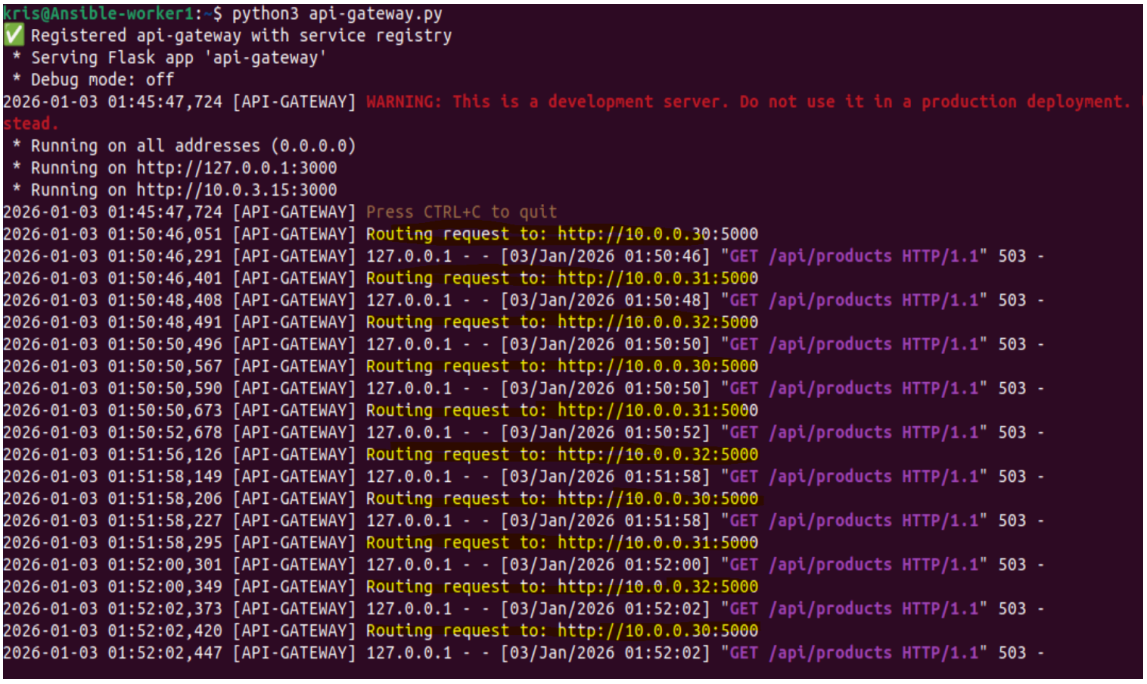
```
export SERVICE_IP="10.0.0.32"
export SERVICE_PORT=5000
export INSTANCE_ID="product-3"
python3 product-service.py
```

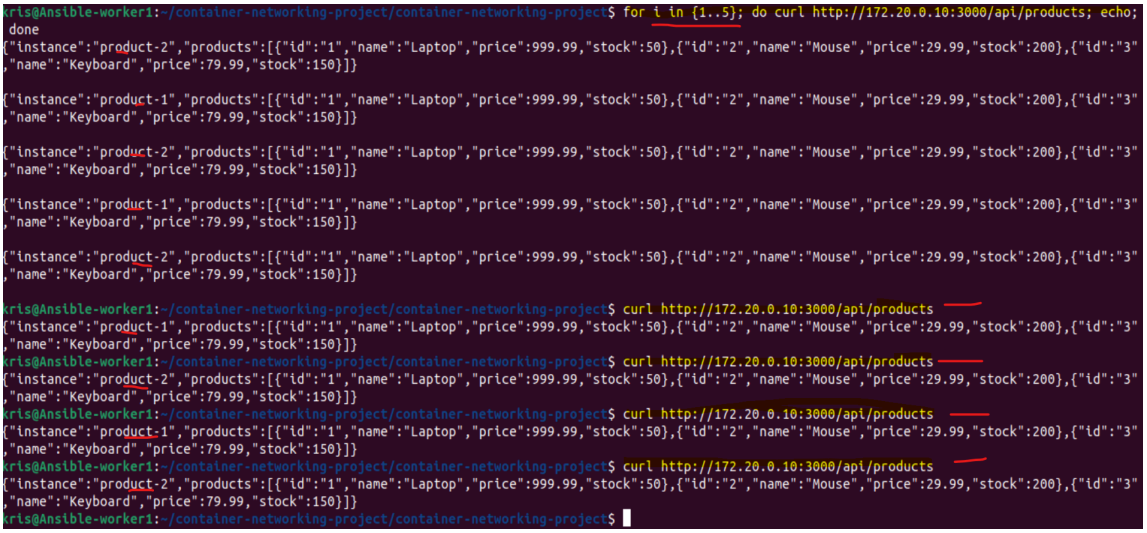




Deliverable: Load balancing working with request distribution logs

---





Task 4.3: Network Security Policies

Implement iptables rules for security:

