Sarapis

This notebook will have Sarapis related resources

- Sarapis Architecture Design
- Managed Pipeline
- 10/9 Meeting Minutes
- Sarapis Development Resources

Sarapis Architecture Design

Overview

Sarapis has an open-source backend data administrator interface tailored for managing Human Services Data Standard (HSDS) datasets, which provide standardized information about health, human, and social services. The project's primary objective will be to create a modular, service-based backend that simplifies the management, validation, and interaction of HSDS datasets.

Project Goals

- **Interoperability and Standardization**: Enhance data interoperability by providing a platform that validates HSDS formats and ensures interface consistency in managing health, human, and social services information across various organizations.
- **Scalability and Modularity:** The service must provide modularity and scalability to ensure seamless integration with other HSDS-supporting applications.

Key Features

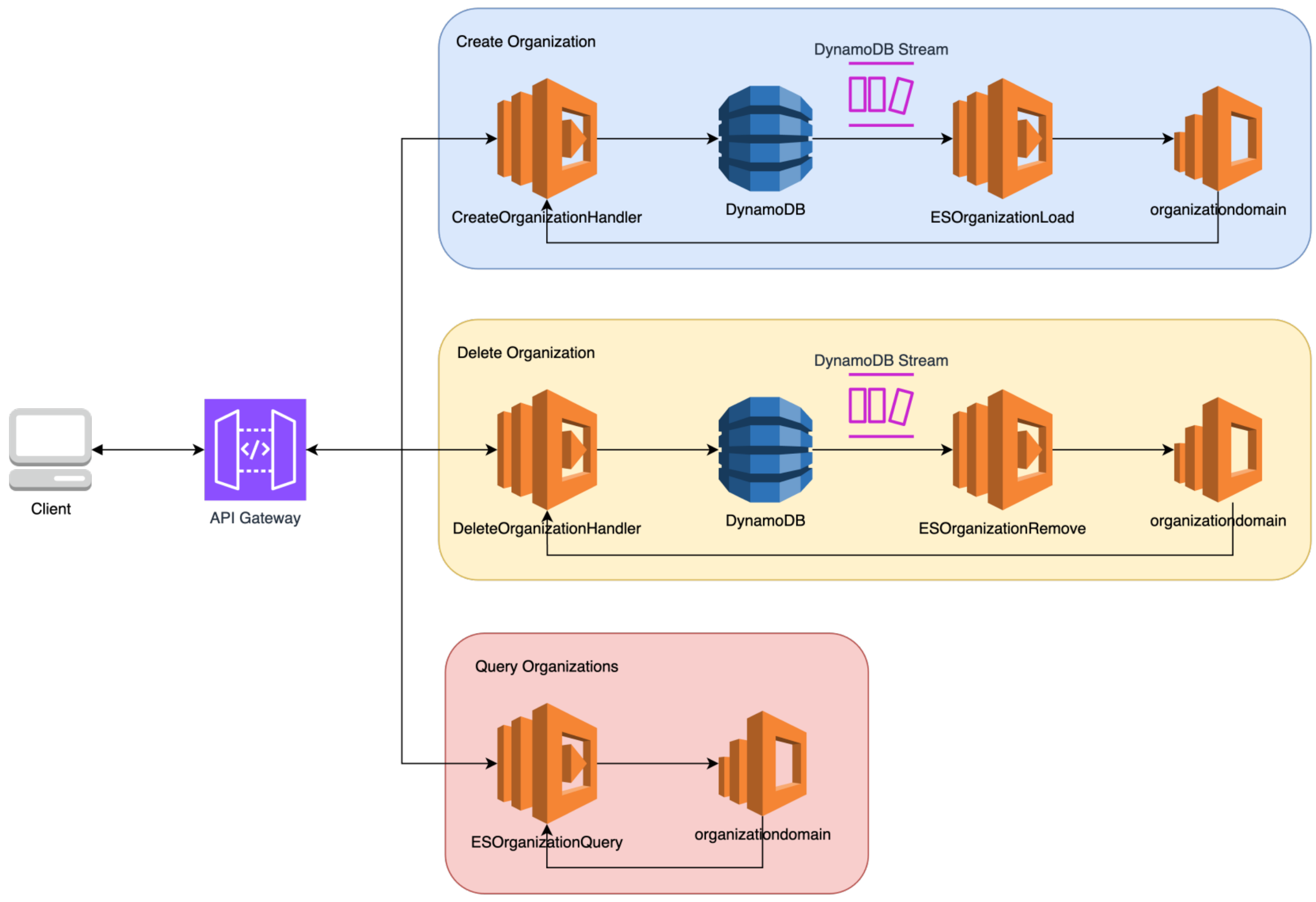
- **Database Management:** CRUD (Create, Read, Update, Delete) operations for managing various HSDS entities
- **Taxonomy Management**: Tools for managing and customizing taxonomies to classify resources and services.
- **Data Validation**: Integration with OpenReferralsUK's schema validator to ensure dataset compliance with HSDS specifications
- **User and Role Management**: Comprehensive user account, permissions, and authentication features to control access and data integrity.
- **Import/Export functionality**: Support for data import and export in multiple formats to facilitate data exchange and integration

