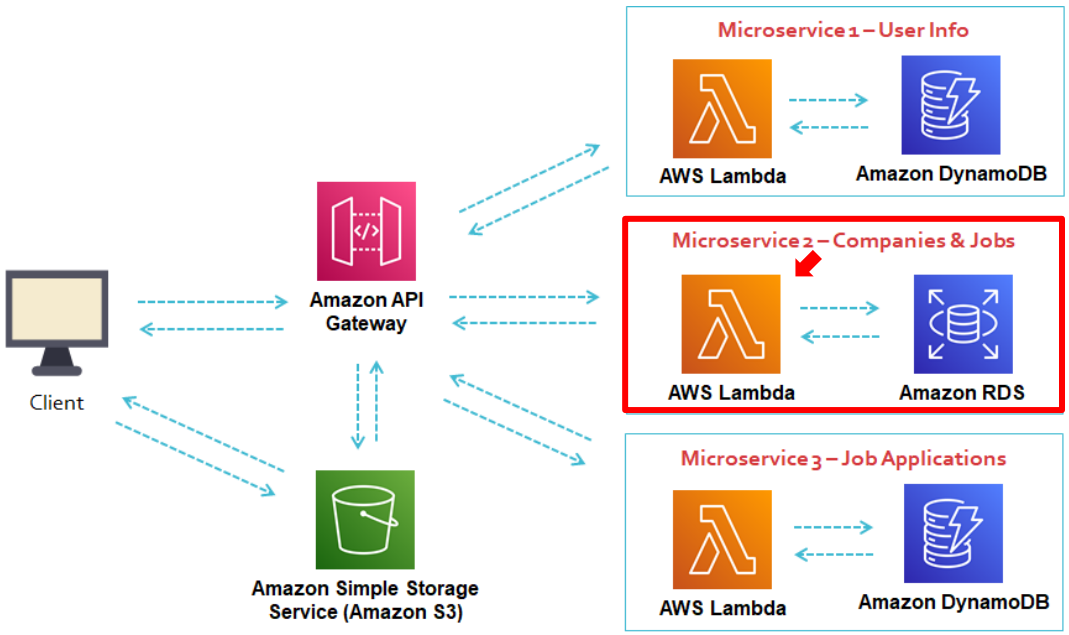
# Lab #06: Integrating Amazon RDS with AWS Lambda

Upon completion of this lab exercise, you would be able to:

- Set up Lambda Function with custom libraries
- Implement Create Retrieve, Update and Delete functions in Lambda that integrates with Amazon RDS

## Introduction

In this lab, you will learn to create AWS Lambda functions that communicates with Amazon RDS to perform CREATE, RETRIEVE, UPDATE and DELETE operations for the tables in Microservice 2.




Companies will be able to login and add in job opportunities. They are also able to modify and delete a job posting. The supporting functions and SQL commands are summarised as follows:

| Functionality | SQL Command |
|---|---|
| View all jobs | `SELECT * from `companies_jobs`.`jobs`;` |
| View jobs by companies | `SELECT * FROM `companies_jobs`.jobs WHERE company_id = ?;` |
| View selected job | `SELECT * FROM companies_jobs.companies, companies_jobs.jobs WHERE companies.id = company_id AND jobs.id = ?;` |
| Add job | `INSERT INTO `companies_jobs`.`jobs` (`title`, `type`, `sector`, `salary`, `role_description`, `qualification`, `experience`, `date_posted`, `company_id`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);` |
| Edit selected job | `UPDATE `companies_jobs`.`jobs` SET `title` = ?, `type` = ?, `sector` = ?, `salary` = ?, `role_description` = ?, `qualification` = ?, `experience` = ? WHERE id = ?;` |
| Delete selected job | `DELETE FROM `companies_jobs`.`jobs` WHERE id = ?` |
| Login | `SELECT * FROM `companies_jobs`.`companies` WHERE `email` = ? AND `password` = ?;` |

## A. Setting up the Environment for our Lambda Function

We will use **Node.js** as the programming language for our Lambda function. As we are using MySQL as our RDS Database, we will need to install **mysql** library. In this section, we will create a folder with the necessary libraries and upload that as our **base code** for the Lambda function.

1. Click on the **Windows** logo and search for **Command Prompt**.

2. Enter the following command to create a new directory, **lambda_rds**:

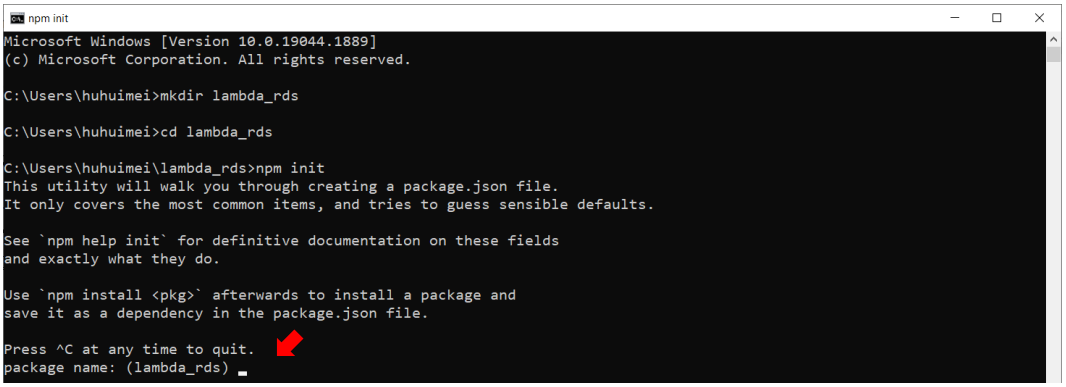
```
mkdir lambda_rds
```

3. Enter the following command to **navigate to the lambda_rds** folder:

```
cd lambda_rds
```

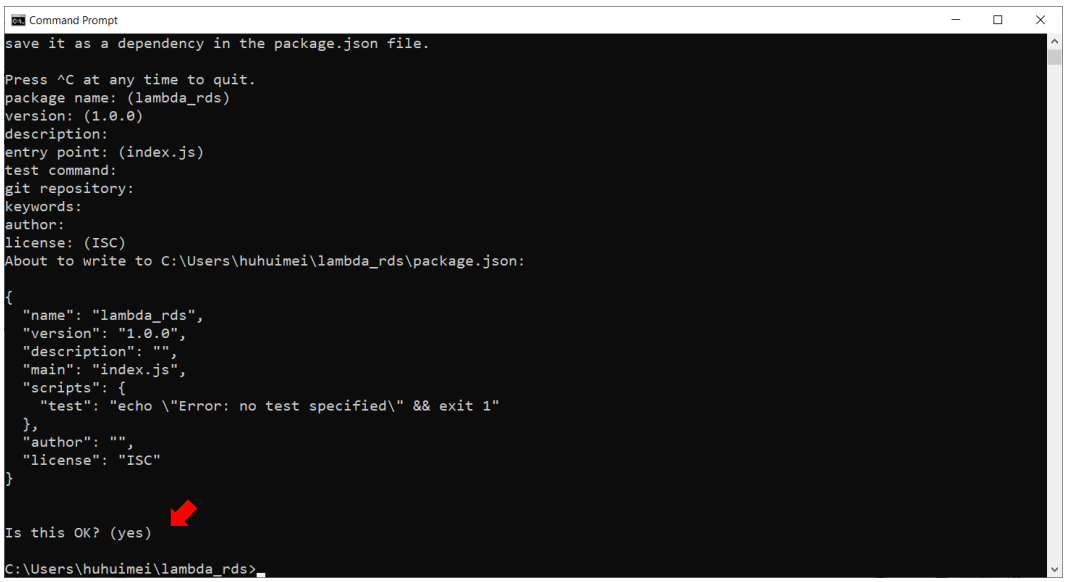
4. Initialise a new Node.js project using the following command:

```
npm init
```




