SOA Lab Exercises

Exercise 1: Overview of Service-Oriented Architecture

Objective:

Introduce students to SOA basics by creating a simple, containerized web service that clients can consume.

Steps / Tasks

1. **Set Up a Basic Service**
   - Choose a framework (Flask/Python, Spring Boot/Java, Express/Node.js).
   - Create a simple endpoint (e.g., `/products` or `/users`) that returns or manipulates data (e.g., CRUD operations in-memory or in a small database like SQLite).
2. **Containerization**
   - Create a `Dockerfile` for your service.
   - Use Docker to containerize and run your service locally (e.g., `docker build`, `docker run`).
3. **Client Consumption**
   - Write a simple client script or another microservice to call your web service.
   - Demonstrate basic operations (GET, POST, PUT, DELETE).
4. **Documentation and Testing**
   - Produce a small `OpenAPI/Swagger` specification to define your API.
   - Use a tool like **Postman** or **curl** to test endpoints.

Key Learning Points:

- Foundational SOA concepts (service exposure, discoverability, loose coupling).
- Introduction to containerization for easy deployment and scaling.

Exercise 2: Principles and Concepts of SOA

Objective:

Implement a loosely coupled service architecture using **asynchronous messaging**.

Steps / Tasks

1. **Choose a Message Broker**
    - Use **RabbitMQ**, **Apache Kafka**, or **ActiveMQ**.
    - Explain how messaging decouples the producer from the consumer.
2. **Publisher Service**
    - Create a simple service that sends messages (e.g., JSON payload) to the broker whenever an event occurs (e.g., new order placed, data update).
    - Containerize it if desired (using Docker) for consistency.
3. **Consumer Service**
    - Implement a separate service that subscribes to the broker and processes messages independently (e.g., logs them, stores them in a DB, triggers a workflow).
    - Ensure no direct coupling between publisher and consumer beyond the message format.
4. **Observability**
    - Introduce logging and monitoring for your publisher and consumer services (e.g., using **Elastic Stack**, **Prometheus**, or built-in broker metrics).

Key Learning Points:

- Asynchronous communication for **loose coupling**.
- Event-driven design principles and decoupled service interactions.
- Understanding of **microservices** patterns.

Exercise 3: Contemporary Trends in SOA

Objective:

Explore Serverless Computing and integrate it into an existing SOA/microservice ecosystem.

Steps / Tasks

1. Set Up a Simple Serverless Function
    - Pick an open source serverless platform like (**OpenFaaS, Apache OpenWhisk, Knative, Kubeless, Fission, or KEDA**).
    - Create a function that performs a specific task, e.g., image resizing, simple text processing, or a quick calculation.
2. Expose the Function as a REST Endpoint
    - Use API Gateway (AWS), Azure Function's HTTP trigger, or Cloud Functions' HTTPS endpoint to make your function externally callable.
    - Verify your function can accept inputs and return outputs.
3. Integration with Other Services
    - Invoke your serverless function from a previously created microservice or a simple client.
    - Demonstrate that the function can be part of a broader SOA. For example, upload an image via a REST service, which triggers the serverless function to resize it and then store the result in a storage service.
4. Observability and Cost Monitoring (Optional Enhancement)
    - Show how to monitor invocation counts, latency and cost metrics for your serverless function.
    - Highlight the ephemeral nature of serverless (cold starts, concurrency limits, etc.).

Key Learning Points:

- Basics of Function-as-a-Service (FaaS).
- Serverless integration with existing services for scalability and event-driven operations.

- Challenges like cold starts, limited runtime environment, debugging in a serverless context.

Exercise 4: Artificial Intelligence (AI) and Machine Learning (ML) in SOA:

Objective:

Build and integrate a **simple AI-driven service** (e.g., classification, sentiment analysis, or basic prediction) within an SOA-based architecture.

Steps / Tasks

1. **Develop/Obtain a ML Model**
    - Use a small classification model (e.g., scikit-learn or TensorFlow).
    - Pre-train or load a pretrained model (e.g., for text sentiment or an image classification dataset like MNIST).
2. **Build a Service for ML Inference**
    - Wrap the model in a REST endpoint (Flask, FastAPI, or any framework).
    - Accept input data (text, image, numeric features) and return inference results.
3. **Containerize**
    - Package the model service with **Docker** for easy deployment.
    - Show how the model can be scaled independently, if needed.
4. **Integration and Testing**
    - Integrate your AI service with a front-end client or another microservice. For instance, the client sends text or image data and the AI service returns a prediction.
    - Demonstrate how updates to the model (newer version, better accuracy) can be swapped in with minimal disruption to the rest of the SOA.
5. **Expand with MLOps**
    - Briefly mention or demonstrate how to track model versions, use a model registry (e.g., MLflow).
    - Automated testing: ensure new model versions do not break the interface or degrade performance.

Key Learning Points:

- Basic AI model serving in a **service-oriented** environment.
- Handling model versioning, data input/output formats and performance considerations.
- Implementation of MLOps