High-Level Architecture Overview

The backend service will be hosted and deployed to AWS. Therefore, we can leverage existing AWS services to achieve some of the key features mentioned in the project specification. The service does not require scalable services for each feature, so the Micro-service architecture is discarded. Instead, the backend service will follow a monolithic architecture where all the endpoints will be published in a single service.

AWS Lambda

AWS is a serverless computing service that lets you run code without provisioning or managing servers. Creating an HTTP server using Lambda involves integrating Lambda with Amazon API Gateway, a managed service that allows you to create, publish, maintain, and secure RESTful APIs. This setup enables Lambda to handle incoming HTTP requests, effectively simulating an HTTP server while leveraging the benefits of a serverless architecture.



Core components

- **API Gateway as the Request Router**: API Gateway serves as the frontend layer, processing incoming HTTP requests and routing them to Lambda functions. It manages the HTTP/HTTPS endpoints, handling HTTP method routing (GET, POST, PUT, DELETE), request authentication, and throttling. API Gateway can also transform incoming requests and outgoing responses to match specific requirements.
- **Lambda Function Execution**: When an HTTP request hits the API Gateway endpoint, API Gateway triggers a Lambda function according to the preconfigured route. Lambda then executes the defined code based on the request's contents. The function’s logic can range from basic CRUD operations to more complex workflows, such as data processing, integration with other AWS services, or calling external APIs.
- **Request-Response Workflow**:
  - **Request Mapping**: API Gateway maps HTTP requests into event objects that the Lambda function can interpret. This mapping typically includes path parameters, query parameters, headers, and body content in JSON format.

    - **Response Mapping**: After executing the function, Lambda returns a response in a specific format that API Gateway can parse and convert to an HTTP response, including headers, status codes, and body content. API Gateway handles any necessary transformation before sending the response to the client.
- **Error Handling and Retries**: Lambda functions can be configured to handle different errors, allowing for customized responses based on HTTP status codes. API Gateway can handle retries for idempotent operations, such as GET requests, ensuring reliability without impacting the underlying function’s state.

Lambdas can also be integrated with AWS Cognito for token-based authentication, or you can also establish custom Lambda authorizers for more specific access control logic.

Since Lambda is inherently stateless:

- **External Databases**: You’ll connect Lambda to external databases (DynamoDB) to store and retrieve data.
- **API Gateway Caching**: You can enable caching in API Gateway for specific endpoints to reduce the frequency of database calls for repeated requests, improving performance and reducing costs.

Limitations and Challenges

1. **Cold Starts**: If not used recently (i.e., a "cold start"), lambda functions may incur latency on the first invocation, impacting applications with strict performance requirements.

Elasticsearch

Elasticsearch serves as a search engine. It is integrated primarily for its ability to index and query, large volumes of data quickly. In this architecture, Elasticsearch complements DynamoDB by providing efficient querying capabilities, particularly for complex searches that are not natively supported in DynamoDB.