*Accept the default package name by clicking on the **Enter** key when prompted.*

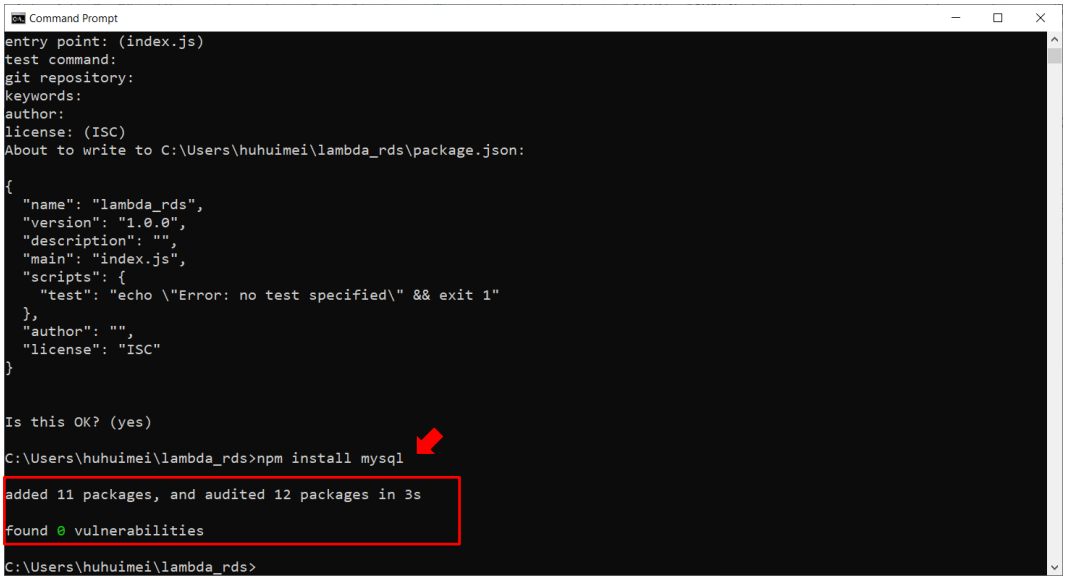
We will accept the **default value** for the other settings as well by clicking on the **Enter** key.



5. Enter the following command to **install mysql** library:

```
npm install mysql
```

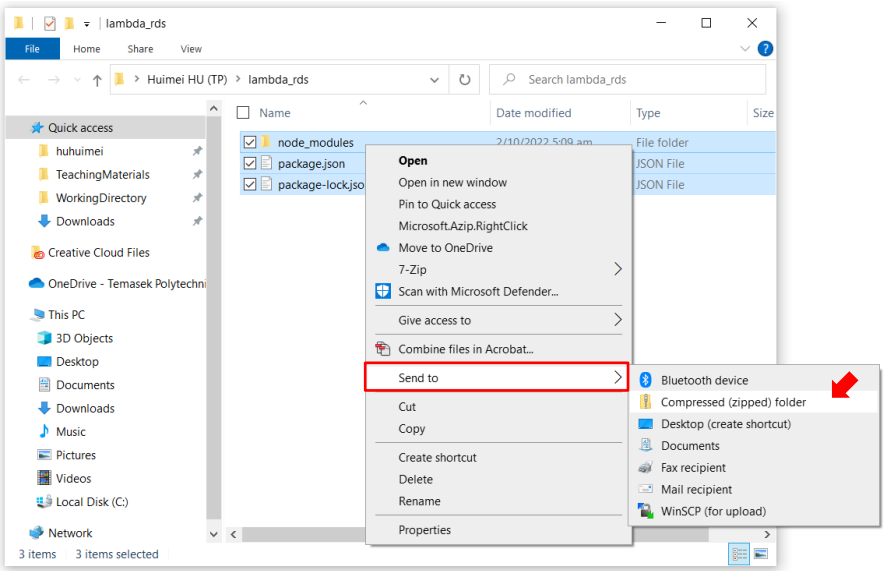
6. Wait for the installation to complete.




7. Enter the following command to open **lambda_rds** folder in **Windows Explorer**:

```
start .
```

8. In Windows Explorer, select all files → right-click on selection → Send to → Compressed (zipped) folder. Name the file as **lambda_rds.zip**.

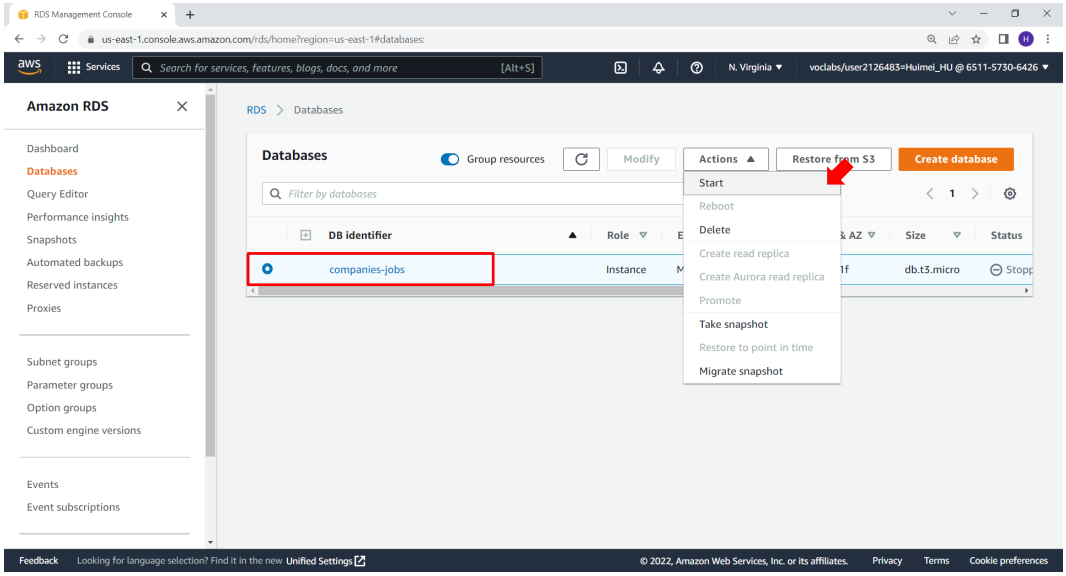



*Take note of the zipped file location as we will need to import it into Lambda in Section C.*

