Stan-Soponaru Ioan-Alexandru

FILS 1241A

**Project Number 4: Project Management Tool**

**a) Investigate, Evaluate, and Compare Architectural Styles: Monolithic, Microservices, and Containerized**

**1. Monolithic Architecture**

**Description:** A single codebase where all components (e.g., UI, business logic, database access) are tightly coupled and deployed together.

**Steps to Design:**

* Identify core modules (e.g., task management, reporting).
* Implement modules as part of the same application with shared resources.
* Use a unified database to store all data.
* Deploy the application as a single package (e.g., WAR, JAR).

**Components and Interactions:**

* Single-tier application.
* Communication is direct through method calls.

**2. Microservices Architecture**

**Description:** Independent, loosely coupled services that communicate over the network.

**Steps to Design:**

* Decompose the application into services (e.g., task management, reporting, user management).
* Use REST or messaging for inter-service communication.
* Assign a database per service to maintain autonomy.
* Implement service discovery (e.g., Eureka) and API Gateway.

**Components and Interactions:**

* Independent services for task management and reporting.
* Communication via REST endpoints or messaging queues.

**3. Containerized Architecture**

**Description:** Deploy the application or services in lightweight, isolated containers.

**Steps to Design:**

* Package each application (monolithic or microservices) in a container.
* Use tools like Docker to containerize services.
* Orchestrate containers with Kubernetes for scaling and management.

**Components and Interactions:**

* Services in isolated containers.
* Container orchestration for scaling and fault tolerance.

**b) Evaluation of Architectures**

**1. Team Collaboration**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Advantages** | **Disadvantages** |
| **Monolithic** | Easier collaboration with a single codebase. | Difficult for large teams due to shared dependencies. |
| **Microservices** | Teams work independently on services. | Coordination is needed for integration. |
| **Containerized** | Clear separation via containers. | Requires DevOps expertise. |

**2. Data Consistency**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Advantages** | **Disadvantages** |
| **Monolithic** | Centralized database ensures consistency. | Scalability challenges for large datasets. |
| **Microservices** | Service-specific databases avoid conflicts. | Complex to ensure consistency across services. |
| **Containerized** | Supports both centralized and distributed databases. | Requires careful database orchestration. |

**3. Deployment Efficiency**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Advantages** | **Disadvantages** |
| **Monolithic** | Simple deployment process. | Downtime during updates. |
| **Microservices** | Independent deployment of services. | Requires CI/CD pipelines. |
| **Containerized** | Rapid deployment with Docker/Kubernetes. | Higher initial setup complexity. |

**c) User Experience Impact**

**Monolithic**

**Strengths:** Fast response due to in-memory operations and single-process execution.

**Weaknesses**: Performance degrades as user load increases.

**Microservices**

**Strengths**: Scalable for high traffic, ensuring consistent user experience.

**Weaknesses**: Latency due to inter-service communication.

**Containerized**

**Strengths**: Reliable performance due to container orchestration.

**Weaknesses**: Initial deployment might face minor delays due to container startup time

**d) Transaction Processing**

**Monolithic**

**Strengths:** ACID compliance is easier with a single database.

**Weaknesses**: Limited concurrency handling.

**Microservices**

**Strengths:** Better concurrent task handling via distributed services.

**Weaknesses**: Complex transactions spanning multiple services.

**Containerized**

**Strengths:** Combines microservices benefits with container scalability.

**Weaknesses**: Relies on orchestration for optimal concurrency.

**Conclusion**

Monolithic is simple but not scalable.

Microservices offer scalability but introduce complexity.

Containerized architecture combines scalability and reliability, making it ideal for modern applications.