# Architecture Design Record

Team Name: Kata-Kata Manis

## 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

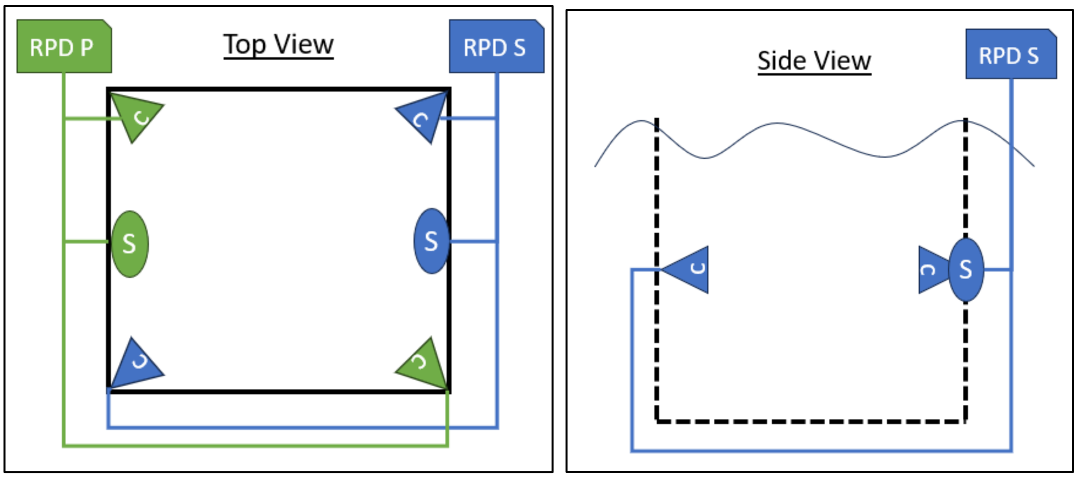
AWS IoT Greengrass Core package is selected at the fish farm – a.k.a edge – because it can filter and aggregate device data.

The rationale of the selected AWS IoT Greengrass Core component as follows:

1. **Greengrass Message Broker**: Handles the communication protocol (MQTT) between the input devices (cameras and sensor) and Greengrass Stream Manager before uploading it to Greengrass Core Device Gateway at cloud.
2. **Greengrass Stream Manager**: Handles data transformation of the raw data from cameras and sensor prior sending it to the AWS cloud and manages data uploading to the AWS cloud in the limited or intermittent connectivity happening at the fish farm.
3. **Greengrass Nucleus**: Besides handling inter-component communication, it also handles the pushed settings deployment from Greengrass Core Device Gateway at cloud to RPD.

## Context

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



2. RPD P denotes RPD Primary and RPD S denotes RDP Secondary.
3. USB-C connection in each RDP for local data retrieval, for limited-to-none network connection fish farm location.
4. Bluetooth Low Energy (BLE) connection in each RPD for isAlive() ping function between the two RDP.

## Consequences

| **Positive** | **Negative** | **Neutral** |
|---|---|---|
| 1. High availability at edge<br>2. Modular; thus, scalable.<br>3. Independent to each other – each RDP with its own network connection.<br>4. Similar setup can be applied for other type of livestock (e.g., cattle). | 1. High installation cost<br>2. Affect operation cost with two mobile subscriptions for each enclosure. | Same software package to be install at all RDP |

## Security

At-Rest

1. Uses Password Protect Volume (LUKS) to encrypt flash memory in the RPD. (ref)

In-Transit

1. Uses TLS for communication with AWS services.
2. Uses Media Transport Protocol (MTP) for USB-C communication.
3. Uses AES-CCM (Advanced Encryption Standard with Cipher Block Chaining Message Authentication Code) in the BLE's Security Manager for communication between RPD and handheld data retrieval device for limited-to-none network connection fish farm location.

## Redundancy

1. Only one RPD is active at a time – RPD Primary (RPD P).
2. The RPD Secondary (RPD S) shall be activated in an event RPD P not available.

## ADR ID

ADR-CL-001

## Context

The edge processing data needs to be transferred to the cloud when there is internet connectivity. This streams of IoT data needs to be processed from many various sources in a fast and efficient manner.

## Assumptions

1. **User Base**: We assume that the application will have a significant user base which will upload at their earlier availability of internet connection
2. **Connectivity**: Low-bandwidth connectivity is the expectation since all sites are fish farms in the middle of rivers and lakes.
3. **Cost**: We assume that the cost of using the selected AWS services aligns with the budgetary constraints of the project. Cost optimization strategies may be required to manage expenses effectively, especially as the application scales.

