



LOMBA KOMPETENSI SISWA (LKS) SEKOLAH MENENGAH KEJURUAN TINGKAT PROVINSI JAWA BARAT TAHUN 2024

INFORMASI DAN KISI-KISI

Bidang Lomba **CLOUD COMPUTING**



PEMERINTAH DAERAH PROVINSI JAWA BARAT DINAS PENDIDIKAN

Jalan Dr. Radjiman No. 6 Telp. (022) 4264813 Fax. (022) 4264881
Website : diskdik.jabarprov.go.id
e-mail: diskdik@jabarprov.go.id/sekretariatdiskdikjabar@gmail.com
BANDUNG - 40171



LOMBA KOMPETENSI SISWA (LKS) SEKOLAH MENENGAH KEJURUAN TINGKAT PROVINSI JAWA BARAT TAHUN 2024

NASKAH SOAL
***(Terbuka / Tertutup)**

Bidang Lomba
CLOUD COMPUTING



PEMERINTAH DAERAH PROVINSI JAWA BARAT DINAS PENDIDIKAN

Jalan Dr. Radjiman No. 6 Telp. (022) 4264813 Fax. (022) 4264881
Website : diskdik.jabarprov.go.id
e-mail: diskdik@jabarprov.go.id/sekretariatdiskdikjabar@gmail.com
BANDUNG - 40171