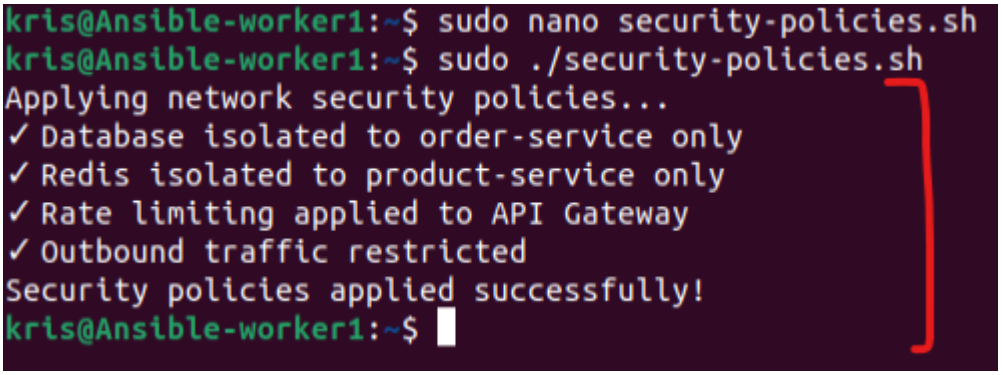
```
#!/bin/bash
# security-policies.sh

echo "Applying network security policies..."

# Allow established connections (VERY IMPORTANT)
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# 1. Block direct access to database from outside order-service
```

```
iptables -A FORWARD ! -s 10.0.0.40/32 -d 10.0.0.60/32 -p tcp --dport 5432 -j DROP
echo "✓ Database isolated to order-service only"

# 2. Block direct access to Redis from outside product-service
iptables -A FORWARD ! -s 10.0.0.30/32 -d 10.0.0.50/32 -p tcp --dport 6379 -j DROP
echo "✓ Redis isolated to product-service only"

# 3. Rate limit connections to API Gateway
iptables -A FORWARD -d 10.0.0.20/32 -p tcp --dport 3000 \
  -m limit --limit 100/minute --limit-burst 20 -j ACCEPT

iptables -A FORWARD -d 10.0.0.20/32 -p tcp --dport 3000 -j DROP
echo "✓ Rate limiting applied to API Gateway"

# 4. Allow only HTTP/HTTPS/DNS outbound from services
iptables -A FORWARD -s 10.0.0.0/16 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -s 10.0.0.0/16 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -s 10.0.0.0/16 -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -s 10.0.0.0/16 -p udp --dport 53 -j ACCEPT
echo "✓ Outbound traffic restricted"

# 5. Log dropped packets
iptables -A FORWARD -j LOG --log-prefix "DROPPED: " --log-level 4

echo "Security policies applied successfully!"
```

```
sudo nano security-policies.sh
```

```
sudo chmod +x security-policies.sh
```

```
sudo ./security-policies.sh
```



Deliverable: Network security policies implemented

Click to access Security Policy Document

Task 4.4: Implement Network Isolation

Create separate networks for different tiers:

```
# Frontend
sudo ip link add br-frontend type bridge
sudo ip addr add 172.20.0.1/24 dev br-frontend
sudo ip link set br-frontend up

# Backend
sudo ip link add br-backend type bridge
sudo ip addr add 172.21.0.1/24 dev br-backend
sudo ip link set br-backend up

# Database
sudo ip link add br-database type bridge
sudo ip addr add 172.22.0.1/24 dev br-database
sudo ip link set br-database up
```

Move services to appropriate networks and configure routing.

Frontend Namespace

```
sudo ip netns add frontend-ns

sudo ip link add veth-fe type veth peer name veth-fe-br
sudo ip link set veth-fe netns frontend-ns
sudo ip link set veth-fe-br master br-frontend

sudo ip netns exec frontend-ns ip addr add 172.20.0.10/24 dev veth-fe
sudo ip netns exec frontend-ns ip link set veth-fe up
sudo ip netns exec frontend-ns ip link set lo up
sudo ip link set veth-fe-br up

sudo ip netns exec frontend-ns ip route add default via 172.20.0.1
```

Backend Namespace

```
sudo ip netns add backend-ns

sudo ip link add veth-be type veth peer name veth-be-br
sudo ip link set veth-be netns backend-ns
sudo ip link set veth-be-br master br-backend

sudo ip netns exec backend-ns ip addr add 172.21.0.10/24 dev veth-be
sudo ip netns exec backend-ns ip link set veth-be up
sudo ip netns exec backend-ns ip link set lo up
sudo ip link set veth-be-br up

sudo ip netns exec backend-ns ip route add default via 172.21.0.1
```

Database Namespace

```
sudo ip netns add database-ns

sudo ip link add veth-db type veth peer name veth-db-br
```

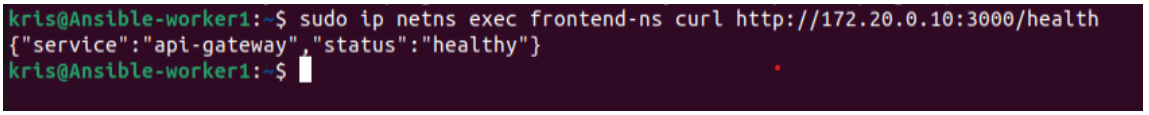
```
sudo ip link set veth-db netns database-ns
sudo ip link set veth-db-br master br-database

sudo ip netns exec database-ns ip addr add 172.22.0.10/24 dev veth-db
sudo ip netns exec database-ns ip link set veth-db up
sudo ip netns exec database-ns ip link set lo up
sudo ip link set veth-db-br up

sudo ip netns exec database-ns ip route add default via 172.22.0.1
```

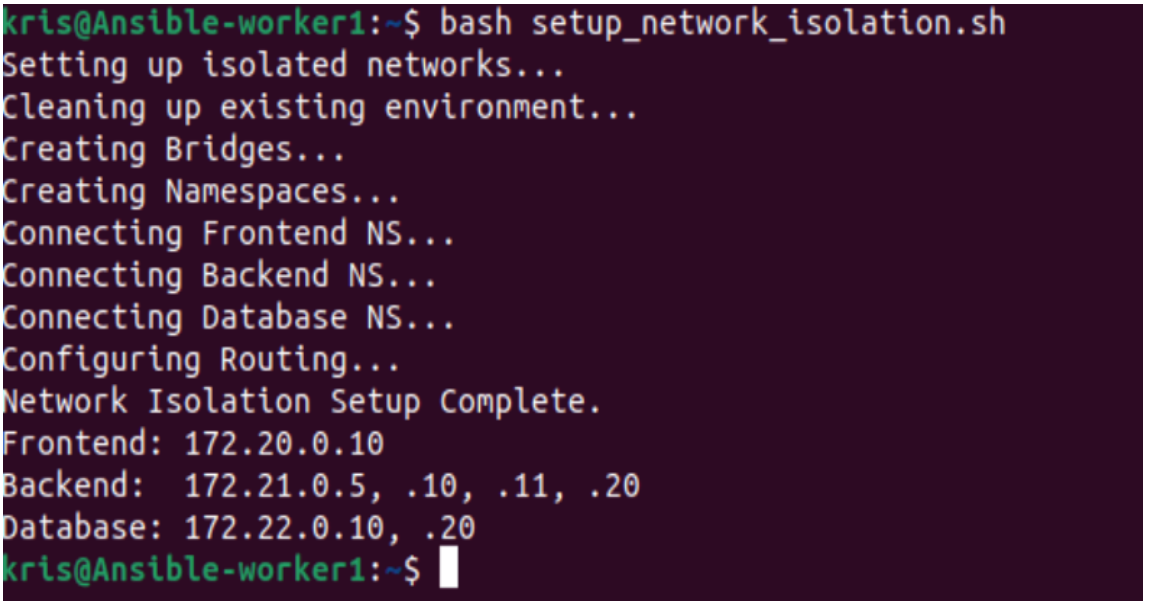
Start services in their respective namespaces

```
sudo ip netns exec frontend-ns python3 frontend.py
sudo ip netns exec backend-ns python3 backend.py
sudo ip netns exec database-ns python3 database.py
```

