Sample Chat Application
Documentation

Backend: Serverless Microservices with AWS Lambda and Amazon API Gateway.
Client Side: Android Application
Amazon Web Services: AWS Lambda, API Gateway, DynamoDB, Cognito, SNS

Summary:

This document describes the use of the above mentioned services specific to Sample Chat Application.

AWS Lambda:

All the handlers (Lambda Java functions) which hold the logic for each backend operation, as a single and independent function, are listed below. These functions are exposed via a REST API maintained in Amazon API Gateway. Proxy integration has been used between API and lambda functions i.e the request is passed on to the function as it is, containing queryStringParameters, pathParameters, headers, body etc.

The commands for creating the following functions with AWS CLI are also listed below.

1. getUserThreads:
   operation: Fetch all conversations for the user along with the list of users in each conversation (Two users = 1 to 1 chat, more than two = group chat)
   input: String username
   output: Map<String, List<String>> where map contains threadId and list of all its users

2. getThreadMessages:
   operation: Fetch the list of messages in a conversation
   input: String threadId
   output: List<Map<String,String> where each Map contains attributes and values for a message item.

3. newThreadMessage:
   operation: Insert a new thread item in DynamoDB Threads Table with auto-generated id. Then insert a new message item with auto-generated id for the new thread in Messages Table.
   Input: String username, String toUser, String message
   Output: void
   TODO: Integerate with SNS

4. newMessage:
   Operation: Insert a new message item in Messages Table with auto-generated id, given threadId and fromUser(username)
   Input: String username, String threadId, String message
   Output: void
   TODO: Integerate with SNS

5. postSignUpConfirmation:
   Operation: insert a new User in DynamoDB Users table. Function is triggered directly by Cognito as it is registered for the user pool's post confirmation event (new user signup complete)
   Input: String username (included in the request when invoked from cognito)
   Output: void

6. createTables:
   Operation: create all the required tables (Users, Threads, Messages) in DynamoDB. Schema is defined in Java Classes that represent a corresponding table.
   Input: none
   Output: none

7. describeTables:
   Operation: describe the schema of all the given Table.
   Input: String tableName
   Output: String tableDescription

8. deleteTables:
   Operation: Delete all the required tables (Users, Threads, Messages) in DynamoDB.
   Input: none
   Output: none

Instructions for defining Lambda Functions:

1. Create the Jar file using maven
2. For ease, define a user environment variable named as ROLE_ARN with the arn of an IAM role as value. E.g arn:aws:iam::875324011155:role/lambda_basic_execution
3. Execute the following commands using AWS command line interface. Note: A credentials file containing access key must be configured for the CLI.

aws lambda create-function --function-name getUserThreads --runtime java8 --memory-size 512 --timeout 10 --role %ROLE_ARN% --handler com.xoho.lambda.MessagingHandlers::getUserThreads --zip-file fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar

```
aws lambda create-function --function-name getThreadMessages --runtime java8 --memory-size
512 --timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::getThreadMessages --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name newThreadMessage --runtime java8 --memory-size
512 --timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::newThreadMessage --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name newMessage --runtime java8 --memory-size 512
--timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::newMessage --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name postSignUpConfirmation --runtime java8
--memory-size 512 --timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.CognitoHandlers::postConfirmationHandler --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name createTables --runtime java8 --memory-size 512
--timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::createTables --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name describeTable --runtime java8 --memory-size 512
--timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::describeTable --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

```
aws lambda create-function --function-name deleteTables --runtime java8 --memory-size 512
--timeout 10 --role %ROLE_ARN% --handler
com.xoho.lambda.MessagingHandlers::deleteTables --zip-file
fileb://target/MessagingHandlers-1.0-SNAPSHOT.jar
```

API Gateway:

Create a REST API with resources and HTTP methods that map to the lambda functions. Use lambda proxy integration when configuring the Integration Request for HTTP methods and then just choose the name of the function to complete the integration. Define query string parameters or path parameters in method request with the same name as the defined in input for each function. Create models for method response and body input for POST methods so that the SDK for the API can be generated and used easily at client side.

DynamoDB:

The following three tables are required and are defined as follows.

1. Users
    - username (String)
    - threads (List)

2. Threads
    - threadId (String auto-generated)
    - users (List)
    - messages (list)

3. Messages
    - messageId (String auto-generated)
    - threadId (String)
    - fromUser (String)
    - message (String)
    - timestamp (String)

Functional Flow:

App gets the list of the threads (conversations) from the Users table for the user that signed In. To show a conversation, App gets the list of users for the thread (2 users = 1 to 1 chat , >2 users = group chat) and gets the list of messages in that thread from the Threads Table. App then displays the messages by getting message attributes(message,time, fromUser) using the list from threads table and identifies which message to show as sent message by looking at fromUser attribute, the rest of the users in that thread are receivers for that message.

Note:

A key attribute can only be defined as auto-generated when the table is generated using the DynamoDB SDK and e.g in java @DynamoDBAutoGeneratedKey annotation is used. Invoke the createTables lambda function to create these tables as defined above.

AWS SNS

Mobile Push Notifications can be used via SNS (Simple Notification Service) and supported messaging platforms. In case of android, the supported platform is Firebase Cloud Messaging

(Google Cloud Messaging is deprecated). SNS uses the API Key provided by Google to create a platform application. Then it uses a token generated by FCM for the app installed on a device to create an end-point which can be used to subscribe to a topic or direct notifications. Creating platform application, end-point, creating/subscribing/publishing to topics etc. all can be done via both the web console and the SDK.

Note: Functionality uptil getting the generated token has been implemented in Android.

Cognito:

Amazon Cognito enables maintaining users for the Application. A userpool can be created and configured easily using the web console and by following the Guide. A userpool can be associated with the App through the Mobile Hub web console.

The Android application uses the SDK provided by amazon for sign up/ sign in/ sign out functions. It also uses a built-in user interface provided by the SDK for the above use cases.

Android:

User Interface for a chat screen (1 to 1) has been implemented. Sent messages and dummy replies can be seen in the activity.

Pending: Using the Android SDK generated by Amazon for the REST API to call backend logic for sending and fetching messages.