Architecture Decision Record (ADR) - Microservice Architecture Framework

Decision Summary

Utilize microservice-based architecture over event-based architecture for the Fish Watch system.

Context

The Fish Watch system requires a scalable and resilient architecture to support its functionalities, including data collection, processing, analytics, and real-time monitoring. The choice between microservice-based architecture and event-based architecture is crucial in determining the system's scalability, flexibility, and maintainability.

Decision Drivers

1. **Scalability**: The architecture must be able to scale horizontally to accommodate the growing volume of data and users in the Fish Watch system.
2. **Flexibility**: The architecture should support independent deployment and evolution of individual components to facilitate agility and innovation.
3. **Resilience**: The system must be resilient to failures and able to recover quickly to ensure uninterrupted operation.
4. **Maintainability**: The architecture should be easy to maintain, debug, and update to support ongoing development and enhancements.

Considered Options

1. **Microservice-Based Architecture**: Decompose the system into loosely coupled, independently deployable microservices that communicate via lightweight protocols.
2. **Event-Based Architecture**: Implement the system as a collection of event-driven components that react to events and trigger actions based on event streams.

Decision Outcome

We choose to implement a microservice-based architecture for the Fish Watch system.

Pros and Cons of the Selected Option

Pros

- **Scalability**: Microservices allow for horizontal scaling of individual components, enabling the system to handle increased loads efficiently.
- **Flexibility**: Decoupled microservices facilitate independent development, deployment, and scaling, promoting agility and innovation.
- **Resilience**: Fault isolation and redundancy in microservices contribute to system resilience, minimizing the impact of failures on overall system performance.
- **Maintainability**: Microservices promote modularity and encapsulation, making it easier to maintain, debug, and update individual components without affecting the entire system.

Cons

- **Complexity**: Managing a distributed system composed of multiple microservices introduces complexity in deployment, monitoring, and orchestration.
- **Communication Overhead**: Inter-service communication may incur latency and overhead, requiring careful design and optimization to ensure performance.

Risks and Mitigations

- **Service Communication**: Implement robust communication patterns and protocols to mitigate potential issues related to inter-service communication latency and reliability.
- **Monitoring and Management**: Utilize comprehensive monitoring and management tools to monitor the health and performance of microservices and detect issues early for timely resolution.

Consequences

The decision to adopt a microservice-based architecture for the Fish Watch system provides scalability, flexibility, resilience, and maintainability benefits. It enables the system to handle increased loads, support independent development and deployment, recover quickly from failures, and facilitate ongoing maintenance and updates. However, it also introduces complexity in managing a distributed system and requires careful design and monitoring to ensure optimal performance and reliability.