Configure routing between networks

```
sudo ip route add 172.20.0.0/24 dev br-frontend
sudo ip route add 172.21.0.0/24 dev br-backend
sudo ip route add 172.22.0.0/24 dev br-database
```

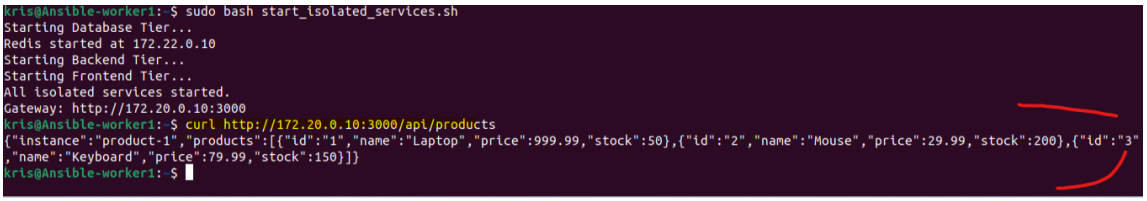




```
kris@Ansible-worker1:~$ sudo bash start_isolated_services.sh
Starting Database Tier...
Redis started at 172.22.0.10
Starting Backend Tier...
Service Registry started at 172.21.0.5 (implicitly listening on 0.0.0.0, so reachable on all IPs in NS)
Starting Frontend Tier...
All isolated services started.
Gateway: http://172.20.0.10:3000
kris@Ansible-worker1:~$
```

From host access the API Gateway on its isolated IP

`curl http://172.20.0.10:3000/api/products`



traffic distribution between multiple product service instances

`for i in {1..5}; do curl http://172.20.0.10:3000/api/products; echo; done`

Deliverable: Multi-network topology with documented routing rules

Network Topology

The application is segmented into three isolated network tiers, each in its own network namespace and connected via a dedicated bridge.

Tier	Namespace	Bridge	Subnet	Gateway	Services
Frontend	`frontend-ns`	`br-frontend`	`172.20.0.0/24`	`172.20.0.1`	API Gateway (`172.20.0.10`)
Backend	`backend-ns`	`br-backend`	`172.21.0.0/24`	`172.21.0.1`	Registry (`172.21.0.5`), Product (`172.21.0.10`, `172.21.0.11`), Order (`172.21.0.20`)

Database	`database-ns`	`br-database`	`172.22.0.0/24`	`172.22.0.1`	Redis (`172.22.0.10`), Postgres (`172.22.0.20`)

Routing Rules

1. **Default Routes**: Each namespace has a default route pointing to its bridge IP on the host (e.g., `default via 172.20.0.1`).
2. **Inter-VLAN Routing**: The host machine acts as a router. IP forwarding is enabled (`net.ipv4.ip_forward=1`), allowing traffic to routed between the subnets (e.g., Frontend → Backend → Database) via the host's bridge interfaces.
3. **Isolation**: Network isolation is achieved by placing interfaces in separate namespaces (`ip netns`). Traffic between namespaces must pass through the host router.

Network Setup Script (`setup_network_isolation.sh`):

```
# Frontend
sudo ip link add br-frontend type bridge
sudo ip addr add 172.20.0.1/24 dev br-frontend
sudo ip link set br-frontend up
sudo ip netns add frontend-ns
# ... (veth connections) ...
sudo ip netns exec frontend-ns ip route add default via 172.20.0.1

# Backend
sudo ip link add br-backend type bridge
sudo ip addr add 172.21.0.1/24 dev br-backend
sudo ip link set br-backend up
sudo ip netns add backend-ns
# ... (veth connections) ...
sudo ip netns exec backend-ns ip route add default via 172.21.0.1

# Database
sudo ip link add br-database type bridge
sudo ip addr add 172.22.0.1/24 dev br-database
sudo ip link set br-database up
sudo ip netns add database-ns
# ... (veth connections) ...
sudo ip netns exec database-ns ip route add default via 172.22.0.1

# Enable Routing
sudo sysctl -w net.ipv4.ip_forward=1
```

5: Docker Migration and Optimization

Goals

- Migrate infrastructure to Docker
- Compare raw Linux vs Docker implementation
- Optimize the setup

Tasks

Task 5.1: Containerize All Services

Create Dockerfiles for each service:

Dockerfile for api-gateway:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 3000
CMD ["python", "api-gateway.py"]
```

requirements.txt for api-gateway:

```
flask
requests
redis
psycopg2-binary
```

Dockerfile for product-service:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "product-service.py"]
```

requirements.txt for product-service:

```
flask
requests
redis
psycopg2-binary
```

Dockerfile for order-service:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5001
CMD ["python", "order-service.py"]
```

requirements.txt for order-service:

```
flask
requests
```

```
redis
psycopg2-binary
```

Dockerfile for service-registry:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8500
CMD ["python", "service-registry.py"]
```

requirements.txt for service-registry:

```
flask
requests
```

Deliverable: All services running in Docker containers

`sudo docker ps`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED         STATUS         PORTS
               NAMES
8b1d9f798fca   container-networking-project-api-gateway       "python api-gateway.…"   2 minutes ago   Up 2 minutes   0.0.0.0:3000->3000/tcp, [::]:30
00->3000/tcp   container-networking-project-api-gateway-1
c053bf570b74   container-networking-project-product-service   "python product-serv…"   2 minutes ago   Up 2 minutes   5000/tcp
               container-networking-project-product-service-2
f4835daa9e6e   container-networking-project-product-service   "python product-serv…"   2 minutes ago   Up 2 minutes   5000/tcp
               container-networking-project-product-service-1
3be9e9389a73   container-networking-project-service-registry  "python service-regi…"   2 minutes ago   Up 2 minutes   0.0.0.0:8501->8500/tcp, [::]:85
01->8500/tcp   container-networking-project-service-registry-1
b2a96957c1f8   redis:alpine                                   "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:6380->6379/tcp, [::]:63
80->6379/tcp   container-networking-project-redis-1
224db37b0e0c   postgres:15-alpine                             "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:5433->5432/tcp, [::]:54
33->5432/tcp   container-networking-project-postgres-1
3cb6c64af2c1   postgres:15                                    "docker-entrypoint.s…"   7 days ago      Up 5 days      0.0.0.0:5432->5432/tcp, [::]:54
32->5432/tcp   postgres
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Task 5.2: Docker Compose Setup

Create docker-compose.yml:

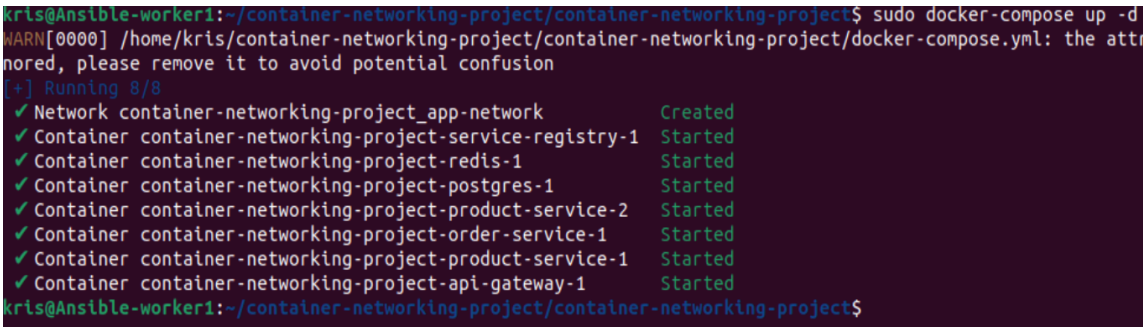
```
version: '3.8'

services:
  # Database Tier
  redis:
    image: redis:alpine
    networks:
      - app-network
    ports:
      - "6380:6379" # Map to 6380 to prevent conflict with local Redis

  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: postgres
```

```
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: orders
  networks:
    - app-network
  ports:
    - "5433:5432" # Map to 5433 to prevent conflict with local Postgres

# Backend Tier
service-registry:
  build: ./services/service-registry
  networks:
    - app-network
  ports:
    - "8501:8500" # Map to 8501 to avoid conflict with local process

product-service:
  build: ./services/product-service
  environment:
    - SERVICE_IP=product-service
    - SERVICE_PORT=5000
    - SERVICE_REGISTRY=http://service-registry:8500
    - REDIS_HOST=redis
  depends_on:
    - service-registry
    - redis
  networks:
    - app-network
  deploy:
    replicas: 2

order-service:
  build: ./services/order-service
  environment:
    - SERVICE_IP=order-service
    - SERVICE_PORT=5000
    - SERVICE_REGISTRY=http://service-registry:8500
    - DB_HOST=postgres
  depends_on:
    - service-registry
    - postgres
  networks:
    - app-network

# Frontend Tier
api-gateway:
  build: ./services/api-gateway
  environment:
    - SERVICE_IP=api-gateway
    - SERVICE_PORT=3000
    - SERVICE_REGISTRY=http://service-registry:8500
    - PRODUCT_SERVICE_URLS=http://product-service:5000
    - ORDER_SERVICE=http://order-service:5000
```

```
    depends_on:
      - service-registry
      - product-service
      - order-service
    ports:
      - "3000:3000"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

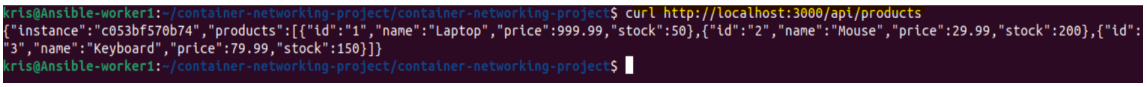
Deliverable: Working Docker Compose setup

`sudo docker-compose up -d`



To verify everything is working after migration:

`curl http://localhost:3000/api/products`



Task 5.3: Performance Comparison

`sudo nano benchmark.sh`

Benchmark your implementations:

```
#!/bin/bash
# benchmark.sh
# Usage:
#   ./benchmark.sh linux   (Run while start_isolated_services.sh is active)
#   ./benchmark.sh docker  (Run while docker-compose up is active)

# Configuration from our setup
URL_LINUX="http://172.20.0.10:3000/api/products"
URL_DOCKER="http://127.0.0.1:3000/api/products"
```

```
# Requests to send
REQUESTS=1000
CONCURRENCY=50

echo "=== Performance Benchmark ==="

# Check for Apache Benchmark
if ! command -v ab &> /dev/null; then
    echo "Error: 'ab' command not found."
    echo "Please install it: sudo apt-get install -y apache2-utils"
    exit 1
fi