Data Ingestion and Synchronization:

- **Data from DynamoDB Streams:** When organizations are created, updated, or deleted in DynamoDB, a DynamoDB Stream captures these changes. Lambda functions, such as `ESOrganizationLoad` and `ESOrganizationRemove`, listen to these streams.
- **Indexing in Elasticsearch:** Upon detecting relevant changes (like organization creation or deletion), these Lambda functions update the corresponding indexes in Elasticsearch. For instance, `ESOrganizationLoad` would add or update organization data in Elasticsearch, while `ESOrganizationRemove` would delete entries.
- **Consistency:** This setup ensures that the data in Elasticsearch mirrors the current state of organizations in DynamoDB, maintaining consistency across both data stores.

Query Operations:

- **Enhanced Search Capabilities:** When users query organizations (handled by the `ESOrganizationQuery` Lambda), Elasticsearch enables more complex and efficient search functionalities. Elasticsearch's indexing and full-text search capabilities allow for faster and more flexible queries, like searching by name, location, or organization type.
- **Direct Queries for High Performance:** By querying directly in Elasticsearch rather than DynamoDB, response times improve, especially for more intricate or resource-intensive search patterns.

Challenges with Elasticsearch Integration

- **Indexing Delays:** Although the Lambda functions attempt to maintain real-time synchronization between DynamoDB and Elasticsearch, some minor delays may occur in propagating changes, depending on the load and configuration.
- **Data Consistency Management:** Ensuring consistency between DynamoDB and Elasticsearch is crucial. In case of errors or failures in Lambda functions, there could be discrepancies, which need monitoring and handling to prevent data integrity issues.
- **Additional Cost and Complexity:** Elasticsearch incurs additional costs and adds complexity, especially for small applications where DynamoDB alone might suffice. Careful management of Elasticsearch indices and resources is necessary to optimize cost.

Appendix

What is a taxonomy?

You can think of a taxonomy as a structured system used to classify and organize information into categories, making it easier to understand, manage, and retrieve. It involves some sort of hierarchical arrangement, where items are grouped into broader categories and further divided into subcategories.

```
Health Services
- Medical Care
- - Primary Care
- - Specialty Care
- Mental Health
Housing Services
- Emergency Shelters
- Affordable Housing
- Housing Assistance Programs
```

Spring Boot on EC2

Spring Boot with EC2 is a traditional setup for deploying a self-contained backend application on virtual machines. The backend application can be packaged as a standalone executable using a

Java build tool such as Maven or Gradle. Spring Boot offers dependency management, pre-configured settings, and integrated database support, making it ideal for creating microservices and RESTful APIs.

The advantage of choosing Sprint Boot is its more flexible runtime. There are no cold starts. The application can be stateful by attaching persistent storage to an EC2 instance.

Limitations and Challenges

- **Server Management**: EC2 instances require management for updates, scaling, patching, and security, which can increase operational overhead.
- **Scaling Complexity**: Scaling on EC2 requires manual intervention or automation setup (e.g., auto-scaling groups), and it may not scale as seamlessly as Lambda.
- **Potential Idle Costs**: With EC2, you pay for uptime regardless of whether the server is fully utilized, which may make it less cost-effective for low-traffic applications.
- **Slower Deployment**: Changes in your application may require redeployment of the EC2 instances, which can be slower than updating a Lambda function.
- **Single Point of Failure**: If not designed with redundancy (e.g., using load balancers and multiple instances), a single EC2 instance can become a single point of failure.