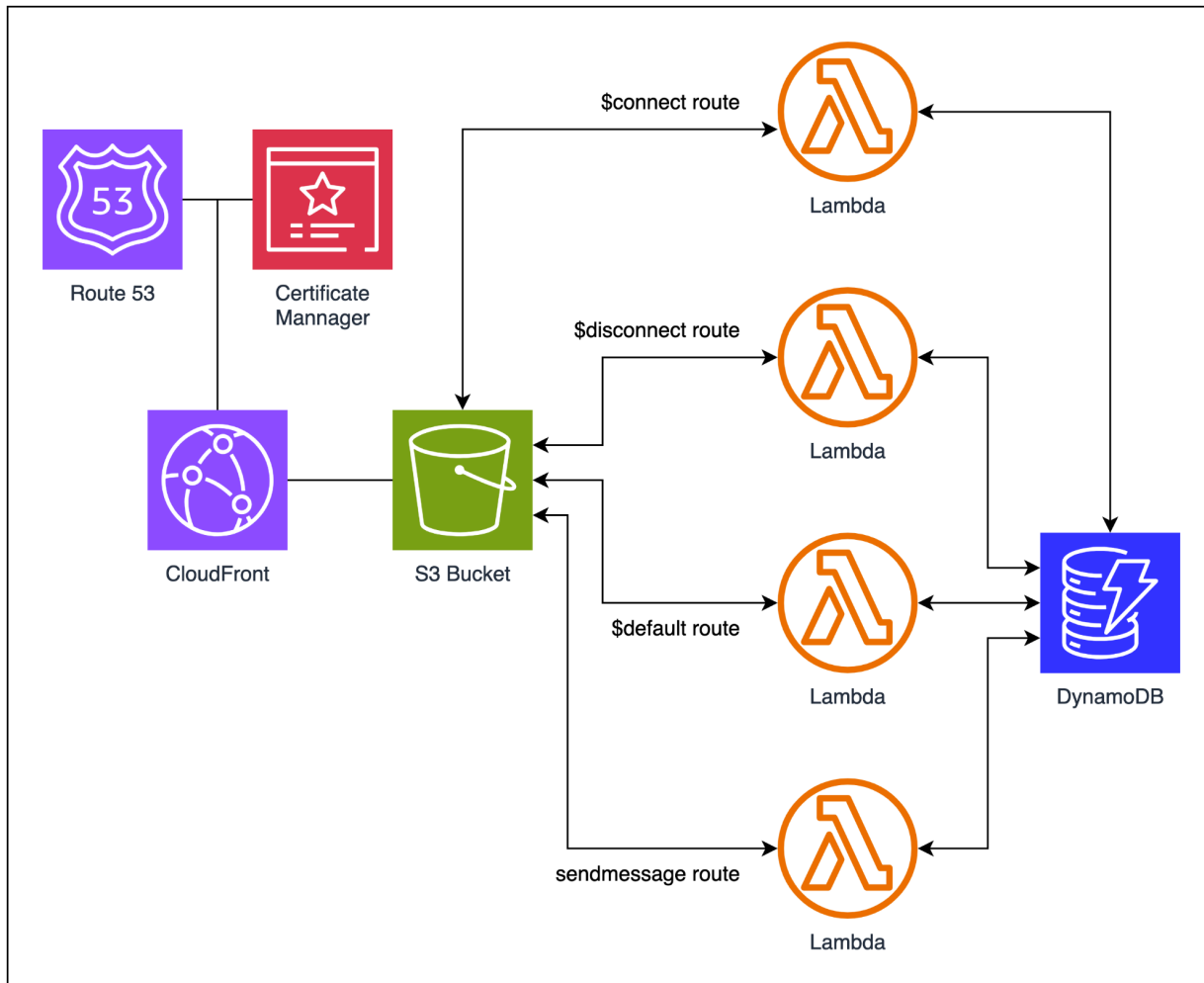
1. Overview

Building a chat application leveraging API Gateway WebSocket, DynamoDB, and AWS Lambda offers a robust and scalable solution. API Gateway WebSocket facilitates real-time bidirectional communication between clients and servers, while Lambda functions handle the application's logic, including message storage and retrieval in DynamoDB. This architecture ensures efficient handling of WebSocket events like connections, disconnections, and message exchanges. Leveraging Node.js enhances compatibility and stability, enabling seamless integration with AWS services. With this setup, developers can create a highly responsive and scalable chat platform, suitable for various real-time communication needs. This time we will create Lambda using Node.js 16 legacy code.

2. General Rules

1. Failure to comply with the rules will result in immediate disqualification.
2. You have 4 hours to finish the tasks.
3. You may use AWS Console and AWS CLI to deploy the solutions. You may not use SAM, CloudFormation or CDK.
4. Between and after the event, you may not access your account. Any activity on AWS during this period is not allowed.
5. During the event, multiple login is not permitted.
6. If you have any questions, do not hesitate to ask.

3. Architecture



1. A website should be hosted on S3 Website Hosting
2. Through provided website user can connect to chat server and chat with other users
3. When connecting, the server will store the connection id in DynamoDB table
4. When disconnecting, the server will remove the connection id from table
5. When user send messages the server will send the messages to other users based on their respective connection id

4. Information

1. This solution must be deployed in ap-southeast-1 region

2. Any deployment outside the designated region will result in major point reduction.
3. Make sure you register your [Certificate](#) in the us-east-1 region.

5. Task

5.1 Create DynamoDB Table

Create DynamoDB Table with following configuration

- Name : `lks-jabar-2024-connection`
- Partition key :
 - Name : `connectionId`
 - Type : `String`
- Sort Key : leave empty
- Table settings : choose `Customize Settings`
- Table class : `DynamoDB Standard`
- Read/write capacity settings : `On-demand`
- Tags :
 - Key : `LKS-CC-2024`
 - Value : `lks-chat-table`

5.2 Create Lambda Functions

5.2.1 Connect Handler

This lambda will be used to connect to chat server

1. Create Lambda Function with following configuration
 - Option : `Author from scratch`
 - Function name : `api-ws-connect-handler`
 - Runtime : `Node.js 16.x`
 - Architecture : `x86_64`
2. Open created Lambda Function
3. Upload `connect-handler.zip` as code for created Lambda Function
4. Go to **Configuration** tab and open **Tags** panel, add tag :
 - Key : `LKS-CC-2024`
 - Value : `lks-chat-connect-handler`

5. Go to **Configuration** tab and open **Environment variables** panel, add environment variable :
 - Key : table
 - Value : lks-jabar-2024-connection
6. Go to **Configuration** tab and open **Permission** panel
7. Open execution role and do the following :
 - Add new policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Action": [
        "dynamodb:BatchWriteItem",
        "dynamodb:PutItem",
        "dynamodb:DescribeTable",
        "dynamodb>DeleteItem",
        "dynamodb:UpdateItem"
      ],
      "Resource": "[arn_of_lks-jabar-2024-connection_table]"
    }
  ]
}
```

- Save policy with name api-ws-connect-handler-db-policy

5.2.2 Disconnect Handler

This lambda will be used to disconnect from chat server

1. Create Lambda Function with following configuration
 - Option : Author from scratch
 - Function name : api-ws-disconnect-handler
 - Runtime : Node.js 16.x
 - Architecture : x86_64
2. Open created Lambda Function
3. Upload disconnect-handler.zip as code for created Lambda Function
4. Go to **Configuration** tab and open **Tags** panel, add tag :
 - Key : LKS-CC-2024
 - Value : lks-chat-disconnect-handler
5. Go to **Configuration** tab and open **Environment variables** panel, add environment variable :

- Key : table
 - Value : lks-jabar-2024-connection
6. Go to **Configuration** tab and open **Permission** panel
 7. Open execution role and do the following :
 - Add new policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Action": [
        "dynamodb:BatchWriteItem",
        "dynamodb:PutItem",
        "dynamodb:DescribeTable",
        "dynamodb>DeleteItem",
        "dynamodb:UpdateItem"
      ],
      "Resource": "[arn_of_lks-jabar-2024-connection_table]"
    }
  ]
}
```