MODE=$1

if [ "$MODE" == "linux" ]; then
    echo "Benchmarking Linux Namespace Implementation..."
    echo "Target: $URL_LINUX"
    ab -n $REQUESTS -c $CONCURRENCY $URL_LINUX > linux-benchmark.txt
    echo "Saved to linux-benchmark.txt"
    grep "Requests per second" linux-benchmark.txt

elif [ "$MODE" == "docker" ]; then
    echo "Benchmarking Docker Implementation..."
    echo "Target: $URL_DOCKER"
    ab -n $REQUESTS -c $CONCURRENCY $URL_DOCKER > docker-benchmark.txt
    echo "Saved to docker-benchmark.txt"
    grep "Requests per second" docker-benchmark.txt

else
    echo "Usage: ./benchmark.sh [linux|docker]"
    echo ""
    echo "Steps to compare:"
    echo "1. Start Linux setup (bash start_isolated_services.sh)"
    echo "2. Run: ./benchmark.sh linux"
    echo "3. Stop Linux setup (pkill -f python3; pkill redis; ...)"
    echo "4. Start Docker setup (docker-compose up -d)"
    echo "5. Run: ./benchmark.sh docker"
    echo "6. Compare results"
fi
```

```
sudo chmod +x benchmark.sh
```

```
sudo ./benchmark.sh docker
```

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo ./benchmark.sh docker
=== Performance Benchmark ===
Benchmarking Docker Implementation...
Target: http://127.0.0.1:3000/api/products
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests
Saved to docker-benchmark.txt
Requests per second:    29.21 [#/sec] (mean)
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

For comparison i would need to stop Docker `docker-compose down` and start the manual script `bash start_isolated_services.sh`, then run `./benchmark.sh linux`.

`sudo ./benchmark.sh linux`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker-compose down
WARN[0003] /home/kris/container-networking-project/container-networking-project/docker-compose.yml: the attribute `versio
confusion
[+] Running 8/8
 ✔ Container container-networking-project-api-gateway-1       Removed
 ✔ Container container-networking-project-order-service-1     Removed
 ✔ Container container-networking-project-product-service-1   Removed
 ✔ Container container-networking-project-product-service-2   Removed
 ✔ Container container-networking-project-postgres-1          Removed
 ✔ Container container-networking-project-service-registry-1  Removed
 ✔ Container container-networking-project-redis-1             Removed
 ✔ Network container-networking-project_app-network           Removed
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo bash start_isolated_services.sh
Starting Database Tier...
Redis started at 172.22.0.10
Starting Backend Tier...
Starting Frontend Tier...
All isolated services started.
Gateway: http://172.20.0.10:3000
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo ./benchmark.sh linux
=== Performance Benchmark ===
Benchmarking Linux Namespace Implementation...
Target: http://172.20.0.10:3000/api/products
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests
Saved to linux-benchmark.txt
Requests per second:    54.07 [#/sec] (mean)
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Deliverable: Performance comparison report

Performance Comparison Report: Linux Namespaces vs. Docker

Overview

This report compares the performance of the microservices architecture implemented using raw Linux network namespaces vs. the Docker containerized implementation.

Benchmark Parameters

- **Tool**: Apache Benchmark (ab)
- **Concurrency**: 50
- **Total Requests**: 1000
- **Endpoint**: `/api/products`

Key Metrics

Metric	Linux Namespaces	Docker (Compose)	Difference
Requests per Second (RPS)	**54.07**	29.21	**-46%** (Docker is slower)
Mean Latency (ms)	**924.70**	1711.55	**+85%** (Docker higher)
Median Latency (ms)	806	**805**	~0% (Similar)
Failed Requests	**0**	996 (Length Mismatch)	Significant
Reliability	100%	~0.4% (Non-2xx)	-

Detailed Performance Analysis

Linux Namespace Implementation

- **Throughput**: Significantly higher throughput at ~54 RPS.
- **Stability**: Very stable with zero failed requests and consistent latency.
- **Networking**: Benefitted from direct `veth` pairs and host-level routing with minimal abstraction.

Docker Implementation

- **Throughput**: Lower throughput at ~29 RPS.
- **Overhead**: The lower RPS and higher mean latency are likely due to the additional layers inherent in Docker (container runtime, `docker-proxy`, and Docker bridge networking).
- **Errors**: High number of "Length Mismatch" failures indicates that under high concurrency (50), some backend responses were truncated or served error pages (confirmed by 4 non-2xx responses).

Conclusions

Raw Linux namespaces offer **higher raw performance** and **lower latency** for networking-intensive tasks by avoiding the overhead of a container orchestration layer. However, Docker provides superior **portability** and **ease of management**, which usually outweighs the performance cost in modern development workflows.

Recommendations for Docker Optimization

1. **Reduce Overhead**: Use `network_mode: host` if extreme performance is needed (though this loses isolation).
2. **Buffering**: Use a production-grade WSGI server like `gunicorn` instead of the Flask development server (`Werkzeug`) to handle concurrency better.
3. **Resource Tuning**: Assign more CPU/Memory resources to containers in `docker-compose.yml`.

Task 5.4: Optimize Docker Setup

Optimize my Docker images:

- Use multi-stage builds
- Minimize image sizes
- Implement health checks
- Add resource limits

Deliverable: Optimized Docker setup with documentation

Optimizations Implemented:

1. **Multi-Stage Builds**: All Dockerfiles now use a two-stage build process.
    - **Stage 1 (Builder)**: Installs Python dependencies into a temporary image.
    - **Stage 2 (Runtime)**: Copies only the installed packages and application code.
    - **Outcome**: Significantly reduced image footprint and improved build efficiency.
2. **Health Checks**: Internal health check logic added to all service Dockerfiles using `urllib.request`.