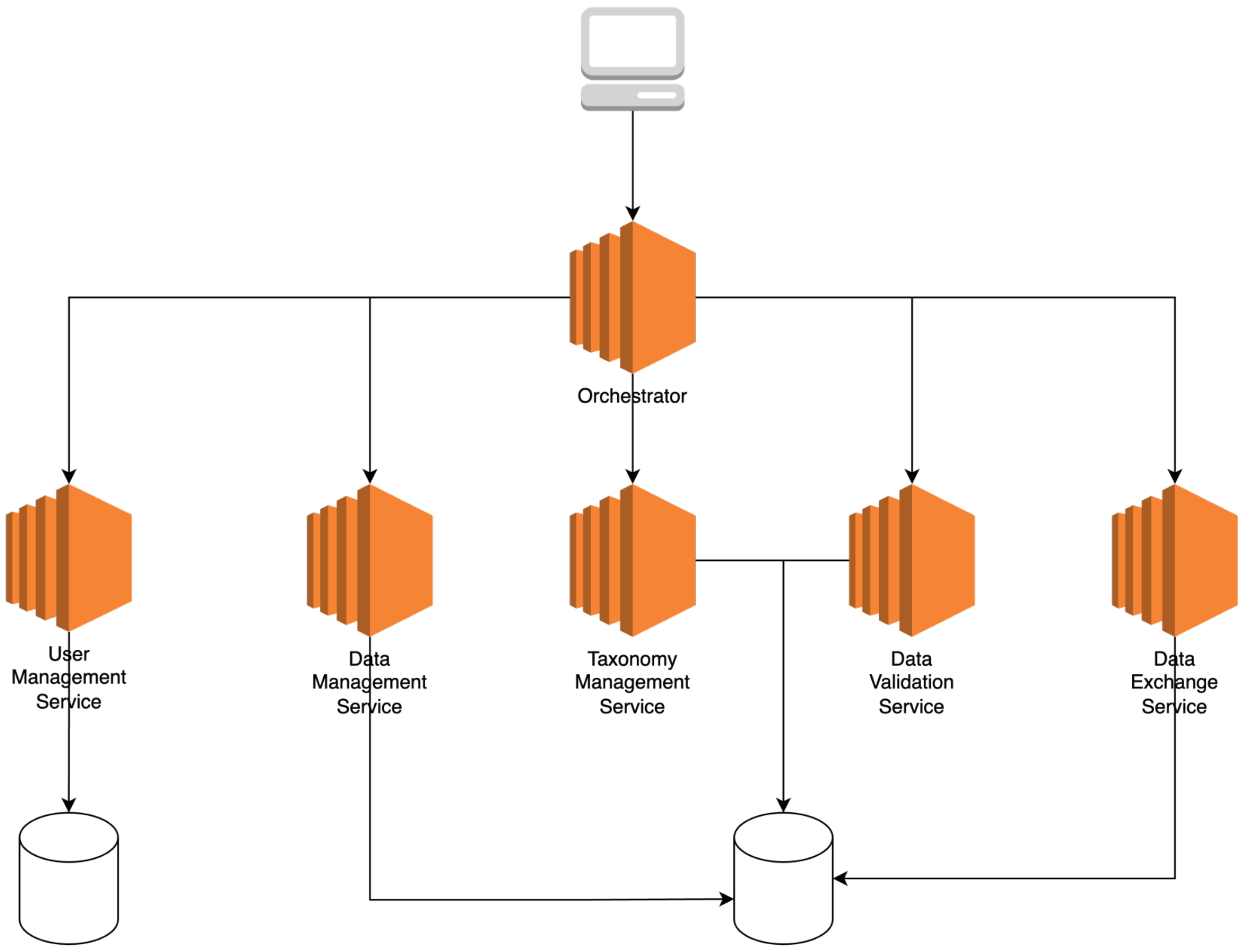
Microservice Architecture

Due to the scalability and flexibility requirements of the project following a micro-services architecture, it will align well with the goals of modularity, scalability, and integration within the broader open-source ecosystem. The microservice architecture for the proposed HSDS backend administration interface can be structured as follows:

- User Management Service
    - **Functionality:** Manages user's accounts, roles, permissions, and authentication
    - **APIs:**
        - `/register`for user registration
        - `/login` for user login and token issuance
        - `/roles` for managing roles and permissions
- Data Management Service
    - **Functionality:** Handles CRUD operations for HSDS data, including organizations, locations, services, and other related entities.
    - **APIs:**
        - `/create` add a new record
        - `/read` to retrieve records
        - `/update` to modify records
        - `/delete` to remove records
- Taxonomy Management Service
    - **Functionality:** Manages taxonomies and classifications used within the HSDS datasets
    - **APIs:**

        - `/create-taxonomy` to add new taxonomies
        - `/update-taxonomy` to edit taxonomies
        - `/list-taxonomies` to view available taxonomies
- Data Validation Service
    - **Functionality:** Validates the schema and compliance of the dataset using OpenReferralsUKs validator
    - **APIs:**
        - `/validate` to validate entire datasets or specific records
- Data Exchange Service
    - **Functionality:** Manages import/export operations for datasets in various formats
    - **APIs:**
        - `/import` to upload and import datasets
        - `/export` to download a dataset given the correct permissions

You can view the HSDS schema here.



