Material Management System Project: AWS and DevOps Documentation

This document provides a detailed explanation of the AWS and DevOps setup implemented for the Material Management System (MMS) project. It leverages serverless microservices built with Node.js and utilizes various AWS services for deployment, management, and monitoring.

1. Development Environment

This project utilizes several AWS services to create a serverless architecture:

- **AWS Lambda:** Serverless compute service for running code without managing servers.
- **AWS API Gateway:** RESTful API for exposing Lambda functions to frontend applications.
- **AWS Step Functions:** Orchestrates complex workflows by chaining multiple Lambda functions.
- **AWS S3:** Scalable object storage for storing static content (e.g., frontend code).
- **AWS DynamoDB:** NoSQL database for storing and querying application data.
- **AWS CloudFormation:** Infrastructure as Code (IaC) tool for managing and automating cloud resources.
- **CloudFront Distribution:** Content Delivery Network (CDN) for delivering frontend content with low latency.
- **AWS Cognito:** Manages user authentication and authorization for secure access.
- **Amazon Route 53:** Domain Name System (DNS) service for managing domain names and routing traffic.
- **Amazon CloudWatch:** Provides monitoring and logging capabilities for AWS resources.

2. Infrastructure Deployment

2.1 Serverless Microservices:

The MMS project is deployed as Node.js serverless microservices on AWS Lambda. Each microservice handles a specific functionality within the application. For example, a "create material" microservice might handle user requests to add new materials to the system.

2.2 Infrastructure as Code (IaC):

AWS CloudFormation is used to automate the deployment of infrastructure resources. A serverless.yml file defines the Lambda functions, API Gateway endpoints, DynamoDB tables, and other resources needed for the application. This file allows for repeatable and consistent deployment across different environments (e.g., development, staging, production).

2.3 Workflow Orchestration:

Complex workflows within the MMS system are orchestrated using AWS Step Functions. These workflows chain together multiple Lambda functions, ensuring they are executed in the correct order. This allows for building complex functionalities like order processing or user onboarding.

3. **Frontend and Backend Deployment:**

- **Frontend:** Static content for the frontend application is stored in an S3 bucket. A CloudFront distribution is configured to serve this content efficiently with low latency.
- **Backend:** The backend serverless microservices are deployed using the serverless.yml file. When changes are made to the backend code, developers trigger a deployment from GitHub Actions.

4. **Secure Access and Routing:**

- **Domain Names:** A custom domain (sparquer.com) is managed through Route 53. Subdomains are created for the backend API (api-dev.sparquer.com) and frontend application (mms-dev.sparquer.com).
- **Amazon Certificate Manager (ACM):** A single SSL certificate is issued from ACM and used for both subdomains, ensuring secure communication between users and the application.
- **API Gateway Configuration:** DNS records for the API Gateway are added to Route 53, enabling users to access the backend services using the api-dev.sparquer.com subdomain.
- **CloudFront Configuration:** The S3 bucket containing the frontend code is linked with a CloudFront distribution. DNS records for the CloudFront distribution are added to Route 53, allowing users to access the frontend application using the mms-dev.sparquer.com subdomain.

5. **Monitoring and Logging:**

- **CloudWatch Logs:** Each API Gateway endpoint is attached to CloudWatch. This allows developers to access logs from different microservices for debugging and troubleshooting purposes. CloudWatch provides insights into application behavior, helping identify and resolve issues.

6. **Continuous Integration and Continuous Delivery (CI/CD) Pipeline:**

- **GitHub Actions:** A CI/CD pipeline is configured using GitHub Actions. Upon code changes in the backend repository, the pipeline triggers a deployment process.
- **Serverless Framework:** The pipeline utilizes the Serverless Framework to automate the deployment of serverless resources to AWS.
- **Deployment Workflow:** The workflow runs on GitHub Actions' free virtual machines for cost-efficiency. It allows developers to trigger deployments from any branch. However, as per discussion, the recommended workflow is:
  - Develop features on a develop branch.
  - Deploy from develop to a pre-production environment (e.g., api-staging.sparquer.com, mms-staging.sparquer.com) for testing on the preproduction branch.
  - Once testing is complete, merge changes from develop to the main branch.
  - Deployments from main trigger deployment to the production environment (`mms.sparquer.com`)