Question 2: Rate Limiting and Request Size Limiting with KONG API Gateway

Overview

This guide demonstrates how to implement rate limiting and request size limiting using KONG API Gateway to protect your APIs from abuse and ensure fair resource usage.

Prerequisites

- Docker and Docker Compose installed
- Basic understanding of APIs
- Completed Question 1 (Flask API running)

Step 1: Setup KONG with Docker Compose

Create docker-compose.yml

yaml

```yaml
version: '3.8'

services:
  kong-database:
    image: postgres:13
    environment:
      POSTGRES_USER: kong
      POSTGRES_DB: kong
      POSTGRES_PASSWORD: kongpass
    ports:
      - "5432:5432"
    volumes:
      - kong-data:/var/lib/postgresql/data
    networks:
      - kong-net

  kong-migration:
    image: kong:3.4
    command: kong migrations bootstrap
    environment:
      KONG_DATABASE: postgres
      KONG_PG_HOST: kong-database
      KONG_PG_USER: kong
      KONG_PG_PASSWORD: kongpass
    depends_on:
      - kong-database
    networks:
      - kong-net

  kong:
    image: kong:3.4
    environment:
      KONG_DATABASE: postgres
      KONG_PG_HOST: kong-database
      KONG_PG_USER: kong
      KONG_PG_PASSWORD: kongpass
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_ADMIN_LISTEN: 0.0.0.0:8001
    ports:
      - "8000:8000"  # Proxy port
      - "8443:8443"  # Proxy SSL port
      - "8001:8001"  # Admin API port
      - "8444:8444"  # Admin API SSL port
    depends_on:
      - kong-database
      - kong-migration
    networks:
      - kong-net

volumes:
  kong-data:

networks:
  kong-net:
    driver: bridge
```

Start KONG

bash

```bash
# Start all services
docker-compose up -d

# Check if KONG is running
curl -i http://localhost:8001/

# You should see KONG's admin API response
```

Step 2: Register Your Flask API as a Service

bash

```bash
# Add your Flask API as a service in KONG
curl -i -X POST http://localhost:8001/services/ \
  --data "name=plagiarism-api" \
  --data "url=http://host.docker.internal:5000"

# Expected response: HTTP/1.1 201 Created with service details
```

Note: Use `host.docker.internal` to access localhost from Docker container.

Step 3: Create a Route

bash

```bash
# Create a route to access the service
curl -i -X POST http://localhost:8001/services/plagiarism-api/routes \
  --data "name=plagiarism-route" \
  --data "paths[]=/api/plagiarism"

# Now your API is accessible at: http://localhost:8000/api/plagiarism/check
```

Step 4: Configure Rate Limiting

Option 1: Basic Rate Limiting (Simple)

bash

```bash
# Add rate limiting plugin (5 requests per minute)
curl -i -X POST http://localhost:8001/services/plagiarism-api/plugins \
  --data "name=rate-limiting" \
  --data "config.minute=5" \
  --data "config.policy=local"

# This limits each client to 5 requests per minute
```

Option 2: Advanced Rate Limiting

bash

```bash
# More granular control
curl -i -X POST http://localhost:8001/services/plagiarism-api/plugins \
  --data "name=rate-limiting" \
  --data "config.second=2" \
  --data "config.minute=10" \
  --data "config.hour=100" \
  --data "config.policy=local" \
  --data "config.fault_tolerant=true"
```

Rate Limits Explained:

- `second=2`: Max 2 requests per second
- `minute=10`: Max 10 requests per minute
- `hour=100`: Max 100 requests per hour
- First limit hit triggers the restriction

Step 5: Configure Request Size Limiting

bash

```bash
# Limit request body size to 5MB
curl -i -X POST http://localhost:8001/services/plagiarism-api/plugins \
  --data "name=request-size-limiting" \
  --data "config.allowed_payload_size=5" \
  --data "config.size_unit=megabytes"
```

Configuration Options:

- `allowed_payload_size`: Maximum size
- `size_unit`: `megabytes`, `kilobytes`, or `bytes`

Step 6: Test Rate Limiting

Create a test script: test_rate_limit.py

python

```python
import requests
import time

API_URL = "http://localhost:8000/api/plagiarism/check"

# Test rate limiting
print("Testing Rate Limiting...")
print("=" * 50)

for i in range(15):
    try:
        response = requests.get(API_URL)
        print(f"Request {i+1}: Status {response.status_code}")

        if response.status_code == 429:
            print(f"  ⚠️  Rate limit exceeded!")
            print(f"  Headers: {dict(response.headers)}")
            break

        time.sleep(0.5)
    except Exception as e:
        print(f"  Error: {e}")

print("=" * 50)
```

