**Project 1: Serverless Image Processing with AWS Lambda, Amazon S3, and DynamoDB**

**Objective:** Develop a serverless image processing application using AWS Lambda, Amazon S3, and DynamoDB.

**Steps:**

1. Create an S3 bucket to store original images.

2. Set up a Lambda function to automatically trigger on image uploads, apply a watermark, and save processed images back to S3.

3. Create a DynamoDB table to store metadata about processed images.

4. Expose REST APIs using API Gateway to retrieve processed image metadata and URLs.

5. Implement Java Lambda functions and jUnit tests for image processing and DynamoDB interactions.

6. Store Lambda function secrets (like watermark details) in AWS Secrets Manager.

7. Implement automated builds using AWS CodeBuild.

8. Integrate SonarQube to ensure code quality during the build process.

9. Develop a React web application that fetches and displays processed images' information.

10. Deploy the React application to an S3 bucket and secure it with Amazon Cognito for user authentication.

**Project 2: Real-time Chat Application with AWS Lambda, Amazon DynamoDB, and WebSocket API**

**Objective:** Build a real-time chat application using AWS Lambda, Amazon DynamoDB, and WebSocket API.

**Steps:**

1. Set up an Amazon DynamoDB table to store chat messages and user information.

2. Develop Lambda functions to handle user authentication, message sending, and retrieval.

3. Create a WebSocket API using API Gateway to enable real-time communication.

4. Implement Java Lambda functions and jUnit tests for user authentication and message handling.

5. Store sensitive Lambda function data, like authentication secrets, in AWS Secrets Manager.

6. Configure automated builds and deployments using AWS CodeBuild.

7. Integrate SonarQube to maintain code quality during the build process.

8. Develop a React-based chat interface that interacts with the WebSocket API.

9. Deploy the React chat interface to an S3 bucket and secure it with Amazon Cognito for user authentication.

**Project 3: E-commerce Recommendation System with AWS Lambda, Amazon Personalize, and Amazon RDS**

**Objective:** Create an e-commerce recommendation system using AWS Lambda, Amazon Personalize, and Amazon RDS.

**Steps:**

1. Set up an Amazon RDS database to store product and user information.

2. Import product and user data into the RDS database.

3. Integrate Amazon Personalize to build recommendation models based on user behavior.

4. Develop Lambda functions to fetch and display personalized recommendations.

5. Implement Java Lambda functions and jUnit tests for recommendation retrieval and display.

6. Store RDS database credentials securely in AWS Secrets Manager.

7. Establish automated builds and deployments using AWS CodeBuild.

8. Integrate SonarQube for maintaining code quality during the build process.

9. Create a React-based web application to showcase personalized product recommendations.

10. Deploy the React application to an S3 bucket and secure it with Amazon Cognito for user authentication.

------------------------------------------------------------------------------------------------------------------------

**Project 4: Document Management System with AWS Lambda, Amazon S3, Amazon DynamoDB, and AWS Secrets Manager**

**Objective:** Develop a document management system using AWS Lambda, Amazon S3, Amazon DynamoDB, and AWS Secrets Manager.

**Steps:**

1.Create an S3 bucket to store documents uploaded by users.

2.Set up Lambda functions triggered by S3 events to process uploaded documents, such as extracting metadata or performing text analysis.

3.Utilize DynamoDB to store metadata associated with each document, including user information and processing status.

4.Store sensitive information like API keys or encryption keys securely in AWS Secrets Manager.

5.Expose REST APIs using API Gateway to enable CRUD operations on documents and their metadata.

6.Implement Java Lambda functions and unit tests for document processing and database interactions.

7.Establish automated builds and deployments using AWS CodeBuild.

8.Integrate SonarQube for code quality analysis as part of the build process.

9.Develop a web-based interface using React for users to upload, view, and manage their documents securely.

10.Deploy the React application to an S3 bucket and authenticate users with Amazon Cognito.

------------------------------------------------------------------------------------------------------------------------

**Project 5: Inventory Management System with AWS Lambda, Amazon DynamoDB, Amazon S3, and AWS CloudFormation**

**Objective:** Build an inventory management system using AWS Lambda, Amazon DynamoDB, Amazon S3, and AWS CloudFormation.

**Steps:**

1. Create a DynamoDB table to store inventory data, including product information, quantities, and locations.

2. Implement Lambda functions to handle CRUD operations on inventory items, triggered by API Gateway endpoints.

3. Utilize Amazon S3 to store product images associated with inventory items.

4. Define infrastructure as code using AWS CloudFormation to provision resources consistently.

5. Secure sensitive data such as database credentials using AWS Systems Manager Parameter Store (SSM Parameter Store).

6. Implement automated builds and deployments using AWS CodeCommit and AWS CodePipeline.

7. Integrate SonarQube for code quality checks in the CI/CD pipeline.

8. Develop a web-based interface using React to interact with the inventory system, including adding, updating, and removing items.

9. Deploy the React application to an S3 bucket and secure it with Amazon Cognito for user authentication.

10. Utilize AWS CloudWatch for monitoring and logging of system activities.

------------------------------------------------------------------------------------------------------------------------

**Project 6:** Task Management Application with AWS Lambda, Amazon ECS, Amazon RDS, and AWS CodeCommit.

**Objective:** Create a task management application using AWS Lambda, Amazon ECS, Amazon RDS, and AWS CodeCommit.

**Steps:**

1.Set up an Amazon RDS database to store task data, including descriptions, deadlines, and assignees.

2.Develop microservices using AWS Lambda and Amazon ECS to handle task management functionalities, such as creating, updating, and assigning tasks.

3.Utilize AWS Secrets Manager to securely store database credentials and other sensitive information.

4.Implement continuous integration and continuous deployment (CI/CD) using AWS CodeCommit for version control and AWS CodePipeline for automated builds and deployments.

5.Integrate SonarQube for code quality analysis in the CI/CD pipeline.

6.Expose RESTful APIs through API Gateway to enable interaction with the task management system.

7.Develop a web-based frontend using React to allow users to view, create, update, and delete tasks.

8.Deploy the React application to an S3 bucket and authenticate users with Amazon Cognito.

9.Utilize AWS CloudWatch for monitoring application performance and tracking user activities.

10.Implement scalability and high availability using AWS ECS to manage containerized microservices.