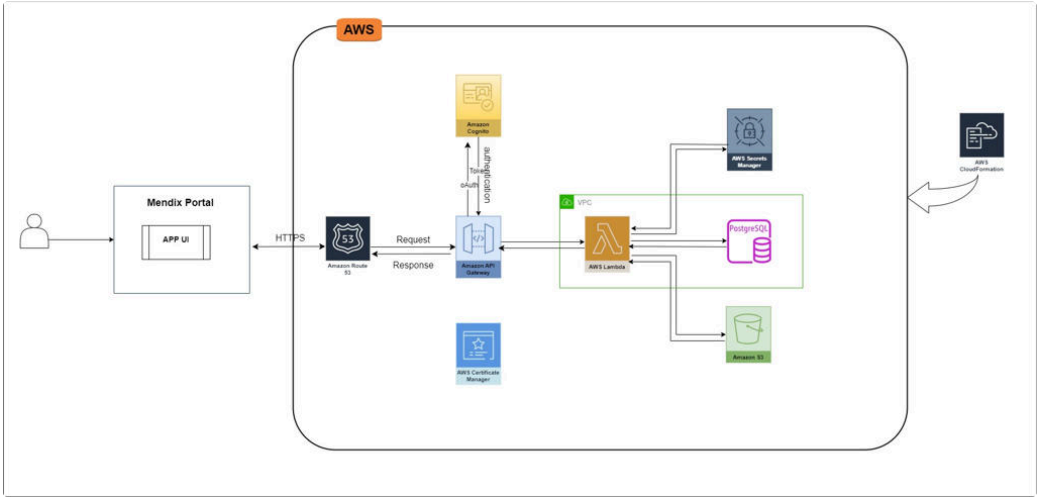
# MRT API: High-Level Architecture

### Overview

The **MRT API** leverages several AWS services to build a robust, serverless architecture for managing and accessing key application data. This architecture enables secure, scalable, and efficient API endpoints for retrieving bucket sizes, database table sizes, user data, and microflow constants. In this document, we will explain the services used in the API architecture, how they integrate with each other, and the role each service plays in the solution.

---




### Key AWS Services and Their Roles

#### 1. Amazon VPC (Virtual Private Cloud)

- **Purpose**: Provides a secure, isolated network for your AWS resources.
- **Role in the Stack**:
  - All Lambda functions are deployed within a **private subnet** of a VPC, ensuring they can securely access internal resources (like RDS and S3) without exposure to the public internet.

- Allows fine-grained control over network traffic between resources (e.g., between Lambda functions and databases).

#### 2. AWS Lambda

- **Purpose**: A serverless compute service that runs code in response to events, such as HTTP requests or data changes.
- **Role in the Stack**:
  - Multiple Lambda functions are created to handle different API requests, such as retrieving bucket size, table size, user data, and microflow constants.
  - Lambda functions interact with other services (e.g., **Amazon RDS**, **Amazon S3**) to fetch or modify data, and they return results back to the API Gateway.
  - Lambda functions are deployed within the **VPC**, ensuring secure communication with resources in private subnets.

#### 3. Amazon API Gateway

- **Purpose**: A fully managed service for creating and exposing APIs.
- **Role in the Stack**:
  - **API Gateway** serves as the entry point for external requests, exposing HTTP API endpoints that trigger Lambda functions.
  - Routes incoming HTTP requests (e.g., `GET /bucket_size`, `GET /table_size`) to the appropriate Lambda function for processing.
  - Supports **CORS** for cross-origin requests and handles authorization via **Cognito** (using a Cognito User Pool Authorizer).
  - The API is accessible through a **custom domain name** (e.g., `maas-api.<dns_domain>`), with secure communication enabled through SSL/TLS certificates managed by **AWS Certificate Manager** (ACM).

#### 4. Amazon Cognito

- **Purpose**: A service for managing user authentication and authorization.
- **Role in the Stack**:
  - **Cognito User Pools** are used to authenticate users, ensuring that only authorized users can access the API.
  - API requests are authorized through **OAuth** tokens issued by the Cognito User Pool, which the **API Gateway** uses to validate incoming requests.

- Custom **scopes** and **resource servers** are defined for fine-grained access control (e.g., read-only access for specific users).
- This integration ensures that only authenticated users can access specific resources exposed by the API.

