LOMBA KOMPETENSI SISWA (LKS) SEKOLAH MENENGAH KEJURUAN TINGKAT KOTA BEKASI - 2025

NASKAH SOAL

Bidang Lomba

Cloud Computing

Modul 1 - Deploying Simple Chat Application using Serverless Technologies

January 23, 2025

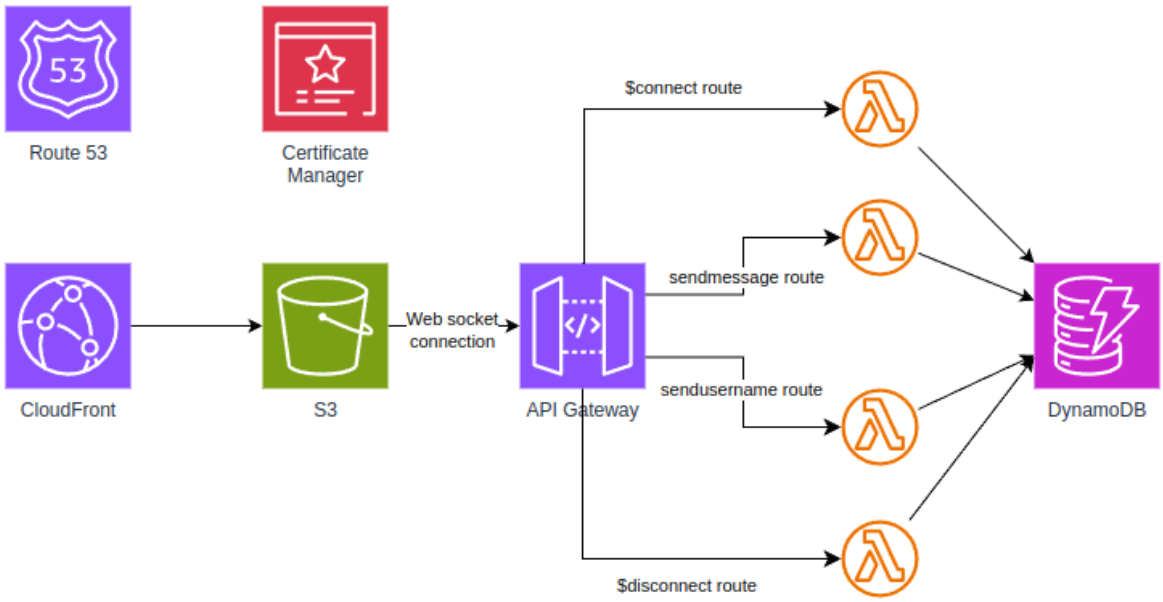

Figure 1: Architecture Diagram

1 Overview

Implementing a serverless architecture on Amazon Web Services (AWS) can provide numerous benefits, including cost optimization, scalability, and reduced administrative burden. Your task is to deploy a simple chat application using serverless architecture.

2 General Rules

1. Failure to comply with the rules will result in immediate disqualification.
2. You have 3 hours to finish the tasks.
3. You may not open any website unless otherwise specified in section 7 and you may open the control panel of your domain provider to update the nameserver to Route 53.
4. Using the search feature from the AWS documentation website is allowed. However opening other AWS website such as AWS Solutions Library is NOT permitted. If you are unsure about the eligibility of the site you want to open, please ask.
5. You may use AWS Console, CLI, SAM, CloudFormation, or CDK to deploy the application.
6. During the event, multiple login is not permitted.
7. If you have any question, do not hesitate to ask.

3 Architecture

Refer to Figure 1.

1. Lambda to handle chat application server's logic.
2. API Gateway (Websocket APIs) to publish the Lambda via websocket protocol.