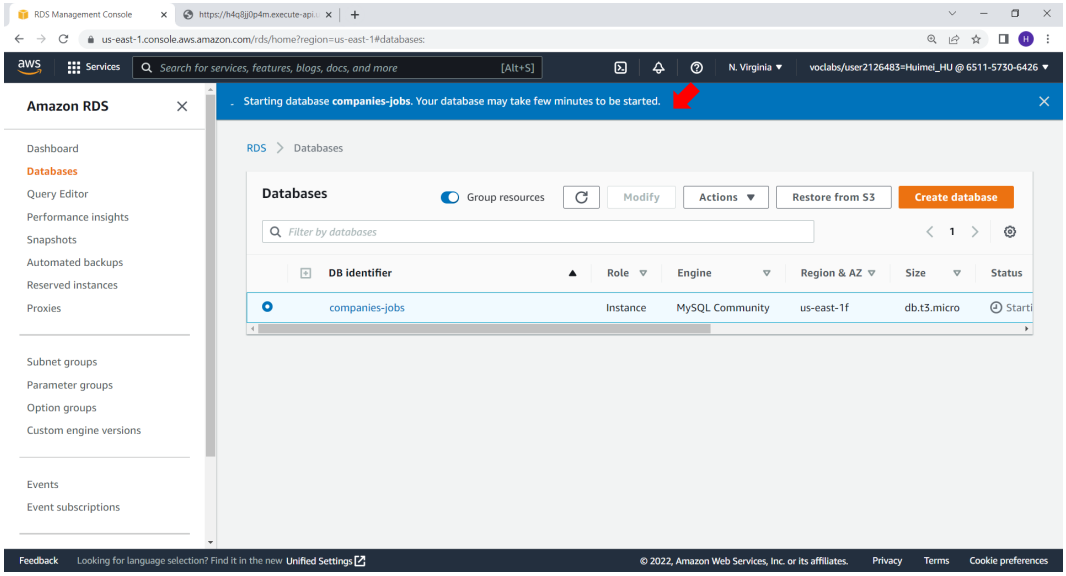
## B. Starting up RDS Database Instance

1. Launch **AWS Management Console**. In the Console Home screen, click on **Services** → **Database** → **RDS**.

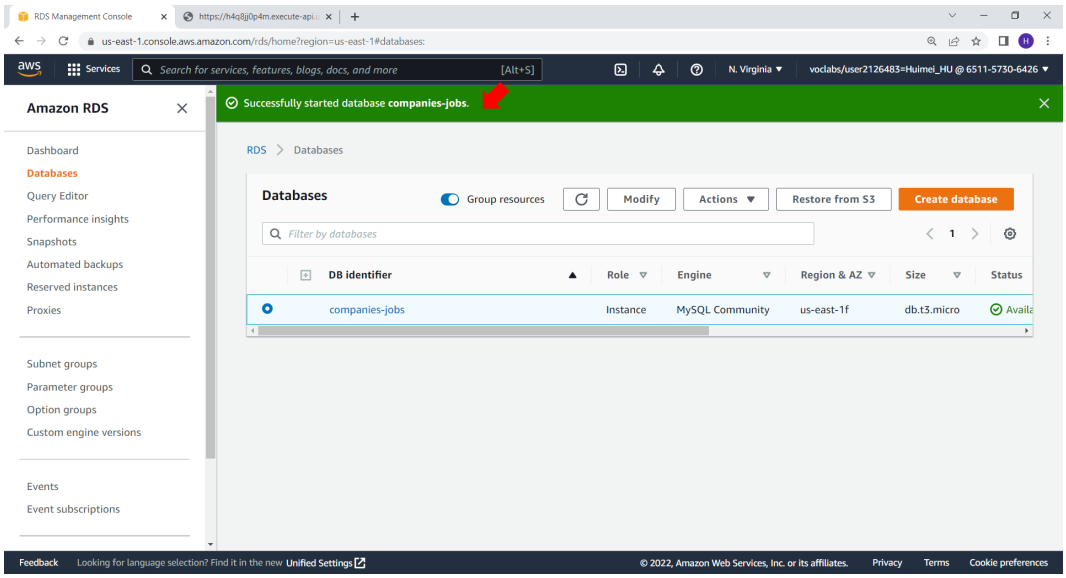
2. Click on the **radio button** beside **companies_job** database instance. Click on **Actions** and select **Start**.