## Decision and Rationale

For managing the IoT data, we use AWS services consisting of Device gateway, IoT Sitewise, IoT Events and SNS.

The rationale of the selected services as follows:

1. Device Gateway:
    - **Real-time Data Updates**: The Device Gateway serves as the entry point for IoT devices connecting to AWS. The Device Gateway manages all active device connections and implements semantics for multiple protocols to verify that devices can securely and efficiently communicate with AWS IoT Core. Currently the Device Gateway supports the MQTT, WebSockets, and HTTPS protocols.
    - **Low latency device management:** For devices that connect using MQTT or WebSockets the Device Gateway will maintain long lived, bidirectional connections, helping these devices send and receive messages at any time with low latency. The Device Gateway is fully managed and scales automatically to support over a billion devices without requiring you to manage any infrastructure.
2. IoT Sitewise:

  - **Scalability and Performance**: AWS IoT SiteWise is a managed service that makes it easy to collect, store, organize and monitor data from IoT equipment at scale to help make better, data-driven decisions.

3. IoT Events:
   - **Authentication and Authorization:** AWS IoT Events makes it possible to easily and cost effectively detect events system-wide and respond with appropriate actions to drive results such as optimizing manufacturing efficiency or improving production quality.
4. SNS:
   - **Communication** Industrial-level managed service that provides message delivery from publishers to subscribers. This is configured to raise alarms for immediate action by receivers.

## Consequences

| | |
|---|---|
| **Positive** | 1. **Scalability and Performance**: <br>• The architecture leverages scalable AWS services like IoT Sitewise and IoT Events which are purpose-built for IoT devices and application, ensuring the infrastructure can handle increasing user loads and data volumes effectively.<br>2. **Real-time Updates and Offline Functionality**:<br>• Users benefit from seamless offline functionality, enhancing their experience and productivity, especially in scenarios with intermittent connectivity. |
| **Negative** | 1. **Complexity and Learning Curve**:<br>• The architecture involves multiple AWS services and technologies, which may increase complexity and require additional time and resources for development and maintenance.<br>2. **Cost Considerations**:<br>• While AWS offers a pay-as-you-go pricing model, the usage of multiple AWS services may incur costs that need to be carefully monitored and optimized to align with budgetary constraints. |

## Security

1. **Authentication and Authorization**:
   - Each connected device or client must have a credential to interact with AWS IoT.
2. **Secure Communication**:
   - All traffic to and from AWS IoT is sent securely over Transport Layer Security (TLS).

## Redundancy Considerations

1. **Multi-AZ Deployment**:
    - Deploy these IoT cloud components across multiple Availability Zones (AZs) within the chosen AWS region. This provides resilience against AZ failures and ensures continuous availability of services in the event of AZ outages.

## ADR ID

ADR-CL-002

## Context

The livestock is to be monitored for any illness or signs of ill health. Therefore, the is a need for industrial level cloud monitoring solution that can analyse this over period of time and share alerts regarding crucial livestock matters.

## Assumptions

1. **Data**: We assume large dataset are available for analysis
2. **Model**: There is a clear documentation of livestock illness that can be easily defined and applied to within context of model
3. **Cost**: We assume that the cost of using the selected AWS services aligns with the budgetary constraints of the project. Cost optimization strategies may be required to manage expenses effectively, especially as the application scales.

## Decision and Rationale

For detection of livestock illness, we are proposing AWS Lookout for Vision services.

The rationale of the selected services as follows:

1. **Dedicated ML Engine**: The service uses machine learning (ML) to see and understand images from any camera as a person would, but with an even higher degree of accuracy and at a much larger scale.

2. **Ease of Use**: Amazon Lookout for Vision is fully managed and comes with anomaly detection techniques for defect detection tasks so that users don’t have to invest time and resources on creating a deep learning pipeline. This can be purposed into livestock illness detection which is industrial grade.

## Consequences

| | |
|---|---|
| **Positive** | 1. **Scalability and Performance**:<br>• The ML architecture leverages scalable AWS services ensuring the infrastructure can handle big data volumes effectively.<br>• |
| **Negative** | 1. **Cost Considerations**:<br>• While AWS offers a pay-as-you-go pricing model, the usage of ML tools may incur costs that need to be carefully monitored and optimized to align with budgetary constraints. |

## Security

1. **Data Privacy**:
    - Any content processed by Amazon Lookout for Vision is encrypted and stored at rest in the AWS region where you are using Amazon Lookout for Vision.
2. **Secure Communication**:
    - All traffic to and from AWS Lookout for Vision is sent securely over Transport Layer Security (TLS).

## ADR ID

ADR-CL-003

## Context

Processing of large datasets that comes from many sources require dedicated solution to process and store them. This can be managed for reference and future use especially on timescale analysis.

