ADR ID

ADR-FF-001

Assumption

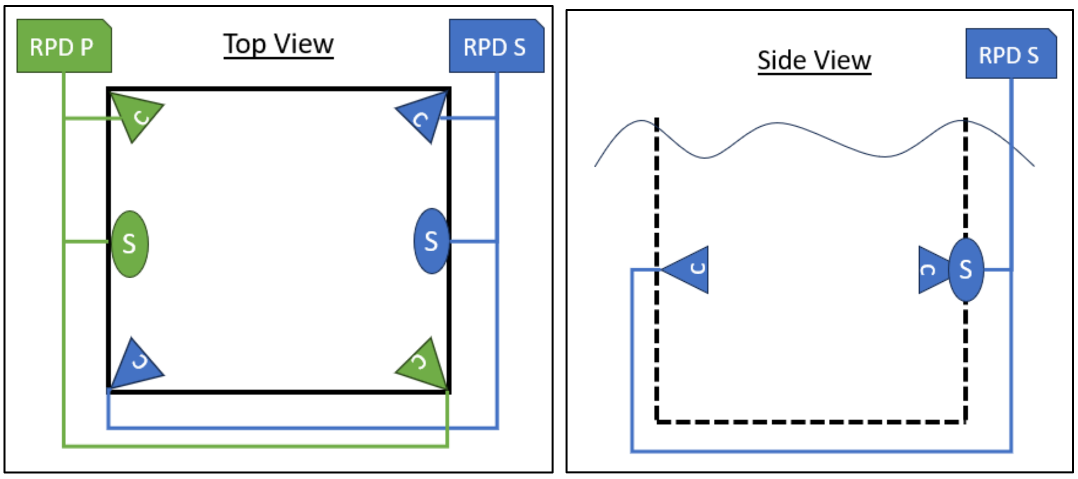
1. Each enclosure is equipped with the below equipment:
    a. 4 x Submersible Cameras – It captures and stream the image of the fishes for "fish"-ual recognition, health, and lifecycle of a fish, from various sides of the enclosure.
    b. 2 x Submersible Multi-Sensors – It captures the water quality information.
    c. 2 x Raspberry Pi Device (RPD) – IoT equipment at edge with AWS IoT Greengrass package and each has physical connection with two cameras and one multi-sensor.
2. Availability of the power source for the Raspberry Pi Devise is not an issue.

Decision and Rationale

We will use a event-driven architecture to build our application.

Context

1. The cameras and sensor location and its connection to the RPD as below to maximized vision field for capturing images:



2. RDP P denotes RDP Primary and RDP S denotes RDP Secondary.
3. USB-C connection in each RDP for local data retrieval, for limited-to-none network connection fish farm location.

Consequences

Positive	Negative	Neutral
1. High availability at edge 2. Modular; thus, scalable.	1. High installation cost 2. Affect operation cost with two mobile subscriptions for each enclosure.	Same software package to be install at all RDP

3. Independent to each other – each RDP with its own network connection.		

Security

At-Rest

1. Encrypted flash memory in the RDP.

In-Transit

1.

Redundancy

1. Bluetooth connection in each RDP for isAlive() ping function between the two RDP.
2. Only one RDP is active at a time – RDP Primary (RDP P).
3. The RDP Secondary (RDP S) shall be activated in an event RDP P not available.

[Open]

Architecture Decision Record

A document that captures an important architectural decision made along with its context and consequences.

Decision

We will use a event-driven architecture to build our application.

Context

We need to develop a scalable, reliable, and maintainable application that can handle multiple types of requests from different sources. We also want to enable continuous delivery and deployment of new features and bug fixes.

Consequences

Positive	**Negative**	**Neutral**
Improved modularity and cohesion of the code base. Increased fault tolerance and resilience of the system. Faster and easier testing and deployment of individual services.	Increased complexity and overhead of managing multiple services. Potential performance and latency issues due to network communication. Need for proper service discovery, monitoring, and coordination mechanisms.	Requires a different set of skills and tools than a monolithic architecture. May not be suitable for some types of applications that have high cohesion and low coupling. May introduce some trade-offs between consistency and availability of the data.

Decision Rationale

We will use AWS Device Gateway

For Front-end Layer:

1. AWS AppSync:
    - **Real-time Data Updates**: AppSync provides native support for real-time subscriptions using GraphQL, which aligns with the requirement for real-time data updates in the application. This feature allows clients to receive updates instantly, so action can be taken on-time to prevent loss or death of the fish.
    - **Offline Functionality:** AppSync, when coupled with AWS Amplify DataStore, facilitates seamless offline functionality by synchronizing data between clients and the backend, ensuring that the application remains functional even in offline scenarios. This is good when the internet connection is not good at the farm.
2. Amazon DynamoDB:
    - **Scalability and Performance:** DynamoDB's managed NoSQL database service offers seamless scalability and high performance, making it suitable for applications with a large user base and varying workload demands.
    - **Real-time Data Access**: DynamoDB's integration with AWS AppSync allows for real-time data access and updates, enabling efficient handling of read and write operations with minimal latency.
3. Amazon Cognito:
    - **Authentication and Authorization:** Cognito offers robust authentication and authorization services, allowing seamless integration with various identity providers and ensuring secure access to application resources. It simplifies user management tasks such as registration, authentication, and user profile management, enhancing the overall security posture of the application.
4. Amazon QuickSight:
    - **Native Integration and Ease of Use:** QuickSight seamlessly integrates with AWS services like DynamoDB, offering a user-friendly interface for developers to create interactive dashboards effortlessly.
    - **Embedding Capabilities for Unified User Experience:** QuickSight's embedding features allow web and mobile clients to integrate dashboards directly into their applications, providing users with easy access to analytics without leaving the app interface.

Context

We need to develop a scalable, reliable, and maintainable application that can handle multiple types of requests from different sources. We also want to enable continuous delivery and deployment of new features and bug fixes.

Consequences

Positive	Negative	Neutral
Improved modularity and cohesion of the code base. Increased fault tolerance and resilience of the system. Faster and easier testing and deployment of individual services.	Increased complexity and overhead of managing multiple services. Potential performance and latency issues due to network communication. Need for proper service discovery, monitoring, and coordination mechanisms.	Requires a different set of skills and tools than a monolithic architecture. May not be suitable for some types of applications that have high cohesion and low coupling. May introduce some trade-offs between consistency and availability of the data.