3. Wait for a **few minutes** for the database instance to be started.

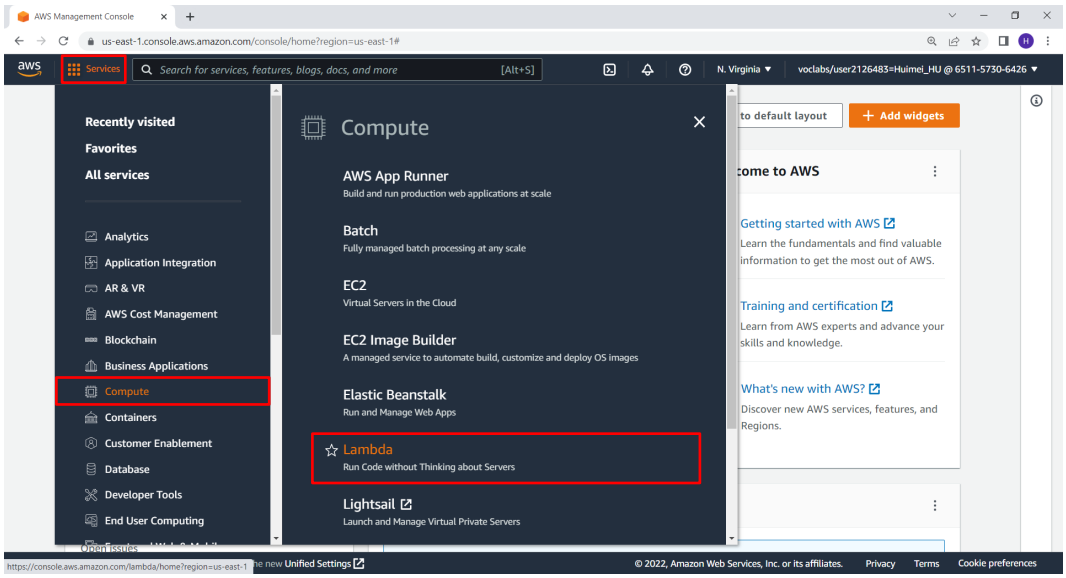


4. You will see the following screen when the database instance had started.



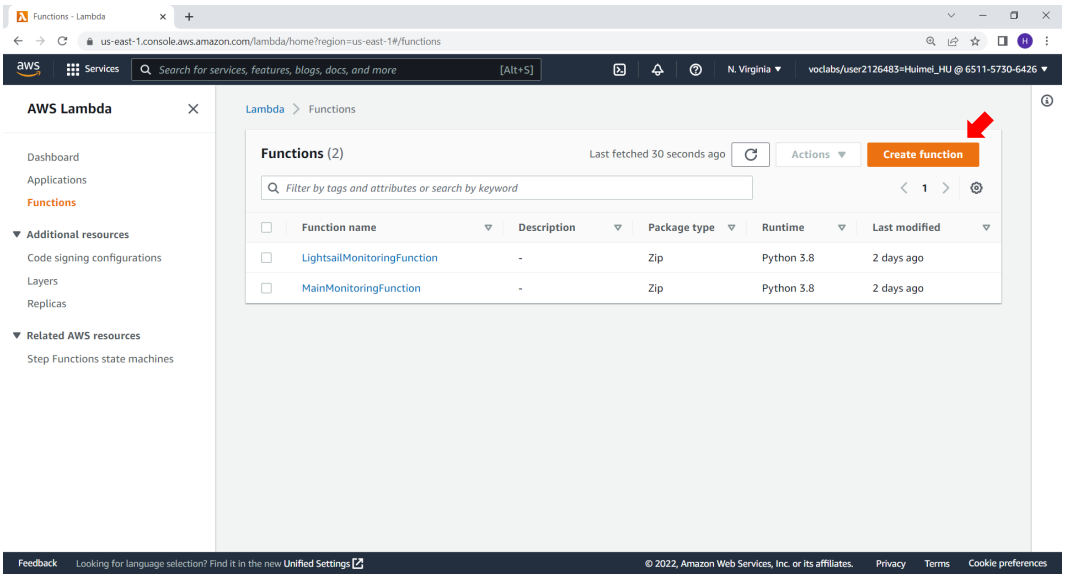

## C. Creating a Lambda Function

1. Launch **AWS Management Console**. In the Console Home screen, click on **Services** → **Compute** → **Lambda**.



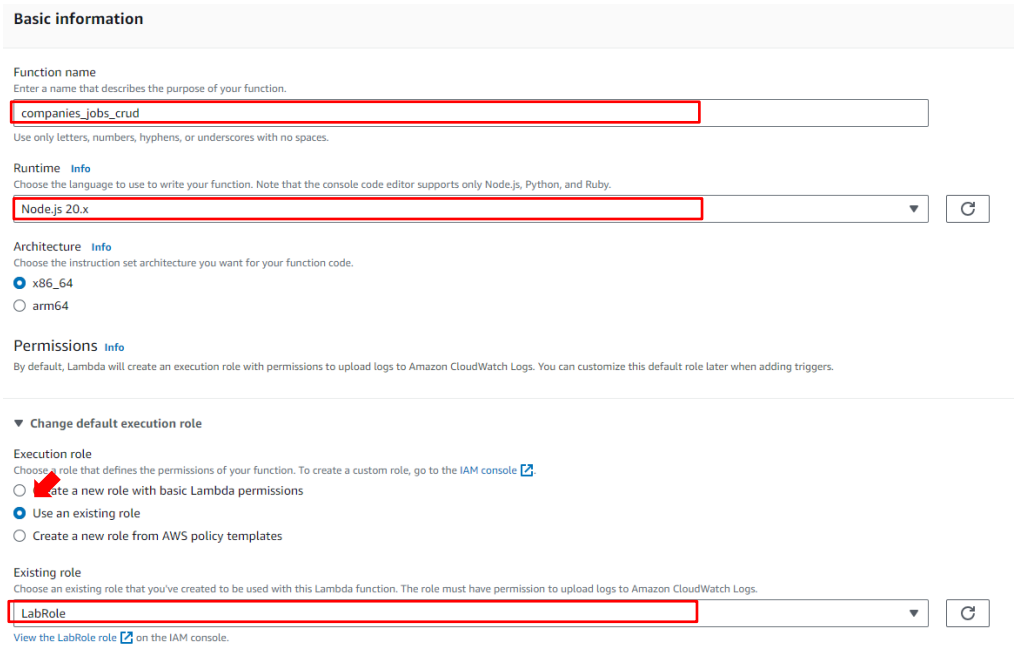

*Wait for a few seconds for the AWS Lambda to load.*

2. In the **Functions** screen, click on the **Create function** button.



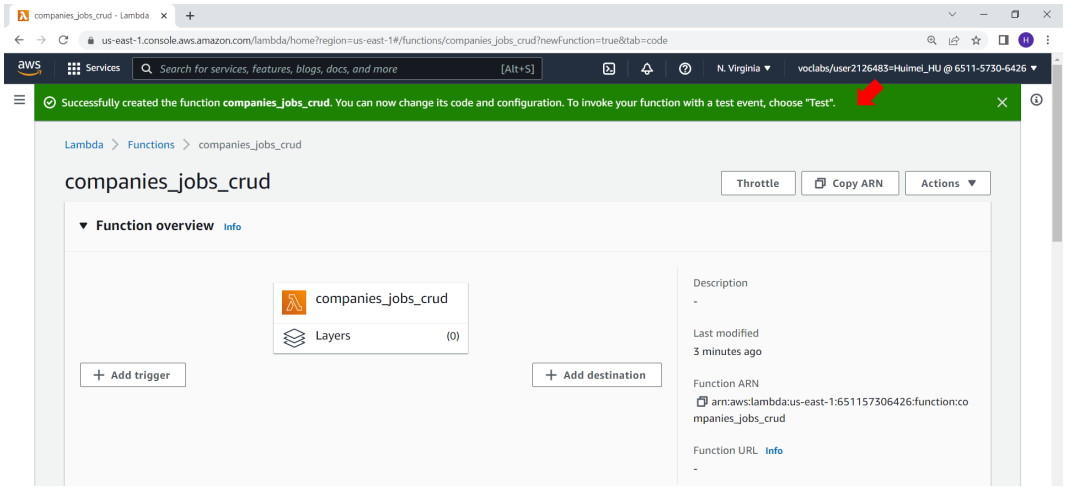

3. In the Create function screen, select **Author from Scratch**. Configure the function, as follows:

- **Function name**: companies_jobs_crud
- **Runtime**: Node.js 20.x
- **Change default execution role**: Use an existing role
- **Existing role**: LabRole

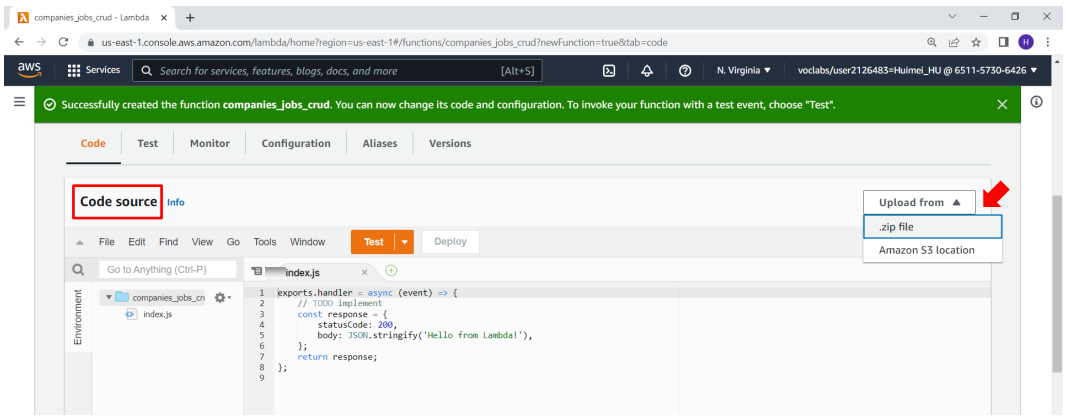



Click on **Create function**.

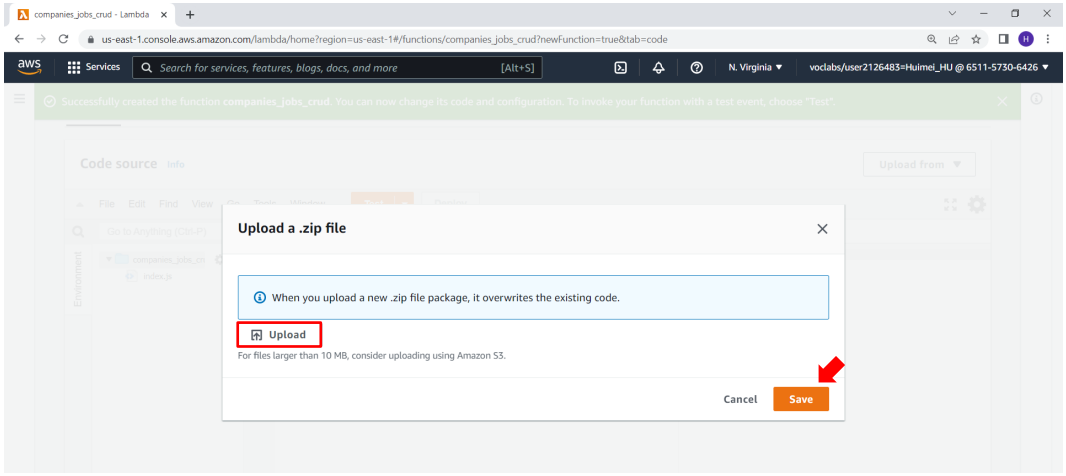
4. Wait a few seconds for the function to be created.