## Assumptions

1. **Data Collection**: Data collection mechanism is clearly defined prior to processing.
2. **Data Type**: Only text and picture data are collected.
3. **Cost**: We assume that the cost of using the selected AWS services aligns with the budgetary constraints of the project. Cost optimization strategies may be required to manage expenses effectively, especially as the application scales.

## Decision and Rationale

For further review and analysis of livestock data, we are proposing combination of AWS cloud services consisting of AWS Kinesis Data Stream, AWS Datalake and AWS Lambda

1. AWS Kinesis Data Stream:
   - **Big Data Ingestion**: AWS Kinesis data stream can Ingest and collect terabytes of data per day from IoT application and service logs and deliver this into data lakes.
2. AWS Datalake:
   - **Big Data Analysis and Storage**: AWS Datalake can manage terabytes of data and perform data analysis at very service that makes it easy to collect, store, organize and monitor data from IoT equipment at scale to help make better, data-driven decisions.
3. AWS Lambda:
   - **Specialized Action**: AWS Lambda can be configured to send specialized alerts and perform actions based on data that is passed through it via either AWS Data Lake or Amazon Kinesis Data Streams. This can be useful for monitoring livestock environmental data.

## Consequences

| | |
|---|---|
| **Positive** | 1. **Scalability and Performance**:<br>• With Amazon Kinesis Data Streams, there are no servers to manage. This is also similar with AWS Datalake and AWS Lambda.<br>2. **Faster time to market**:<br>• With Amazon Datalake, the time to setup is measured in days instead of months |
| **Negative** | 1. **Cost Considerations**:<br>• While AWS offers a pay-as-you-go pricing model, the usage of this big data cloud components may incur costs that need to be carefully monitored and optimized to align with budgetary constraints. |

## Security

1. **Data Privacy**:

    - Any content processed by Amazon Kinesis Data Stream is encrypted and stored at rest in the configured AWS region.

## Redundancy Considerations

- AWS Kinesis Data Streams are synchronously replicated across three Availability Zones (AZs) in an AWS Region

## ADR ID

ADR-APP-001

## Context

We are designing the architecture for a new web and mobile application named FishWatch that requires real-time data updates on fish and farm conditions, offline functionality when no internet connectivity, user authentication and authorization, and analytics and reporting capabilities. We have selected AWS as our cloud provider and need to decide on the specific AWS services to use in our architecture.

## Assumptions

4. **User Base**: We assume that the application will have a significant user base, which necessitates the need for scalable and high-performance AWS services like AppSync and DynamoDB to handle varying workload demands.
5. **Connectivity**: We assume that users will have reliable internet connectivity most of the time, as features like real-time data updates and authentication rely on connectivity to function effectively.
6. **Data Volume**: We assume that the volume of data stored in DynamoDB tables will be manageable within the scalability limits of DynamoDB and the performance requirements of the application. The records regarding the fish and the farm would be fit in DynamoDB without issue.
7. **User Authentication**: We assume that users will primarily authenticate using email & password through Amazon Cognito.
8. **Analytics Usage**: We assume that users will actively use the analytics and reporting features provided by QuickSight to gain insights into the fish and farm conditions. The effectiveness of these features may depend on user engagement and interest in data analysis.
9. **Cost**: We assume that the cost of using the selected AWS services aligns with the budgetary constraints of the project. Cost optimization strategies may be required to manage expenses effectively, especially as the application scales.
10. **Development Resources**: We assume that development resources are available to implement and maintain the architecture effectively. This includes expertise in AWS services, GraphQL, and front-end development for web and mobile platforms.

## Decision and Rationale

For Application layer of FishWatch, we will use AWS services consists of AppSync, DynamoDB, Cognito and QuickSight.

The rationale of the selected services as follows:

3. 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 really useful when the internet connection is not good at the farm, to view data that not needed to be real-time, like the sizing, fish condition, count of fish, etc.
4. 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.
5. 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.
6. 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.