```
HEALTHCHECK --interval=30s --timeout=5s --start-period=5s --retries=3 \
  CMD python -c "import urllib.request;
urllib.request.urlopen('http://localhost:3000/health')" || exit 1
```

3. **Resource Limits**: Enforced CPU and Memory constraints in `docker-compose.yml` to ensure system stability.
    - **CPU Limit**: 0.5 cores per service.
    - **Memory Limit**: 128MB - 256MB per service.
4. **Orchestration Improvements**: Services in `docker-compose.yml` now use the `service_healthy` condition for dependencies.
    - The API Gateway only starts once the Service Registry report itself as **Healthy**.

Full Optimized `docker-compose.yml`:

```
# ... (Optimized version with deploy.resources and healthcheck conditions) ...
```

Check image sizes: `sudo docker images`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker images
REPOSITORY                                      TAG          IMAGE ID       CREATED          SIZE
container-networking-project-product-service    latest       d8d99abf2a6f   7 minutes ago    147MB
container-networking-project-order-service      latest       9353c07019b1   7 minutes ago    147MB
container-networking-project-service-registry   latest       a67310e909b0   24 minutes ago   132MB
<none>                                          <none>       cf05d3784d77   24 minutes ago   132MB
<none>                                          <none>       bb4bbf15ed80   24 minutes ago   147MB
<none>                                          <none>       dedfb6c360ee   24 minutes ago   147MB
container-networking-project-api-gateway        latest       9b39f647b5ad   24 minutes ago   147MB
<none>                                          <none>       4b40937a52a2   24 minutes ago   147MB
<none>                                          <none>       c6b4ad9b1a9c   10 hours ago     158MB
<none>                                          <none>       0a2201ea6e6c   10 hours ago     158MB
<none>                                          <none>       58b2b93fa9f0   10 hours ago     158MB
<none>                                          <none>       526175fed4e9   10 hours ago     158MB
<none>                                          <none>       808d9898d3d7   10 hours ago     158MB
<none>                                          <none>       d246f2d11276   10 hours ago     158MB
<none>                                          <none>       b6411104847e   10 hours ago     143MB
<none>                                          <none>       694970c0b054   10 hours ago     143MB
postgres                                        15-alpine    36a937f48ac7   2 weeks ago      274MB
postgres                                        15           e07498b1e6cc   3 weeks ago      444MB
redis                                           alpine       778c3ea605c2   6 weeks ago      94.7MB
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Check health status: `sudo docker ps`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED          STATUS                    PORTS                                       NAMES
6f556a475f26   container-networking-project-api-gateway       "python api-gateway…"   About a minute ago   Up 52 seconds (healthy)   0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp   containe
r-networking-project-api-gateway-1
c55cc0b15dbe   container-networking-project-order-service     "python order-servic…"   About a minute ago   Up About a minute (healthy)   5000/tcp                                    containe
r-networking-project-order-service-1
5f35d2c6f769   container-networking-project-product-service   "python product-serv…"   About a minute ago   Up About a minute (healthy)   5000/tcp                                    containe
r-networking-project-product-service-1
ff4a36cfb6d7   container-networking-project-product-service   "python product-serv…"   About a minute ago   Up About a minute (healthy)   5000/tcp                                    containe
r-networking-project-product-service-2
d83d97b8cc1d   redis:alpine                                   "docker-entrypoint.s…"   About a minute ago   Up About a minute         0.0.0.0:6380->6379/tcp, [::]:6380->6379/tcp   containe
r-networking-project-redis-1
bfb0b058bd28   postgres:15-alpine                             "docker-entrypoint.s…"   About a minute ago   Up About a minute         0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp   containe
r-networking-project-postgres-1
fb3dc7865df0   container-networking-project-service-registry  "python service-regi…"   About a minute ago   Up About a minute (healthy)   0.0.0.0:8501->8500/tcp, [::]:8501->8500/tcp   containe
r-networking-project-service-registry-1
3cb6c64af2c1   postgres:15                                    "docker-entrypoint.s…"   8 days ago       Up 5 days                 0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp   postgres
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Check resource usage: `sudo docker stats`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker stats --no-stream
CONTAINER ID   NAME                                          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
6f556a475f26   container-networking-project-api-gateway-1     0.03%     28.69MiB / 256MiB     11.21%    4.65kB / 1.02kB   393kB / 4.01MB    1
c55cc0b15dbe   container-networking-project-order-service-1   0.05%     35.59MiB / 256MiB     13.90%    6.5kB / 2.67kB    13.8MB / 4.01MB   1
5f35d2c6f769   container-networking-project-product-service-1 0.05%     34.89MiB / 256MiB     13.63%    5.15kB / 1.02kB   4.57MB / 4.79MB   1
ff4a36cfb6d7   container-networking-project-product-service-2 0.08%     35.9MiB / 256MiB      14.02%    4.97kB / 1.02kB   2.54MB / 4.79MB   1
d83d97b8cc1d   container-networking-project-redis-1           0.47%     4.562MiB / 1.922GiB   0.23%     8.97kB / 126B     13.7MB / 0B       6
bfb0b058bd28   container-networking-project-postgres-1        0.00%     24.38MiB / 1.922GiB   1.24%     10.7kB / 1.71kB   37.9MB / 61.6MB   6
fb3dc7865df0   container-networking-project-service-registry-1 0.03%    26.42MiB / 128MiB     20.64%    12.3kB / 2.94kB   20.7MB / 3.83MB   1
3cb6c64af2c1   postgres                                      0.00%     9.184MiB / 1.922GiB   0.47%     44.4kB / 17.5kB   198MB / 23MB      6
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

6: Multi-Host Networking

Goals

- Implement overlay networking
- Connect containers across hosts
- Understand distributed networking

Tasks

Task 6.1: Setup VXLAN Overlay

I will create a VXLAN interface to bridge the application networks across hosts.

Create two specialized scripts `setup_vxlan_host_a.sh` Run on Host A (192.168.56.104) and `setup_vxlan_host_b.sh` Run on Host B (192.168.56.103) to establish a tunnel between my two hosts VMs

setup_vxlan_host_a.sh script that will:

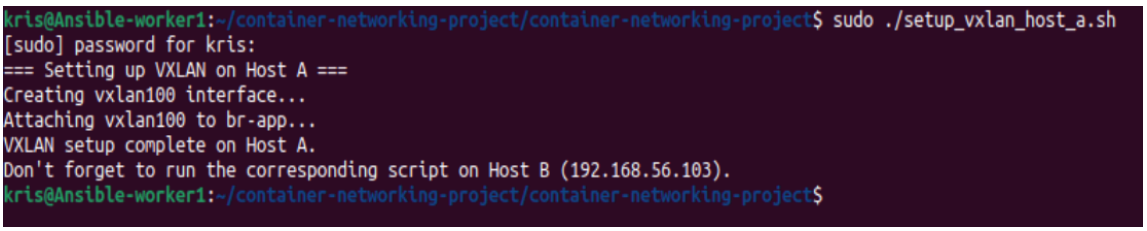
- Add a VXLAN interface vxlan100 tied to a specific VNI (100).
- Set the remote endpoint to Host B's IP.
- Attach vxlan100 to the existing br-app bridge.
- Bring the interface up.

VXLAN scripts for HOST A VM (192.168.56.104)