5. Scroll to the **Code source** section. Click on **Upload from** and select **.zip file**.

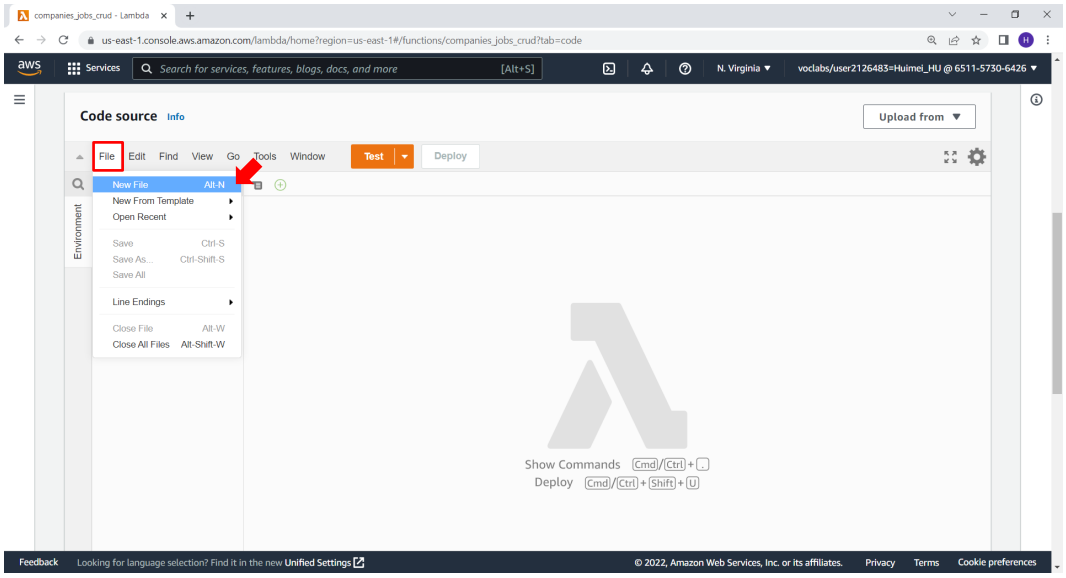



6. We will upload the Node.js project folder that we had created earlier on. Click on the **Upload** button and locate the **lambda_rds.zip** file.

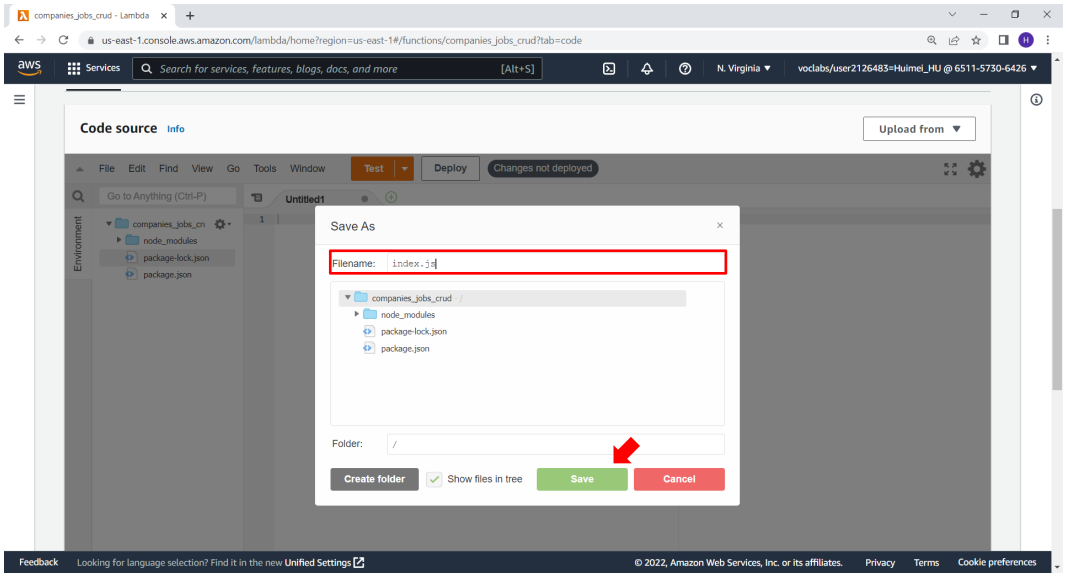



Click on **Save**.

7. We will upload re-create the **index.js** file. Click on **File** → **New File**.




8. Click on **File** → **Save As** and enter **index.js** as the **filename**.




Click on **Save**.

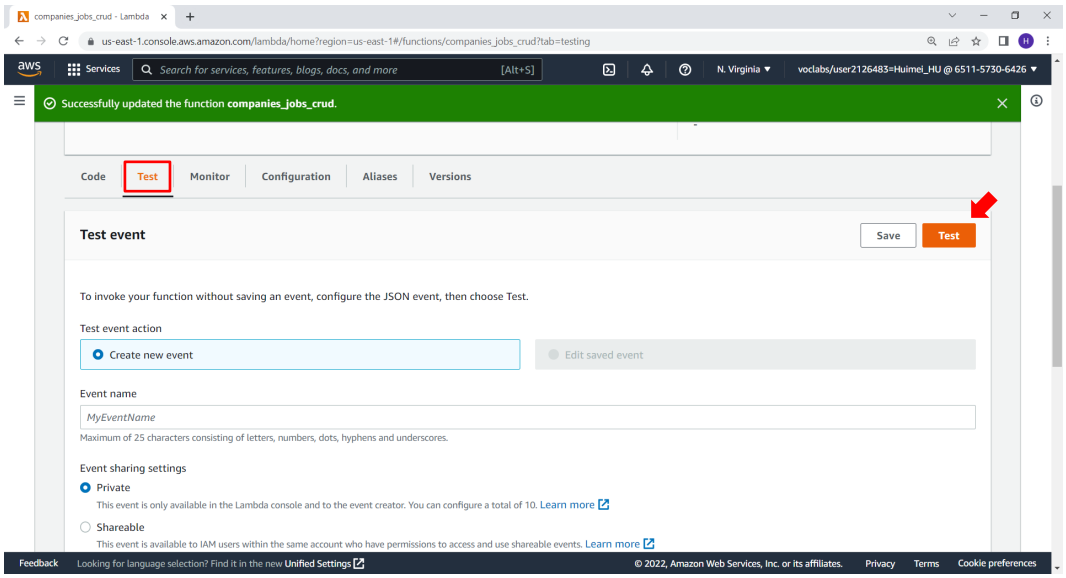
9. Enter the following to connect to the **Amazon RDS database** that was created in Lab 03. We will include a function to **retrieve all companies** from the database.

```
// Import mysql library
const mysql = require('mysql');

// Connect to the Amazon DBS companies_job that was created in Lab 03
const con = mysql.createConnection({
  host     : "companies-jobs.cw????ddcxpw.us-east-1.rds.amazonaws.com",
  user     : "admin",
  password : "password",
  port     : "3306",
  database  : "companies_jobs"
});

exports.handler =  (event, context, callback) => {

  let sql;
  context.callbackWaitsForEmptyEventLoop = false;

  sql = "SELECT * from `companies_jobs`.`jobs`;";
  con.query(sql, function (err, result) {
   if (err) throw err;
   return callback(null, result);
   });

};
```

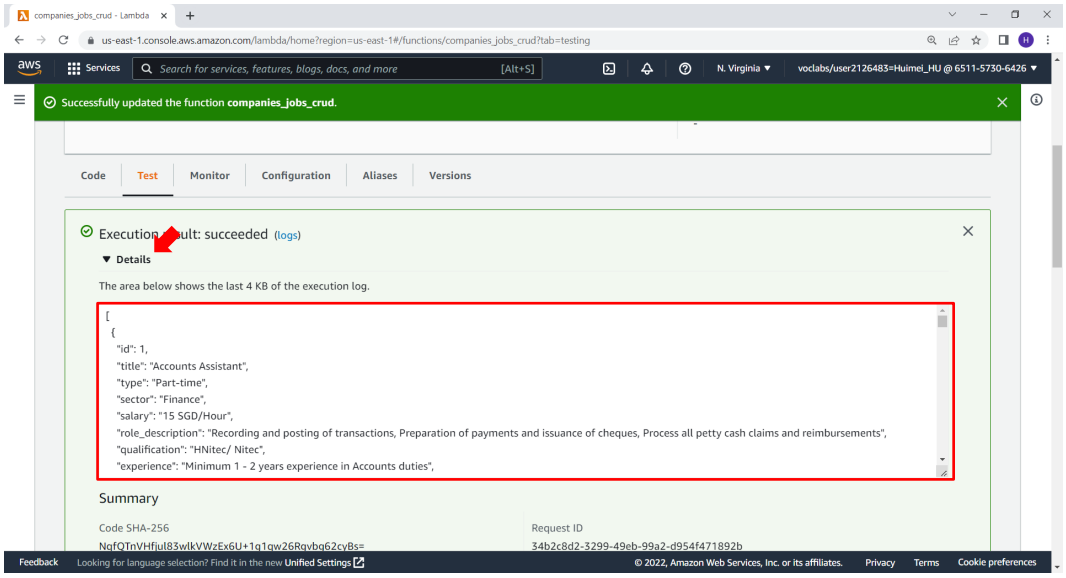
*Refer to Lab 04 to retrieve the hostname for your Amazon RDS database.*

Click on **Deploy** to save and deploy the code.