## Consequences

| | |
|---|---|
| **Positive** | 3. **Scalability and Performance**:<br>• The architecture leverages scalable AWS services like AppSync and DynamoDB, ensuring the application can handle increasing user loads and data volumes effectively.<br>4. **Real-time Updates and Offline Functionality**:<br>• Users benefit from real-time data updates and seamless offline functionality, enhancing their experience and productivity, especially in scenarios with intermittent connectivity.<br>5. **Security and User Management**:<br>• Integration with Amazon Cognito provides robust authentication and authorization capabilities, ensuring secure access to application resources and simplifying user management tasks.<br>6. **Analytics and Reporting**:<br>• Integration with Amazon QuickSight enables users to analyze and visualize application data effectively, empowering data-driven decision-making and enhancing overall user experience. |
| **Negative** | 2. **Complexity and Learning Curve**:<br>• The architecture involves multiple AWS services and technologies, which may increase complexity and require additional time and resources for development and maintenance. Developers may need to acquire expertise in AWS services and GraphQL.<br>3. **Cost Considerations**:<br>• While AWS offers a pay-as-you-go pricing model, the usage of multiple AWS services may incur costs that need to be carefully monitored and optimized to align with budgetary constraints. |
| **Neutral** | 1. **Internet Connectivity Dependency**:<br>• The effectiveness of features like real-time updates and authentication may depend on internet connectivity. While this is inherent to the nature of web and mobile applications, it's important to consider the impact on user experience in scenarios with limited connectivity.<br>2. **Data Volume and Usage Patterns**:<br>• The architecture assumes manageable data volumes and usage patterns within the scalability limits of DynamoDB and the performance requirements of the application. As data volumes and usage patterns evolve, adjustments may be necessary to ensure optimal performance and cost-effectiveness. |

## Security

2. **Data Protection and Privacy**:
    - Ensuring the confidentiality, integrity, and availability of data is paramount. Data stored in Amazon DynamoDB should be encrypted both at rest and in transit to protect against unauthorized access.
3. **Authentication and Authorization**:
    - Secure authentication mechanisms must be implemented to verify the identity of users accessing the application. Amazon Cognito should be configured with strong password policies, multi-factor authentication (MFA), and proper user role management to prevent unauthorized access.
4. **Secure Communication**:
    - All communication between the web/mobile clients and backend services should be encrypted using HTTPS/TLS to prevent eavesdropping and man-in-the-middle attacks. Appropriate SSL/TLS configurations and certificate management practices should be followed.
5. **Protection Against Common Web Attacks**:
    - The architecture should incorporate measures to mitigate common web application vulnerabilities such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and others. Input validation, output encoding, and proper error handling should be implemented to prevent these attacks.
6. **Security Logging and Monitoring**:
    - Comprehensive logging and monitoring solutions should be implemented to detect and respond to security incidents in real-time. AWS CloudTrail, Amazon CloudWatch, and other monitoring services can be used to monitor application activity, detect anomalous behavior, and investigate security incidents.
7. **Secure Deployment and Configuration**:
    - Secure deployment practices should be followed to ensure that all AWS services and components are configured securely. This includes regularly updating software, applying security patches, and following least privilege principles to minimize the attack surface.
8. **Secure Third-Party Integrations**:
    - Any third-party integrations, such as QuickSight dashboards or external APIs, should be carefully vetted for security risks. Access controls, data encryption, and secure communication should be enforced when interacting with external services to prevent data leakage or unauthorized access.

## Redundancy Considerations

2. **Multi-AZ Deployment**:

    - Deploy critical AWS services such as AWS AppSync, Amazon DynamoDB, and Amazon Cognito across multiple Availability Zones (AZs) within the chosen AWS region. This provides resilience against AZ failures and ensures continuous availability of services in the event of AZ outages.

3. **Load Balancing**:

    - Implement load balancing for web and mobile clients to distribute traffic across multiple instances or endpoints. AWS Elastic Load Balancing (ELB) can automatically distribute incoming application traffic across multiple targets, enhancing availability and fault tolerance.

4. **Database Replication**:

    - Enable DynamoDB global tables to replicate data across multiple AWS regions for disaster recovery and data locality. DynamoDB global tables provide automatic multi-region replication with eventual consistency, ensuring data availability even in the event of regional outages.

5. **Multi-Region Deployment**:

    - Consider deploying critical components of the architecture, such as AppSync and Cognito, across multiple AWS regions for additional redundancy and disaster recovery capabilities. Multi-region deployments ensure resilience against regional failures and improve overall fault tolerance.

6. **Stateless Application Components**:

    - Design application components to be stateless whenever possible. Stateless components can be easily replicated and scaled horizontally, improving availability and fault tolerance.

7. **Automated Failover and Recovery**:

    - Implement automated failover and recovery mechanisms using AWS services such as Amazon Route 53 for DNS failover, AWS Lambda for automated recovery actions, and AWS CloudWatch Alarms for monitoring and alerting. Automated failover reduces downtime and ensures seamless recovery in case of failures.

8. **Data Backups and Disaster Recovery**:

    - Regularly backup application data stored in DynamoDB using automated backup and restore features. Implement disaster recovery plans to restore data and services quickly in the event of catastrophic failures or data loss incidents.

End of Document