Managed Pipeline

With large projects such as Sarapis with multiple components, it is important to define a process for testing and releasing new versions. As such, we are proposing a managed pipeline that automates testing, health checks, and deployments to different evironments. These environments would include:

- Development (Dev)
- Quality Assurance (QA)
- Production (Prod)

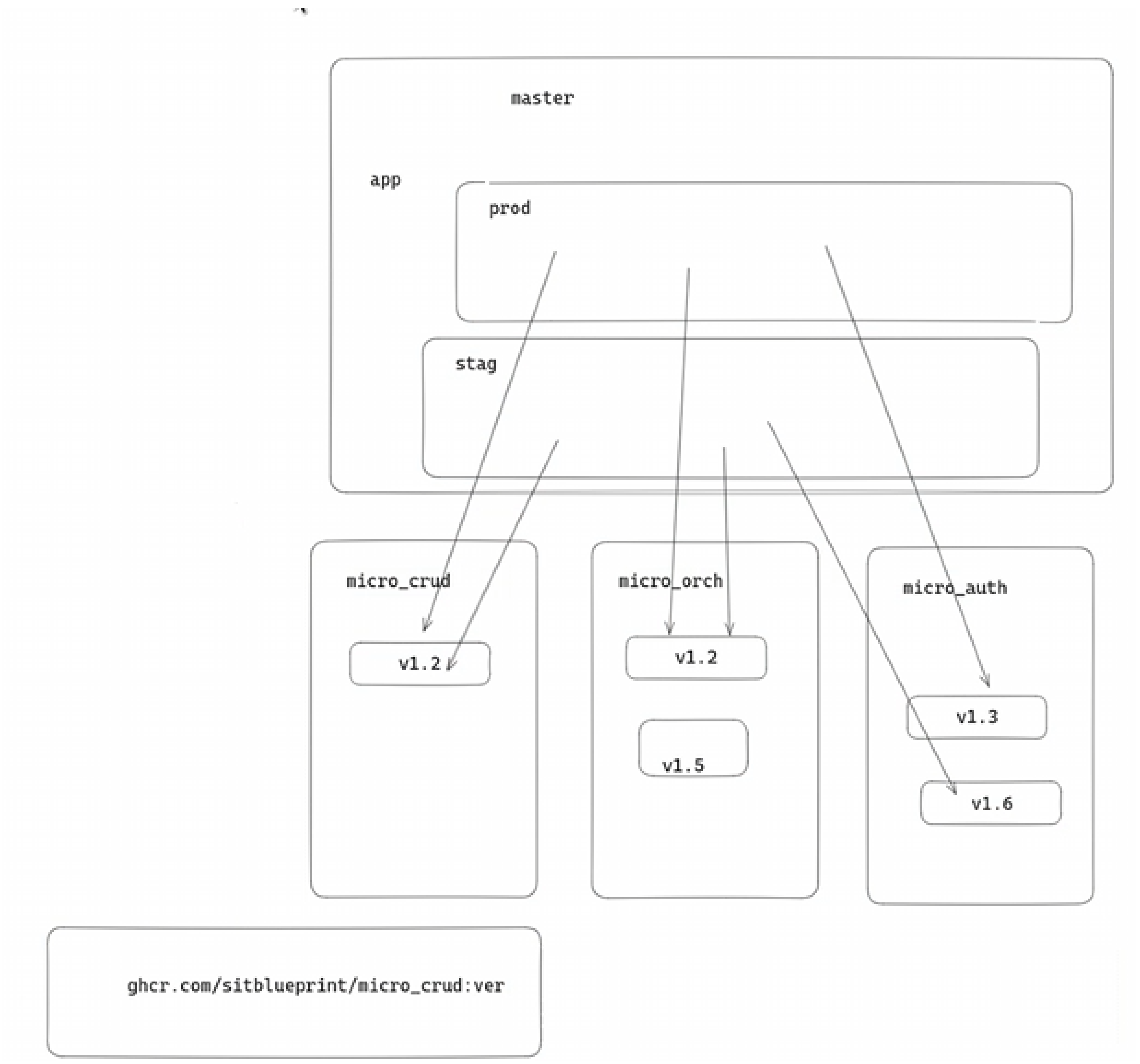
The pipeline should also allow Blueprint to easily rollback versions on the Software. Due to limited resources, we are looking to build this rapidly with minimum Infrastructure cost. A viable tool that we can use to fit this objective is GitHub actions, that way are infrastructure remains close to the code. We may have to use additional tools like Jenkins to handle builds, but that is what this document will define/explore.