Expected Output:

```
Request 1: Status 200
Request 2: Status 200
Request 3: Status 200
Request 4: Status 200
Request 5: Status 200
Request 6: Status 429
  ⚠️  Rate limit exceeded!
  Headers: {'X-RateLimit-Limit-Minute': '5', 'X-RateLimit-Remaining-Minute': '0'}
```

Step 7: Test Request Size Limiting

Create test_size_limit.py

python

```python
import requests

API_URL = "http://localhost:8000/api/plagiarism/check"

# Create a large file (6MB - exceeds 5MB limit)
large_content = "A" * (6 * 1024 * 1024)  # 6MB of 'A's

with open('large_file.txt', 'w') as f:
    f.write(large_content)

# Try to upload large file
print("Testing Request Size Limiting...")
print("=" * 50)

try:
    files = {
        'original': open('large_file.txt', 'rb'),
        'submission': open('large_file.txt', 'rb')
    }

    response = requests.post(API_URL, files=files)
    print(f"Status Code: {response.status_code}")

    if response.status_code == 413:
        print("✅ Request size limit working!")
        print(f"Response: {response.text}")
    else:
        print(f"Response: {response.json()}")

except Exception as e:
    print(f"Error: {e}")

print("=" * 50)
```

Step 8: Monitor and Manage

View All Plugins

bash

```bash
curl -i http://localhost:8001/plugins
```

View Service Configuration

bash

```bash
curl -i http://localhost:8001/services/plagiarism-api
```

View Routes

bash

```bash
curl -i http://localhost:8001/routes
```

Delete a Plugin (if needed)

bash

```bash
# Get plugin ID first
curl http://localhost:8001/plugins

# Delete using ID
curl -i -X DELETE http://localhost:8001/plugins/{plugin-id}
```

Step 9: Advanced Configuration

Consumer-Based Rate Limiting

bash

```bash
# Create a consumer
curl -i -X POST http://localhost:8001/consumers \
  --data "username=student1"

# Create API key for consumer
curl -i -X POST http://localhost:8001/consumers/student1/key-auth \
  --data "key=student1-api-key"

# Enable key authentication
curl -i -X POST http://localhost:8001/services/plagiarism-api/plugins \
  --data "name=key-auth"

# Now different rate limits per consumer
curl -i -X POST http://localhost:8001/consumers/student1/plugins \
  --data "name=rate-limiting" \
  --data "config.minute=20"
```

Screenshots to Capture

Screenshot 1: KONG Admin API Running

bash

```bash
curl http://localhost:8001/ | jq
```

Screenshot 2: Service Registration

bash

```bash
curl http://localhost:8001/services/plagiarism-api | jq
```

Screenshot 3: Rate Limiting Plugin Active

bash

```bash
curl http://localhost:8001/services/plagiarism-api/plugins | jq
```

Screenshot 4: Rate Limit Exceeded (429 Response)

- Run the test_rate_limit.py script
- Capture the 429 response with headers

Screenshot 5: Request Size Limit (413 Response)

- Run the test_size_limit.py script
- Capture the 413 response

Screenshot 6: Successful Request Within Limits

- Show a successful API call with rate limit headers

Verification Checklist

✅ KONG is running (docker-compose ps) ✅ Service registered in KONG ✅ Route created and accessible ✅ Rate limiting plugin configured ✅ Request size limiting plugin configured ✅ Rate limit tested and working (429 response) ✅ Size limit tested and working (413 response) ✅ Headers showing rate limit information

Troubleshooting

Issue: Cannot connect to Flask API from KONG

Solution: Use `host.docker.internal` instead of `localhost` in service URL

Issue: 404 Not Found

Solution: Check route configuration: `curl http://localhost:8001/routes`

Issue: Rate limiting not working

Solution:

- Verify plugin is enabled: `curl http://localhost:8001/plugins`
- Check plugin configuration
- Try different client IPs

Issue: Docker containers not starting

Solution:

- Check logs: `docker-compose logs kong`
- Ensure ports 8000, 8001, 5432 are not in use
- Run: `docker-compose down -v` and restart

Cleanup

bash

```
# Stop all services
docker-compose down

# Remove volumes (careful: deletes all data)
docker-compose down -v

# Remove images
docker rmi kong:3.4 postgres:13
```

Summary

You have successfully:

1. ✅ Set up KONG API Gateway with Docker
2. ✅ Registered your Flask API as a service
3. ✅ Configured rate limiting (requests per time period)
4. ✅ Configured request size limiting (max payload size)
5. ✅ Tested both limits with verification scripts
6. ✅ Captured evidence screenshots

This implementation protects your API from:

- **Abuse**: Rate limiting prevents excessive requests
- **DoS attacks**: Request size limits prevent memory exhaustion
- **Resource overuse**: Fair usage across all clients