10. To test our Lambda function, click on the **Test** tab and **Test** button.



11. Expand **Details**. You should be able to see the retrieved jobs.




## D. Creating a Lambda Function – Event JSON

1. To re-use the same Lambda functions for multiple functionalities, we will need to give each functionality a name. We will give "**View all jobs**" the name of "**GET /jobs**". Let's switch back to the **Code** tab and modify the code in **index.js**, as shown:

```
// Import mysql library
const mysql = require('mysql');

// Connect to the Amazon DBS companies_job that was created in Lab 03
const con = mysql.createConnection({
  host     : "companies-jobs.cw????ddcxpw.us-east-1.rds.amazonaws.com",
  user     : "admin",
  password : "password",
  port     : "3306",
  database  : "companies_jobs"
});

exports.handler =  (event, context, callback) => {

  let sql;
  context.callbackWaitsForEmptyEventLoop = false;

  switch (event.routeKey) {
    case 'GET /jobs':
      sql = "SELECT * from `companies_jobs`.`jobs`;";
      con.query(sql, function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    default:
      throw new Error("Unsupported route: " + event.routeKey);
  }

};
```

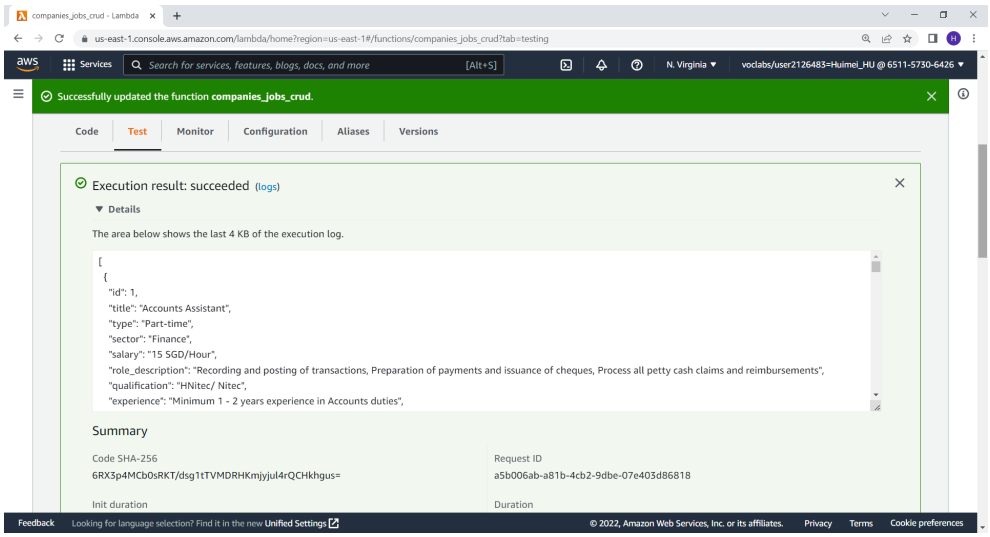
Click on **Deploy** to save and deploy the code.

2. To test our Lambda function, click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** for our event:

```
{
  "routeKey" : "GET /jobs"
}
```

Click on the **Test** button.

3. Expand **Details**. You should still be able to see the retrieved jobs.

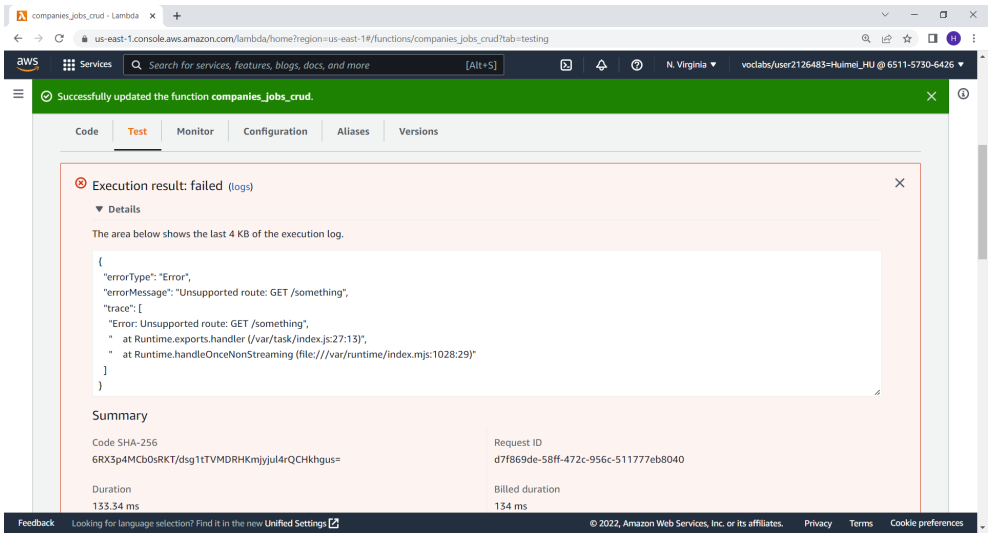



4. Let's enter an invalid routeKey. Scroll to the **Event JSON** section and enter the following:

```
{
  "routeKey" : "GET /something"
}
```

Click on the **Test** button.

5. Expand **Details**. You should see an error.



## E. Creating a Lambda Function - Retrieve

In this section, we will complete the implementation of all the **Retrieve** functionalities.

| Functionality | routeKey | SQL Command |
|---|---|---|
| View all jobs | `GET /jobs` | `SELECT * from `companies_jobs`.`jobs`;` |
| View jobs by companies | `GET /jobs_companies/{id}` | `SELECT * FROM `companies_jobs`.jobs WHERE company_id = ?;` |
| View selected job | `GET /jobs/{id}` | `SELECT * FROM companies_jobs.companies, companies_jobs.jobs WHERE companies.id = company_id AND jobs.id = ?;` |

To **access the id** in the routeKey, we will make use of **pathParameters**.

1. Switch back to the **Code** tab and modify the code in **index.js**, as shown:

```
// Import mysql library
const mysql = require('mysql');

// Connect to the Amazon DBS companies_job that was created in Lab 03
const con = mysql.createConnection({
  host     : "companies-jobs.cw????ddcxpw.us-east-1.rds.amazonaws.com",
  user     : "admin",
  password : "password",
  port     : "3306",
  database  : "companies_jobs"
});

exports.handler =  (event, context, callback) => {

  let sql;
  context.callbackWaitsForEmptyEventLoop = false;

  switch (event.routeKey) {
    case 'GET /jobs':
      sql = "SELECT * from `companies_jobs`.`jobs`;";
      con.query(sql, function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    case 'GET /jobs_companies/{id}':
      sql = "SELECT * FROM `companies_jobs`.jobs WHERE company_id = ?;";
      con.query(sql, [event.pathParameters.id], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    case 'GET /jobs/{id}':
      sql = "SELECT * FROM companies_jobs.companies, companies_jobs.jobs
WHERE companies.id = company_id AND jobs.id = ?;";
      con.query(sql, [event.pathParameters.id], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    default:
      throw new Error("Unsupported route: " + event.routeKey);
  }

};
```