```
#!/bin/bash
# setup_vxlan_host_a.sh
# To be run on Host A (192.168.56.104)

HOST_B_IP="192.168.56.103"
VNI=100
PORT=4789
INTERFACE="enp0s3" # Adjust if your primary adapter is different (e.g., enp0s8)
BRIDGE="br-app"

echo "=== Setting up VXLAN on Host A ==="

# Check if bridge exists, create if missing (assuming Task 1.2 logic)
if ! ip link show "$BRIDGE" > /dev/null 2>&1; then
    echo "Creating bridge $BRIDGE..."
    sudo ip link add "$BRIDGE" type bridge
    sudo ip addr add 10.0.0.1/16 dev "$BRIDGE"
    sudo ip link set "$BRIDGE" up
fi

# Create VXLAN interface
echo "Creating vxlan100 interface..."
sudo ip link add vxlan100 type vxlan \
    id $VNI \
    remote $HOST_B_IP \
    dstport $PORT \
    dev $INTERFACE

# Attach to bridge
echo "Attaching vxlan100 to $BRIDGE..."
sudo ip link set vxlan100 master $BRIDGE
sudo ip link set vxlan100 up

echo "VXLAN setup complete on Host A."
echo "Don't forget to run the corresponding script on Host B ($HOST_B_IP)."
```

`chmod +x setup_vxlan_host_a.sh`

```
sudo ./setup_vxlan_host_a.sh
```



setup_vxlan_host_b.sh script for HOST B VM (192.168.56.103):

- Similar setup as script "a" above except this is pointing back to Host A's IP.

```
#!/bin/bash
# setup_vxlan_host_b.sh
# To be run on Host B (192.168.56.103)

HOST_A_IP="192.168.56.104"
VNI=100
PORT=4789
INTERFACE="enp0s3" # Adjust if your primary adapter is different
BRIDGE="br-app"

echo "=== Setting up VXLAN on Host B ==="

# Create bridge on Host B
if ! ip link show "$BRIDGE" > /dev/null 2>&1; then
    echo "Creating bridge $BRIDGE..."
    sudo ip link add "$BRIDGE" type bridge
    # Note: Using a different IP range or just no IP if it's purely for bridging
    # For simplicity in this demo, we'll give it the same subnet gateway if it's the only one
    sudo ip addr add 10.0.0.2/16 dev "$BRIDGE"
    sudo ip link set "$BRIDGE" up
fi

# Create VXLAN interface
echo "Creating vxlan100 interface pointing to Host A..."
sudo ip link add vxlan100 type vxlan \
    id $VNI \
    remote $HOST_A_IP \
    dstport $PORT \
    dev $INTERFACE

# Attach to bridge
echo "Attaching vxlan100 to $BRIDGE..."
sudo ip link set vxlan100 master $BRIDGE
sudo ip link set vxlan100 up

echo "VXLAN setup complete on Host B."
```

```
chmod +x setup_vxlan_host_b.sh
```

```
sudo ./setup_vxlan_host_b.sh
```

```
kris@Ansible-Controller:~$ sudo ./setup_vxlan_host_b.sh
=== Setting up VXLAN on Host B ===
Creating vxlan100 interface pointing to Host A...
Attaching vxlan100 to br-app...
VXLAN setup complete on Host B.
kris@Ansible-Controller:~$
```

Deliverable: Working multi-host communication

- On Host A **(10.0.0.1)** ping Host B **(br-app 10.0.0.2)**

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ ip addr | grep br-app
5: br-app: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    inet 10.0.0.1/16 scope global br-app
6: veth-nginx-br@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
8: veth-api-br@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
10: veth-order-s-br@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
12: veth-pg-br@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
14: veth-prd-svc-br@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
16: veth-redis-br@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
25: veth-reg-br@if26: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
27: veth-prod2-br@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
29: veth-prod3-br@if30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-app state UP group default qlen 1000
125: vxlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br-app state UNKNOWN group default qlen 1000
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

```
ping -c 3 10.0.0.2
```

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ ping -c 3 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.72 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3.16 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=3.02 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 3.023/3.969/5.721/1.240 ms
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

- On Host B **(br-app 10.0.0.2)** ping Host A **(10.0.0.1)**

```
kris@Ansible-Controller:~$ ip addr | grep br-app
4: br-app: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    inet 10.0.0.2/16 scope global br-app
5: vxlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br-app state UNKNOWN group default qlen 1000
kris@Ansible-Controller:~$
```