- Save policy with name api-ws-disconnect-handler-db-policy

5.2.3 Sendmessage Handler

This lambda will be used to send message to chat server

1. Create Lambda Function with following configuration
 - Option : Author from scratch
 - Function name : api-ws-sendmessage-handler
 - Runtime : Node.js 16.x
 - Architecture : x86_64
2. Open created Lambda Function
3. Upload sendmessage-handler.zip as code for created Lambda Function
4. Go to **Configuration** tab and open **Tags** panel, add tag :
 - Key : LKS-CC-2024
 - Value : lks-chat-sendmessage-handler
5. Go to **Configuration** tab and open **Environment variables** panel, add environment variable :
 - Key : table
 - Value : lks-jabar-2024-connection
6. Go to **Configuration** tab and open **Permission** panel

7. Open execution role and do the following

- Add new policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:Query",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:ConditionCheckItem",
        "dynamodb:DescribeTable"
      ],
      "Resource": "[arn_of_lks-jabar-2024-connection_table]"
    },
    {
      "Sid": "Statement2",
      "Effect": "Allow",
      "Action": "execute-api:ManageConnections",
      "Resource":
        "arn:aws:execute-api:[region]:[account-id]:**/POST/@connections/**"
    }
  ]
}
```

- Save policy with name api-ws-sendmessage-handler-db-policy

5.2.4 Default Handler

1. Create Lambda Function with following configuration
 - Option : Author from scratch
 - Function name : api-ws-default-handler
 - Runtime : Node.js 16.x
 - Architecture : x86_64
2. Open created Lambda Function
3. Upload default-handler.zip as code for created Lambda Function.
4. Go to **Configuration** tab and open **Tags** panel, add tag :
 - Key : LKS-CC-2024
 - Value : lks-chat-default-handler
5. Go to **Configuration** tab and open **Permission** panel
6. Open execution role and do the following :

- Add new policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Action": "execute-api:ManageConnections",
      "Resource":
"arn:aws:execute-api:[region]:[account-id]:*/*/POST/@connections/*"
    },
    {
      "Sid": "Statement2",
      "Effect": "Allow",
      "Action": "execute-api:ManageConnections",
      "Resource":
"arn:aws:execute-api:[region]:[account-id]:*/*/GET/@connections/*"
    }
  ]
}
```

- Save policy with name `api-ws-default-handler-db-policy`

5.3 Create WebSocket API

1. Open [API Gateway Console](#).
2. Choose **Create API**. Then for **WebSocket API**, choose **Build**.
3. For **API name**, enter `ws-chat-app`.
4. For **Route selection expression**, enter `request.body.action`.
5. Choose **Next**.
6. For **Predefined routes**, choose **Add \$connect**, **Add \$disconnect**, and **Add \$default**.
7. For **Custom routes**, choose **Add custom route**. For **Route key**, enter `sendMessage`
8. Choose **Next**.
9. Under **Attach integrations**
 - Integration for **\$connect**
 - Integration type : `Lambda`
 - AWS Region : your region
 - Lambda function : **Connect Handler** lambda you created previously
 - Integration for **\$disconnect**
 - Integration type : `Lambda`
 - AWS Region : your region
 - Lambda function : **Disconnect Handler** lambda you created previously
 - Integration for **\$default**

- Integration type : Lambda
 - AWS Region : your region
 - Lambda function : **Default Handler** lambda you created previously
 - Integration for **sendmessage**
 - Integration type : Lambda
 - AWS Region : your region
 - Lambda function : **Sendmessage Handler** lambda you created previously
10. Choose **Next**.
11. Under **Stages**
- Stage name : `production`
12. Choose **Next**.
13. Review your new WebSocket API.
14. Choose **Create and deploy**.
15. Open WebSocket API you have created on previous steps.
16. Select **API Settings** menu on left menu
17. Under **Tags**, add tag :
- Key : LKS-CC-2024
 - Value : lks-chat-websocket-api
18. Select **Stages** menu on left menu.
19. On **Stages** select **production**.
20. Note down **WebSocket URL** value.