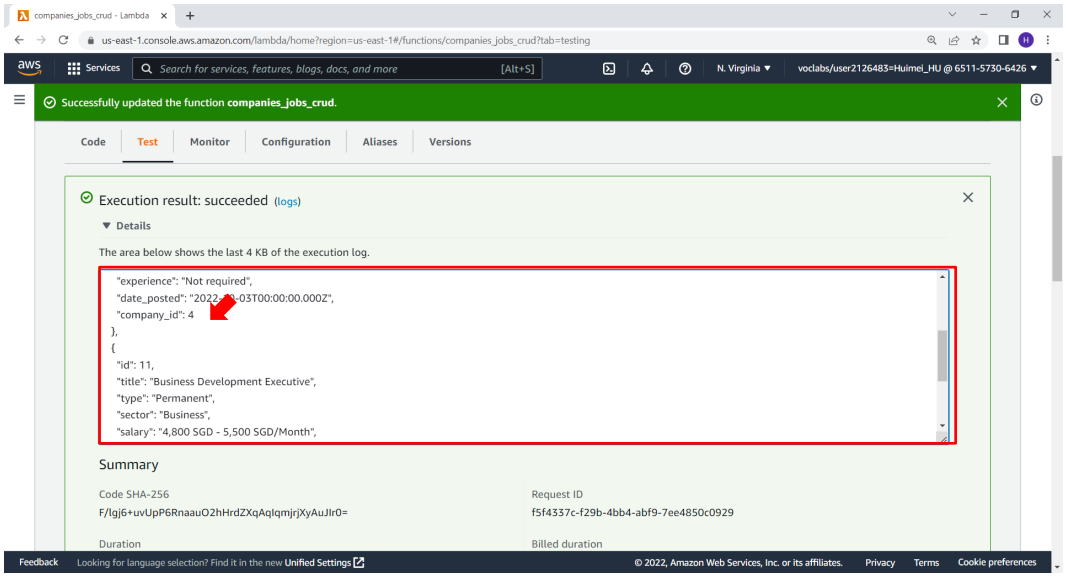
2. Click on **Deploy** to save and deploy the code.

3. Let's test the "View jobs by companies" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** and **pathParameters** for our event:

```
{
  "routeKey" : "GET /jobs_companies/{id}",
  "pathParameters": {"id": 4}
}
```

Click on the **Test** button.

4. Expand **Details**. You should still be able to see the retrieved information of jobs offered by company id 4.

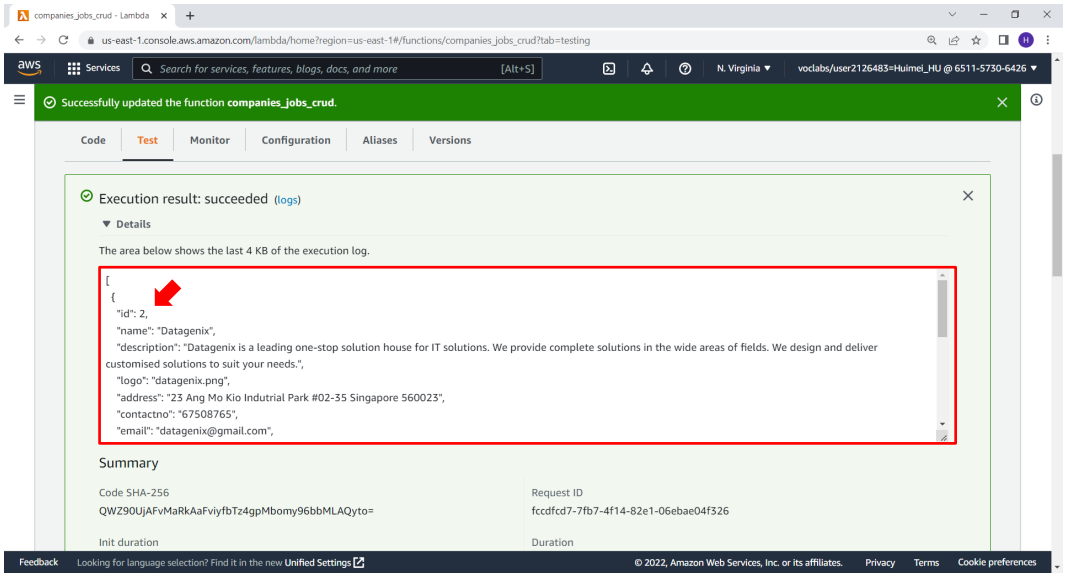



5. To test our " View selected job" functionality, scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** and **pathParameters** for our event:

```
{
  "routeKey" : "GET /jobs/{id}",
  "pathParameters": {"id": 2}
}
```

Click on the **Test** button.

6. Expand **Details**. You should still be able to see the retrieved information of job id 2.




## F. Creating a Lambda Function – Create/ Login

In this section, we will complete the implementation of the **Create** and **login** functionalities. These functionalities are similar as we are expecting a **Form** for the user to fill up before they create a job listing or for user to enter their credentials. We will make use of **body** to access these values.

| Functionality | routeKey | SQL Command |
|---|---|---|
| Add job | POST /jobs | INSERT INTO `companies_jobs`.`jobs` (`title`, `type`, `sector`, `salary`, `role_description`, `qualification`, `experience`, `date_posted`, `company_id`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?); |
| Login | POST /login | SELECT * FROM `companies_jobs`.`companies` WHERE `email` = ? AND `password` = ?; |

1. Switch back to the **Code** tab and modify the code in **index.js**, as shown:

```
...

exports.handler =  (event, context, callback) => {

  ...

  switch (event.routeKey) {
    ...
    case 'POST /jobs':
      body  = JSON.parse(event.body)
      sql = "INSERT INTO `companies_jobs`.`jobs` (`title`, `type`, `sector`, `salary`, `role_description`, `qualification`, `experience`, `date_posted`, `company_id`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);";
      con.query(sql, [body.title, body.type, body.sector, body.salary, body.role_description, body.qualification, body.experience,  new Date().toISOString().slice(0, 10), body.company_id], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
```

```
    case 'POST /login':
      body = JSON.parse(event.body)
      sql = "SELECT * FROM `companies_jobs`.`companies` WHERE `email` = ?
AND `password` = ?;";
      con.query(sql, [body.email, body.password], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    default:
      throw new Error("Unsupported route: " + event.routeKey);
  }

};
```

*To access the input values, we make use of **body**, parse it into JSON format and access the individual values by specifying the key. **POST /jobs** adds a new record in the jobs table and **POST /login** retrieves the company information if the email and password provided matches.*

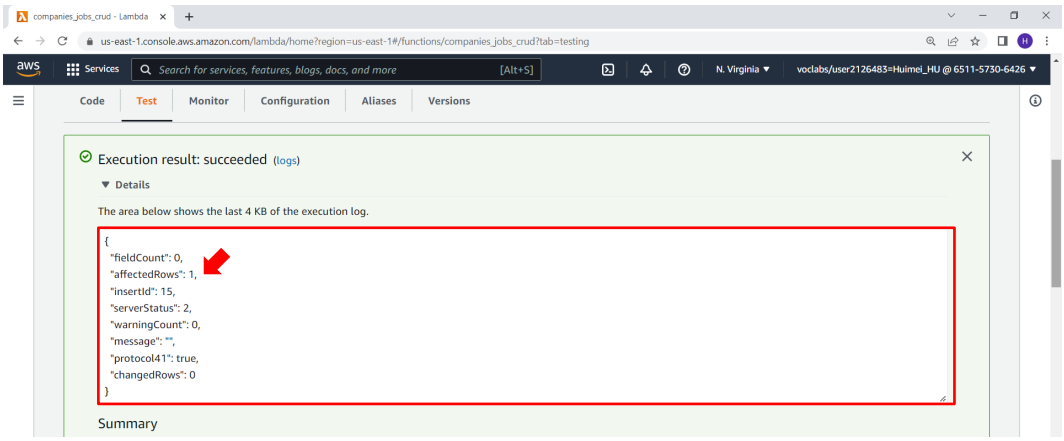
2. Click on **Deploy** to save and deploy the code.

