Distributed Systems Exercise 9

Riccardo Benedetti
Leon Harz

MicroService 1: Order Service

Features:

- **Endpoints:**
  - GET /orders/<order_id>: Retrieve an order by ID
  - POST /orders: Create a new order
- **Key Functionalities:**
  - Stores orders in an in-memory database (orders dictionary)
  - Validates the existence of a user by calling the **User Service** (/users/<user_id>)
  - Creates a new order with:
    - **ID**: Automatically generated
    - **User ID**: Referenced from the User Service
    - **Items**: List of order items
    - **Total**: Order total

Code Highlights:

- External dependency check via requests.get to the User Service
- Example of structured API responses:
  - **Success**: HTTP 201 with the order details
  - **Error**: HTTP 404 if the user or order is not found

MicroService 2:

Features:

- **Endpoints:**
  - GET /users/<user_id>: Retrieve a user by ID
  - POST /users: Create a new user
- **Key Functionalities:**
  - Stores users in an in-memory database (users dictionary)
  - Creates a new user with:
    - **ID**: Automatically generated
    - **Name**: User's name
    - **Email**: User's email address

Code Highlights:

- Example of simple resource management:
  - **Success**: HTTP 201 with user details
  - **Error**: HTTP 404 if the user is not found

Workflow/Interaction

1. **Order Creation Workflow**:
    - **Step 1**: User Service validates the user
    - **Step 2**: Order Service creates an order if the user exists
    - **Step 3**: Order Service returns the newly created order
2. **Microservice Interaction**:
    - The **Order Service** depends on the **User Service** for validation, creating a modular design where services are loosely coupled

Monitoring

1. **Application-Level Monitoring**
   - Use tools like **Prometheus** and **Grafana**:
     - Collect metrics (e.g., request counts, response times, error rates)
     - Visualize metrics through dashboards
2. **Log Management**
   - Tools like **ELK Stack (Elasticsearch, Logstash, Kibana)** or **Fluentd**:
     - Aggregate and analyze application logs
     - Use structured logging (e.g., json format) for easy parsing
   - Libraries:
     - Use logging or flask-logging for capturing logs
3. **Distributed Tracing**
   - Use **Jaeger** or **Zipkin**:
     - Trace requests across microservices
     - Identify bottlenecks in service interactions
   - Libraries:
     - Integrate with Flask using **OpenTelemetry**
4. **Container-Level Monitoring**
   - Use **Docker Monitoring Tools** like:
     - **cAdvisor**: Provides resource usage metrics for containers
     - **Docker Stats**: Built-in monitoring for CPU, memory, and I/O

Scaling

1. **Vertical Scaling**
   - Modify Docker resource limits in deployment configurations
2. **Horizontal Scaling**
   - Run multiple instances of each service and distribute traffic using a load balancer
3. **Docker Compose Scaling**
   - Add `scale` options in `docker-compose.yml`
4. **Kubernetes**
   - Deploy the services in a **Kubernetes Cluster**:
   - Use **Deployments** to manage replicas of each service
5. **Load Balancing**
   a. Use tools like **NGINX**, **Traefik**, or **Kubernetes Services** to distribute requests across instances.