5.4 Create S3 Website Hosting

5.4.1 Create S3 Bucket

1. Open [S3 Console](#)
2. Choose **Create Bucket**.
3. For **Bucket name**, enter `ws-chat-web-[account_id]`
4. Under **Object Ownership** :
 - Choose ACLs enabled.
 - Under **Object ownership**, choose **Bucket owner preferred**.
5. Under **Block Public Access settings for this bucket** :
 - Uncheck **Block all public access**.
 - Check ***I acknowledge that the current settings might result in this bucket and the objects within becoming public.***
6. Under **Tags**, add tag :
 - Key : LKS-CC-2024

- Value : lks-chat-ws-chat-bucket
7. Choose **Create Bucket**.

5.4.2 Enabling website hosting

1. Open S3 bucket you have created previously.
2. Choose **Properties** tab.
3. Under **Static website hosting**, choose **Edit**.
4. Under **Static website hosting**, choose **Enable**.
5. Under **Hosting type**, choose **Host a static website**.
6. In **Index document**, enter `index.html`.
7. Choose **Save changes**.

5.4.3 Uploading website to S3 bucket

1. Download `web.zip` and extract it on your computer.
2. Open S3 bucket you have created previously.
3. Choose **Upload**.
4. Under **Upload** :
 - Add extracted files into files list
 - Under **Permission / Access control list (ACL)**, choose **Grant public-read access**
 - Check ***I understand the risk of granting public-read access to the specified objects.***
5. Choose **Upload**.

5.4.4 Testing your website

1. Open S3 bucket you have created previously.
2. Choose **Properties** tab.
3. Under **Static website hosting** copy your website URL.
4. Open URL in browser.
5. You should see something like th

Connect Disconnect

5.5 Create distribution

5.5.1 Create Cloudfront distribution

1. Open [Cloudfront Console](#).
2. Choose **Create Distribution**.
3. Under **Origin**
 - For **Origin domain**, enter domain part of your S3 Website URL.
 - For **HTTP Port**, enter 80
 - For **Name**, enter `ws-chat-distribution`
4. Under **Default cache behavior**
 - For Viewer protocol policy, choose **Redirect HTTP to HTTPS**
5. For **Web Application Firewall**, choose **Do not enable protections**.
6. Under **Settings**
 - For **Alternate domain name**, choose **Add item** and enter a custom domain name that matches with your registered domain name. If you registered the domain name `example.com`, enter `chat.example.com` as a custom domain name.
 - For **Custom SSL Certificate**, choose a certificate from AWS Certificate Manager created for your registered domain.
7. Choose **Create distribution**
8. Open you created distribution and go to **Tags** tab
9. Add tag :
 - Key : LKS-CC-2024
 - Value : lks-chat-ws-chat-distribution

5.5.2 Create DNS Record

1. Open [Route 53 Console](#).
2. Open hosted zone for your registered domain.
3. Choose **Create record**, add new record with following configuration :
 - Record name : `chat`.
 - Record type : `CNAME`.
 - Value : domain name of your **Cloudfront distribution**
 - TTL : 300

5.6 Chat using your hosted website

1. Copy the alternate domain of your Cloudfront distribution and use it as your website URL.
2. Open your website URL in two or more browser tabs/windows.

3. Try to communicate between those tabs/windows :

- For **URL**, enter your **WebSocket URL**.
- Click **Connect**.
- Enter your message and click Send

The screenshot shows a web browser window with the address bar displaying `https://chat.example.com`. Below the address bar, there is a form with a label "URL" and a text input field containing `wss://ghyqfip0sk.execute-api.ap-southeast-1.amazon`. To the right of the input field are two buttons: "Connect" (grey) and "Disconnect" (red). Below this, there is a "Message*" label and a text input field containing the word "there". To the right of this input field is a "Send" button (purple). Below the "Send" button, there are three message bubbles: a light blue bubble containing "here", a light green bubble containing "there", and a light blue bubble containing "and everywhere".

6. Reference

- [DynamoDB](#)
- [Lambda](#)
- [API Gateway](#)
- [Simple Storage Service \(S3\)](#)
- [CloudFront](#)
- [Route 53](#)