A Minimum Viable Product Might Include:

- Hosting for Dev, QA, and Prod environments
- Authentication/Authorization for interacting with Dev, QA, and Prod environments
- Automated test suites that can be configured different projects
- Automated build processes that can be cofigured for different projects
- Ability to rollback production deployments to previous versions
- Notification system for any failures in the environments

10/9 Meeting Minutes

Multiple Repos:



Manage integrations through:

- GitHub Action
- GHCR

Non-Profit Meeting

- Is there better validtor services for HSDS data than UKValidator?
- Present architecture
- Proposing a shared database, might run into concurrency issues

Ownership:

Jonathan: User Management Service

Terrence: Data Management Service

Devin Meeting:

- Sarapis Team Members In-person: Next Wednesday 3:15-3:45 P.M
- Thursday/Friday Next Week Miguel/Ezri/Jonathan/Terrence
- Miguel will create repositories and project boards

Sarapis Development Resources

HSDS Reference

https://docs.openreferral.org/en/v2.0.1/hsds/reference/

http://docs.openreferral.org/en/latest/hsds/schema_reference.html

Tutorial: Create a CRUD HTTP API with Lambda and DynamoDB

https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html

Realize A Simple CRUD API on Amazon DynamoDB

https://medium.com/@guraycintir/realize-a-simple-crud-api-on-amazon-dynamodb-1457a5e124b7

API Gateway Lambda DynamoDB CRUD Flow Repo

https://github.com/gcintir/api-gateway-lambda-dynamodb-crud-flow/tree/main

AWS CDK

https://docs.aws.amazon.com/cdk/v2/guide/hello_world.html

Testing with SAM

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-cdk-testing.html

Locally invoke Lambda functions with AWS SAM

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-using-invoke.html

SAM invoke documentation

https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/invoke.html

Key condition expressions for the Query operation in DynamoDB

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.KeyConditionExpressions.html

Create an Amazon Cognito user pool for a REST API

https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-create-cognito-user-pool.html

Add more features and security options to your user pool

https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-next-steps.html

AWS Cognito with ReactJS for authentication

https://medium.com/@adi2308/aws-cognito-with-reactjs-for-authentication-c8916b873ccb

Use API Gateway Lambda authorizers

https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html

DynamoDB Local Sample Java Project

https://github.com/awslabs/amazon-dynamodb-local-samples

Configure the URLConnection-based HTTP client

https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/http-configuration-url.html

DTOs

https://medium.com/linkit-intecs/getting-started-with-crud-operations-in-spring-boot-and-dynamodb-a-beginners-guide-75ecad3b0452

Amazon Cognito

Integrating Amazon Cognito for Authentication and Authorization in a Spring Boot Application

https://medium.com/@abhishekranjandev/integrating-amazon-cognito-for-authentication-and-authorization-in-a-spring-boot-application-fe5fe7d78db

The Easiest Way to Deploy Containers on AWS

https://www.fernandomc.com/posts/easiest-way-to-deploy-aws-containers/

Spring Boot Role-Based Authentication with AWS Cognito

https://howtodoinjava.com/spring-security/spring-boot-role-based-authentication-with-aws-cognito/