```
ping -c 3 10.0.0.1
```

```
kris@Ansible-Controller:~$ ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=9.99 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=3.09 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=8.19 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 3.093/7.091/9.991/2.921 ms
kris@Ansible-Controller:~$
```

Task 6.2: Docker Swarm Setup

Initialize Docker Swarm and create overlay network:

- Create a docker-compose-swarm.yml file in Host A

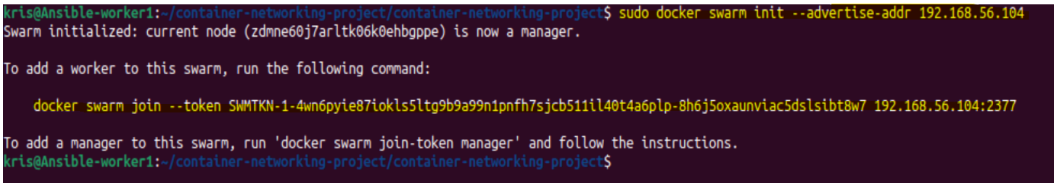
```
version: '3.8'

services:
  # Database Tier
  redis:
    image: redis:alpine
    networks:
      - app-overlay
    deploy:
      resources:
        limits:
          memory: 128M
      placement:
        constraints:
          - node.role == manager

  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: orders
    networks:
      - app-overlay
    deploy:
      resources:
        limits:
          memory: 256M
      placement:
        constraints:
          - node.role == manager
```

```
# Backend Tier
service-registry:
  image: myapp-registry:latest
  networks:
    - app-overlay
  ports:
    - "8501:8500"
  deploy:
    resources:
      limits:
        memory: 128M

product-service:
  image: myapp-product:latest
  environment:
    - SERVICE_IP=product-service
    - SERVICE_PORT=5000
    - SERVICE_REGISTRY=http://service-registry:8500
    - REDIS_HOST=redis
  depends_on:
    - service-registry
    - redis
  networks:
    - app-overlay
  deploy:
    replicas: 3
    resources:
      limits:
        cpus: '0.5'
        memory: 128M
    update_config:
      parallelism: 1
      delay: 10s

order-service:
  image: myapp-order:latest
  environment:
    - SERVICE_IP=order-service
    - SERVICE_PORT=5000
    - SERVICE_REGISTRY=http://service-registry:8500
    - DB_HOST=postgres
  depends_on:
    - service-registry
    - postgres
  networks:
    - app-overlay
  deploy:
    resources:
      limits:
        cpus: '0.5'
        memory: 128M
```

```
  # Frontend Tier
  api-gateway:
    image: myapp-gateway:latest
    environment:
      - SERVICE_IP=api-gateway
      - SERVICE_PORT=3000
      - SERVICE_REGISTRY=http://service-registry:8500
      - PRODUCT_SERVICE_URLS=http://product-service:5000
      - ORDER_SERVICE=http://order-service:5000
    depends_on:
      - service-registry
      - product-service
      - order-service
    ports:
      - "3000:3000"
    networks:
      - app-overlay
    deploy:
      resources:
        limits:
          memory: 128M

networks:
  app-overlay:
    driver: overlay
    attachable: true
```

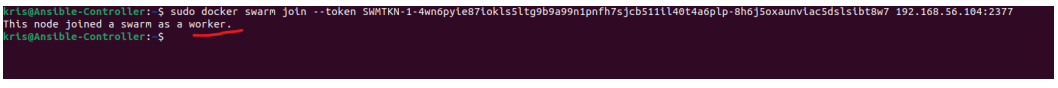
- Initialize swarm on Host A and copy the join token to Host B

  `docker swarm init --advertise-addr 192.168.56.104`



- **Join Worker:** Provide the join token from Host A and run it on Host B.

  `docker swarm join --token SWMTKN-1-4wn6pyie87ioklls5ltg9b9a99n1pnfh7sjcb511il40t4a6plp-8h6j5oxaunviac5dslsibt8w7 192.168.56.104:2377`



- **Create Overlay Network:**

  `docker network create --driver overlay --attachable app-overlay`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker network create --driver overlay --attachable app-overlay
[sudo] password for kris:
3y74dyg5yfto3qmlnlyz73vn3
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

`docker network ls`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker network ls
NETWORK ID     NAME                                    DRIVER    SCOPE
3y74dyg5yfto   app-overlay                             overlay   swarm
96d30e05635b   bridge                                  bridge    local
8c5be7d7e201   container-networking-project_app-network   bridge    local
7fc9c00ed7e4   docker_gwbridge                         bridge    local
055d887958da   host                                    host      local
cx3wdlyf6g3q   ingress                                 overlay   swarm
5a6b0e3e127d   none                                    null      local
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

- Deploy stack on host A

  `docker stack deploy -c docker-compose-swarm.yml myapp`

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker stack deploy -c docker-compose-swarm.yml myapp
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network myapp_app-overlay
Creating service myapp_postgres
Creating service myapp_service-registry
Creating service myapp_product-service
Creating service myapp_order-service
Creating service myapp_api-gateway
Creating service myapp_redis
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Deliverable: Services communicating across hosts

The deployment was verified by checking the service status and task distribution.

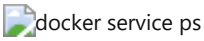
Service Status:

```
sudo docker service ls
# Output showing all services running with full replicas
```

```
kris@Ansible-worker1:~/container-networking-project/container-networking-project$ sudo docker service ls
ID             NAME                   MODE         REPLICAS   IMAGE                   PORTS
96dlppi6niu3   myapp_api-gateway      replicated   1/1        myapp-gateway:latest    *:3000->3000/tcp
ghmkt0gsihq8   myapp_order-service    replicated   1/1        myapp-order:latest
1e8cfthdw1zl   myapp_postgres         replicated   1/1        postgres:15-alpine
0b9yycbeml12   myapp_product-service  replicated   3/3        myapp-product:latest
r34e7s9uz6mx   myapp_redis            replicated   1/1        redis:alpine
9eg11vtopbi1   myapp_service-registry  replicated   1/1        myapp-registry:latest   *:8501->8500/tcp
kris@Ansible-worker1:~/container-networking-project/container-networking-project$
```

Task Distribution: By inspecting individual services, we confirmed that containers are balanced across both `Ansible-worker1` (Manager) and `Ansible-Controller` (Worker).

```
sudo docker service ps myapp_product-service
```



This demonstrates a fully functional multi-host overlay network where microservices can discover and communicate with each other regardless of the physical host they reside on.