#### 5. AWS IAM (Identity and Access Management)

- **Purpose**: Controls access to AWS services and resources.
- **Role in the Stack**:
  - **IAM Roles** are created for Lambda functions, granting them permissions to interact with other AWS services, such as **Amazon RDS**, **Amazon S3**, and **CloudWatch**.
  - IAM policies ensure that Lambda functions can only access the specific resources they need, following the **principle of least privilege**.
  - This ensures that your Lambda functions can securely fetch data from services like RDS (database queries) and S3 (bucket size calculations) without over-permissioning.

#### 6. AWS Certificate Manager (ACM)

- **Purpose**: Manages SSL/TLS certificates for securing communication over HTTPS.
- **Role in the Stack**:
  - An **SSL certificate** is provisioned through ACM for the **custom domain** used by the API Gateway (e.g., `maas-api.<dns_domain>`).
  - This ensures that all communication between clients and the API Gateway is encrypted using **HTTPS**, protecting sensitive data from eavesdropping and tampering.

#### 7. Amazon Route 53

- **Purpose**: A highly available and scalable DNS service.
- **Role in the Stack**:
  - **Route 53** manages DNS records for the custom API domain.
  - A **CNAME record** is created to map the custom domain (`maas-api.<dns_domain>`) to the API Gateway's regional domain name, enabling users to access the API via a friendly, branded URL.

---

### Service Interactions: How They Connect

The MRT Stack uses these services in a seamless, integrated manner to deliver a fully functional serverless solution. Here's a breakdown of how these services interact:

1. **User Authentication (Cognito)**:
   - When a user tries to access the API, they authenticate via **Amazon Cognito**. Cognito issues OAuth tokens for successful login attempts.
   - **API Gateway** validates these tokens using **Cognito Authorizer** to ensure the request is from an authenticated user.
2. **API Gateway and Lambda Functions**:
   - Once a request is authenticated, **API Gateway** forwards the request to the appropriate **Lambda function** based on the requested route (e.g., `/bucket_size`, `/table_size`, etc.).
   - Lambda functions execute the backend logic, such as retrieving data from **Amazon S3** (bucket sizes), **Amazon RDS** (table sizes), or performing other tasks.
   - Lambda functions are granted specific permissions through their **IAM roles** to interact with these services securely.
3. **Secure Communication (ACM & Route 53)**:
   - All communication between the client and **API Gateway** is secured using an SSL certificate provisioned by **AWS Certificate Manager**.
   - **Route 53** ensures that the custom domain (e.g., `maas-api.<dns_domain>`) maps to the correct API Gateway endpoint, ensuring that requests are routed to the right location.

---

### Key Use Cases

The MRT Stack is designed to handle a variety of data management tasks through simple, secure API calls. Below are the key use cases:

#### 1. Bucket Size Retrieval

- **API Endpoint**: `/bucket_size`
- **Description**: Fetches the size of a specific S3 bucket or other storage resources. Useful for monitoring and optimizing storage costs.

#### 2. Database Table Size Retrieval

- **API Endpoint**: `/table_size`
- **Description**: Retrieves the size of a database table (likely stored in **Amazon RDS**). This can be used for database optimization, monitoring, or scaling decisions.

#### 3. User Data Access

- **API Endpoint**: `/users`
- **Description**: Retrieves user-related data, such as metadata or user profiles, typically for reporting or analytics purposes.

#### 4. Microflow Constants Management

- **API Endpoints**:
  - `GET /micro_flow_constants`: Retrieves current constants used in the application's microflows.
- **Description**: Allows for the retrieval of constants used by the application for workflow management.

---

### Security and Permissions

The MRT Stack employs a **zero trust** security model to ensure that only authorized users and services can access sensitive data:

- **IAM Roles and Policies**: Lambda functions are assigned specific IAM roles with the minimum necessary permissions to interact with services like **Amazon RDS**, **Amazon S3**, and **CloudWatch**.
- **Cognito Authorization**: Only authenticated users, verified through **Cognito User Pools**, can access the API.
- **VPC Isolation**: Lambda functions operate within a **VPC**, ensuring they can securely communicate with internal resources without exposing them to the public internet.

---

### Conclusion

The MRT API is a comprehensive solution that leverages multiple AWS services to build a secure, scalable, and serverless application. By integrating **AWS Lambda**, **API Gateway**, **Cognito**, and other key services, the stack provides an efficient architecture for managing and accessing critical application data. The security-first approach ensures that only authorized users can access sensitive endpoints, while the serverless nature of the stack provides flexibility and cost efficiency.