Project Documentation: AI-Powered Fruit Disease Detection

1. Executive Summary & Project Goal

For our AWS re/Start graduation project, I developed an AI-powered system to solve a practical problem: the rapid and accurate detection of fruit diseases. The goal was to build a solution that is not only technologically advanced but also **accessible, scalable, and extremely cost-effective**.

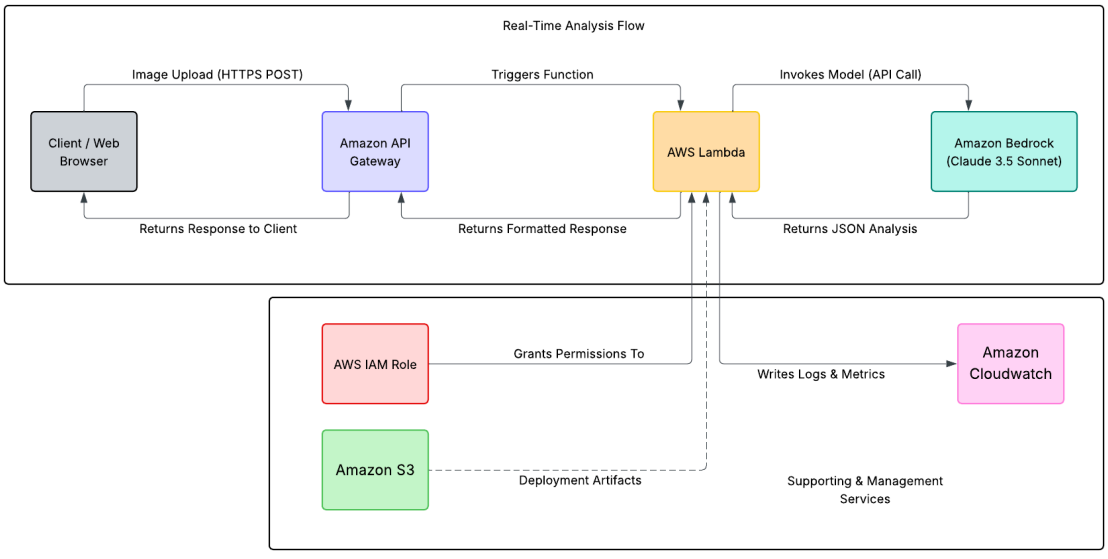
The core of this project is a serverless API on AWS. A user can send an image of a piece of fruit, and in seconds, receive an AI-generated analysis that identifies potential diseases and suggests treatments. This demonstrates a real-world application of cloud and AI technology to the agricultural sector.

2. The Architecture: A Serverless-First Approach

To achieve our goals, I chose a **serverless, event-driven architecture**. This was a deliberate decision to avoid the cost and management overhead of traditional servers. The entire system operates on a pay-per-use basis, meaning if no one is using the API, the cost is effectively zero.

3. Architectural Diagram

This diagram illustrates the flow of data from the user to the AI and back. It is a simple, linear process designed for low latency and high reliability.



AWS Real-Time Inference Architecture Diagram

3. Service Breakdown & Design Justification

Here's a breakdown of each service and why I chose it for this specific project.

Amazon API Gateway:

Function: The secure "front door" for our application.
Justification: I chose API Gateway because it's a fully managed service that lets me create a secure, public API without worrying about servers, scaling, or security patches. Its native integration with Lambda is the key to our event-driven design.

AWS Lambda:

Function: The "brain" of the operation; it runs our Python code.
Justification: Lambda is the heart of a serverless architecture. It allowed me to focus solely on writing the application logic (preparing the prompt, calling Bedrock) and not on server administration. The pay-per-execution model was the most cost-effective way to implement our core logic.

Amazon Bedrock (Claude 3.5 Sonnet):

Function: Provides the AI for image analysis.
Justification: Building and hosting a state-of-the-art AI model is technically complex and extremely expensive. Bedrock gives us API access to powerful models like Claude 3.5 Sonnet on a simple, on-demand basis. This was the only feasible way to incorporate advanced AI into the project within our constraints.

AWS IAM (Identity and Access Management):

Function: Our application's security guard.
Justification: Security is non-negotiable. I used IAM to create a specific role for the Lambda function, locking it down so it could only access the exact resources it needed to do its job. This is a fundamental security best practice.

Amazon S3 & AWS SAM:

Function: S3 acts as the storage backend for our deployment process, managed by AWS SAM.
Justification: To achieve Operational Excellence, I used the AWS Serverless Application

Model (SAM) to define my entire application as code. When I deploy, SAM packages my code and uploads it to S3, which then serves as the source for CloudFormation to build the application. This automates the entire deployment pipeline.

Amazon CloudWatch:

Function: Our logging and monitoring service.
Justification: In a serverless application, you can't log into a server to see what went wrong. CloudWatch is essential because it automatically collects all logs from Lambda. It's the first place I'd look to debug any issues, making it critical for maintaining the system.

4. Example Output (AI Response)

Here is a sample JSON response returned by the API after analyzing an image of a strawberry with gray mold. The prompt given to the AI model specifically requested the output in this structured format.


```
{
  "fruit_type": "strawberry",
  "disease_detected": true,
  "disease_name": "gray mold",
  "confidence": 0.8,
  "severity": "moderate",
  "treatment_recommendations": [
    "Remove infected fruits and plant debris",
    "Improve air circulation around plants",
    "Apply fungicide specifically formulated for Botrytis"
  ]
}
```


5. Future Enhancements

This architecture provides a strong foundation for future growth. Potential next steps include:

- **Storing Analysis History:** Integrate a DynamoDB table to store a history of user requests and AI responses, allowing for data analysis and potential fine-tuning.

- **Asynchronous Processing:** For larger files like videos, the architecture could be extended with S3 event notifications and an SQS queue to handle processing asynchronously.