3. DynamoDB to store user information and chat logs.
4. S3 to host the front-end application.

4 Information

1. The repository URL for the required source code to deploy this solution is: https://github.com/kensasongko/lksbk2025
2. This solution must be deployed in **us-east-1 (North Virginia)** region. Deploying in another region will result in a major point reduction.
3. An automated scoring system will check your AWS account to calculate the score of your task.
4. In order check your progress, the scoring system needs an AWS credential with administrator privilege. You will be asked to provide the AWS credential.
5. AWS tag is used by the scoring system to find your solution. Failing to add the required tag or adding the same tag value to multiple instance may result in an unexpected behavior and thus lower your score.
6. The system will calculate 90% of the final score. The remaining 10% will be evaluated by checking the assignment manually.
7. If multiple participants completed the assignment, their completion times will factor into the manual scoring process. For instance, the first participant to finish will receive a full score, while subsequent finishers will incur minor penalties.

5 Task

Your goal is to deploy the solution from the section 3. Follow the following instructions to deploy the solution.

1. Create a hosted zone for your domain in AWS Route 53 and point the nameservers accordingly.
2. Create a certificate for *.[YOU_DOMAIN] in ACM for CloudFront and API Gateway
3. Create 2 Dynamo DB tables with the following configurations:
   (a) User table
   - Table name: chat-user-table
   - Partition key: connectionId (String)
   - Read/Write capacity settings: On-Demand
   - Table class: DynamoDB Standard
   - Tag: Key=LKS-CC-BEKASI-2025, Value=user-table
   - Add global secondary index (GSI):
     - Index name: byIsActive
     - Partition key: isActive (Number)
     - Sort key: createdAtConnectionId (String)

   (b) Message table
   - Table name: chat-message-table
   - Partition key: messageId (String)
   - Read/Write capacity settings: On-Demand
   - Table class: DynamoDB Standard

- Tag: Key=LKS-CC-BEKASI-2025, Value=message-table
- Add global secondary index (GSI):
  - Index name: byIsDeleted
  - Partition key: isDeleted (Number)
  - Sort key: createdAtConnectionId (String)

4. Create IAM Role and Policy

   (a) Create an IAM Policy with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/chat-*-table",
                "arn:aws:dynamodb:*:*:table/chat-*-table/index/*"
            ]
        }
    ]
}
```

   (b) Create an IAM Role for Lambda service, attach the policy you have previously created and a AWS-managed policy named "AWSLambdaBasicExecutionRole".

5. Create and deploy Lambda functions for the chat application server's logic.

   - The source code of the lambda function can be found in section 4.
   - Every function requires the following configurations:
     - Runtime: Node.js 22.x
     - Architecture: arm64
     - Execution role: IAM Role you have created in task number 4.

   (a) Connect route handler function
   - Source directory: sources/functions/connect/
   - Environment Variables:
     - userTable: chat-user-table
   - Tag: Key=LKS-CC-BEKASI-2025, Value=connect-handler

   (b) Disconnect route handler function
   - Source directory: sources/functions/disconnect/
   - Environment Variables:
     - messageTable: chat-message-table
     - userTable: chat-user-table
     - isActiveUserIndex: byIsActive
   - Tag: Key=LKS-CC-BEKASI-2025, Value=disconnect-handler

(c) Sendusername route handler function

- Source directory: sources/functions/sendusername/
- Environment Variables:
  - messageTable: chat-message-table
  - userTable: chat-user-table
  - isActiveUserIndex: byIsActive
- Tag: Key=LKS-CC-BEKASI-2025, Value=sendusername-handler

(d) Sendmessage route handler function

- Source directory: sources/functions/sendmessage/
- Environment Variables:
  - messageTable: chat-message-table
  - userTable: chat-user-table
  - isActiveUserIndex: byIsActive
- Tag: Key=LKS-CC-BEKASI-2025, Value=sendmessage-handler

6. Create WebSocket API with a custom domain

(a) Create WebSocket API with the following configurations:

- Route selection expression: request.body.action
- Predefined routes: $connect, $disconnect.
- Custom Routes: sendmessage, sendusername.
- Stage: production
- Attach lambda functions you have previously created in task number 5 to the routes.
- Tag: Key=LKS-CC-BEKASI-2025, Value=chat-api

(b) **Test:** At this point you should be able to connect to the websocket API, for example using wscat:

```
wscat -c wss://xxxxx.execute-api.us-east-1.amazonaws.com/production/
```

Replace the websocket URL with your websocket URL, you can find it in the Stages page. And the expected output should be:

```
Connected (press CTRL+C to quit)
>
```

After you have successfully connected to the websocket, the user table on DynamoDB should have at least 1 new item. On the prompt, type the following json and press enter:

```
Connected (press CTRL+C to quit)
> {"action":"sendusername","username":"testuser1"}
>
```

If you have setup the sendusername handler correctly, you should see the username is now stored in user table and the message table should at least have 1 new item.

(c) Add custom domain to the API with the following configuration:

- Domain name: api.[YOUR_DOMAIN] (e.g., api.cloudkeong.com)
- Endpoint type: Regional
- Tag: Key=LKS-CC-BEKASI-2025, Value=api-domain

(d) Create API mappings for your custom domain with the following configurations:

- API: Choose you websocket API you have just created.
- Stage: production

(e) Create a CNAME record to the API Gateway domain name (e.g., d-xxxxxxxxxx.execute-api.us-east-1.amazonaws.com) in Route 53.

(f) **Test:** Now you should be able to connect to the websocket API using your custom domain, for example using wscat:

```
wscat -c wss://api.[YOUR\_DOMAIN]
```

Replace [YOUR_DOMAIN] with your domain. The expected output should be:

```
Connected (press CTRL+C to quit)
>
```

7. Update IAM Role to allow lambda functions to manage WebSocket API connection, including sending a message.

(a) Create a new IAM Policy with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement2",
            "Effect": "Allow",
            "Action": [
                "execute-api:ManageConnections"
            ],
            "Resource": [
                "arn:aws:execute-api:*:*:[YOUR_API_ID]/*"
            ]
        }
    ]
}
```

(b) Replace [YOUR_API_ID] with your WebSocket API ID.

(c) Attach the new IAM Policy to the IAM Role you have previously created in task number 4. There should be 3 policies attached to the IAM Role.

8. Create S3 bucket to host the chat website

(a) Create an S3 bucket with the following configuration:

- Enable ACLs
- Disable block all public access
- Enable static website hostings
- Static website hosting's index document: index.html
- Tag: Key=LKS-CC-BEKASI-2025, Value=web-bucket

(b) Upload the web from the directory sources/web/ to the bucket with public-read access enabled.

(c) **Test:** After uploading, you should be able to open the website by opening the bucket website endpoint in a browser.

9. Create CloudFront distribution

(a) Create a new distribution with the following configurations:

- Origin domain: Your S3 bucket
- Use website endpoint instead of bucket endpoint
- Viewer protocol policy: Redirect HTTP to HTTPS
- Allowed HTTP methods: GET, HEAD, OPTIONS, PUT, POST, PATH, DELETE
- Cache policy: Caching Optimized
- Origin request policy: CORS-S3Origin
- Response headers policy: CORS-With-Preflight
- Do not enable security protections
- Price class: Use all edge locations (best performance)
- Alternate domain name (CNAME): chat.[YOUR_DOMAIN] (e.g., chat.cloudkeong.com)
- Custom SSL certificate: Use the certificate you created in task number 2
- Tag: Key=LKS-CC-BEKASI-2025, Value=web-cdn

(b) In Route 53, create an alias record from chat.[YOUR_DOMAIN] to your CloudFront distribution.

6 How to test your deployment

1. Using a browser, open http://chat.[YOUR_DOMAIN]. The server should redirect you to https://chat.[YOUR_DOMAIN].
2. Input wss://api.[YOUR_DOMAIN] into the URL textbox and user1 into the Username textbox. Press the Join button.
3. Open another browser window or tab and repeat step 1 and 3 but now enter a different username (e.g., user2)
4. The other chat window should receive a message '[user] has joined the chat'.
5. Type hello and press the Send button. The other chat window should receive a message '[user]: Hello'.
6. Press Leave button and the other chat window should receive a messge '[user] has left the chat'.

7 References

- Amazon DynamoDB
- AWS IAM
- AWS Lambda
- Amazon API Gateway
- Amazon S3
- Amazon CloudFront
- Amazon Route 53
- npm
- wscat - npm
- Use wscat to connect to a WebSocket API and send messages to it

Good luck!