3. Let's test the "Add job" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** and **body** for our event:

```
{
    "routeKey": "POST /jobs",
    "body": "{ \"title\": \"Associate Engineer\", \"type\": \"Permanent\",
      \"sector\": \"IT\", \"salary\": \"2,000 SGD - 2,800 SGD/Month\",
      \"role_description\": \"You will support a team of Senior IT
      Consultants to deliver solutions to company's clients. You will learn
      about the latest industry cybersecurity products and how it can be
      used to solve customer's problems. Our experience staff will show you
      the ropes, teach and guide you so you can be successful in your role.
      This job is exciting and challenging as you come to interact with
      customers across various industries and you help tackle challenges at
      various levels. The realm of cybersecurity is highly regarded, now is
      the time to join us and be part of the exciting team that tackles
      real world problems!\", \"qualification\": \"Diploma\",
      \"experience\": \"Not Required\", \"company_id\": 1 }"
}
```

Click on the **Test** button.

4. Expand **Details**. You should see the value of **affectedRows** is 1, indicating that one row is added.

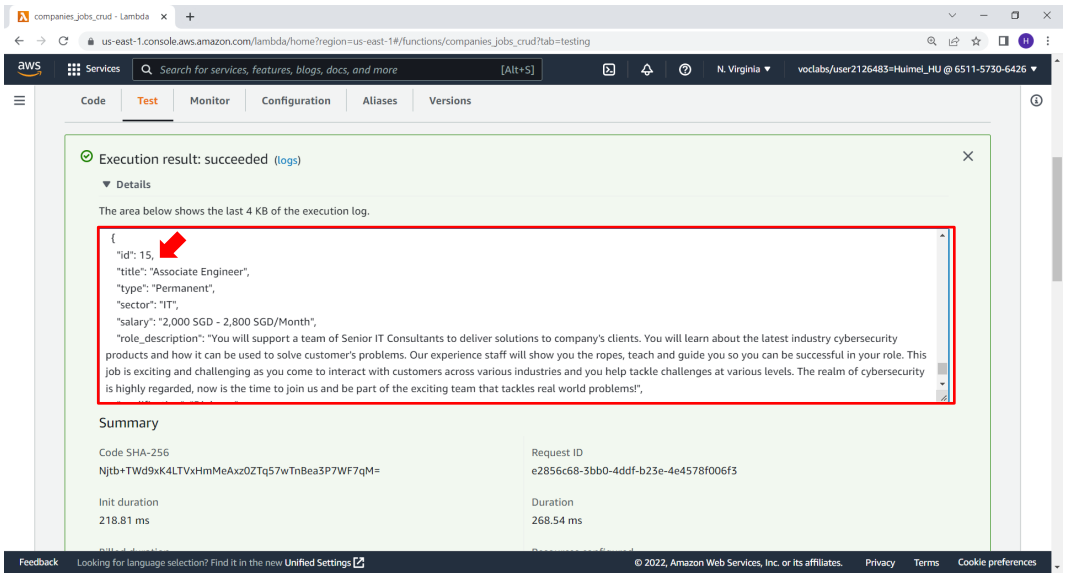


5. Let's verify that a new job is added using the "View all jobs" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey**:

```
{
    "routeKey": "GET /jobs"
}
```

Click on the **Test** button.

6. Expand **Details**. Scroll to the bottom of the result. You should see the new job information.

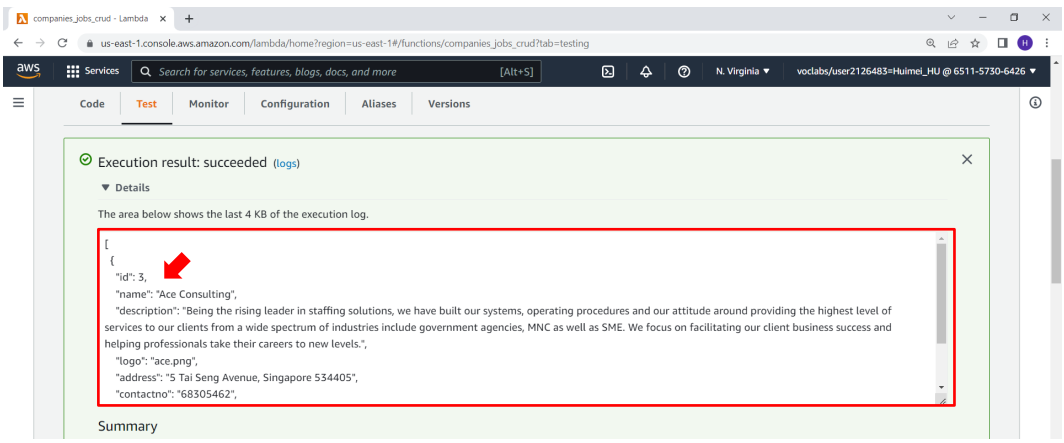



7. Let's test the "Login" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** and **body** for our event:

```
{
  "routeKey": "POST /login",
  "body": "{\"email\": \"ace@gmail.com\", \"password\": \"123456\"}"
}
```

Click on the **Test** button.

8. Expand **Details**. You should see the **company information** since the email/password matches.

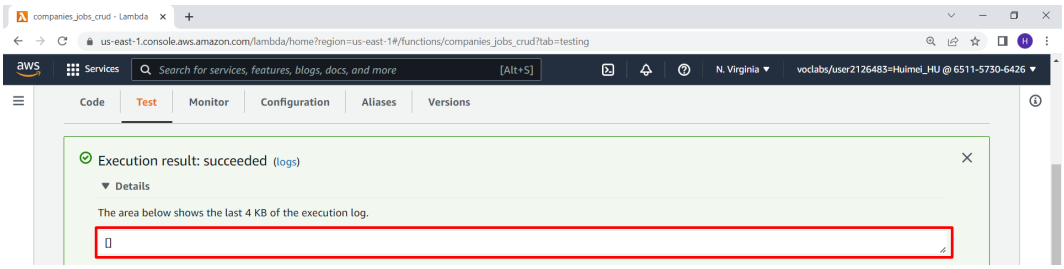


9. Let's verify that if the email/password does not match, no company information should be returned. Scroll to the **Event JSON** section and enter the following JSON data:

```
{
  "routeKey": "POST /login",
  "body": "{\"email\": \"ace@gmail.com\", \"password\": \"654321\"}"
}
```

Click on the **Test** button.

10. Expand **Details**. Scroll to the bottom of the result. You should see the an empty array.




## G. Creating a Lambda Function – Update

In this section, we will complete the implementation of the **Edit** functionality. These functionalities are similar to Section F as we are expecting a **Form** for the user to fill up before they edit a job listing. We will make use of **body** to access these values and **pathParameters** to retrieve the job id.

| Functionality | routeKey | SQL Command |
|---|---|---|
| Edit selected job | PUT /jobs/{id} | UPDATE `companies_jobs`.`jobs` SET `title` = ?, `type` = ?, `sector` = ?, `salary` = ?, `role_description` = ?, `qualification` = ?, `experience` = ? WHERE id = ?; |

1. Switch back to the **Code** tab and modify the code in **index.js**, as shown:

```
...

exports.handler =  (event, context, callback) => {
  ...

  switch (event.routeKey) {
    ...
    case 'PUT /jobs/{id}':
      body  = JSON.parse(event.body)
      sql = "UPDATE `companies_jobs`.`jobs` SET `title` = ?, `type` = ?,
`sector` = ?, `salary` = ?, `role_description` = ?, `qualification` = ?,
`experience` = ? WHERE id = ?;";
      con.query(sql, [body.title, body.type, body.sector, body.salary,
body.role_description, body.qualification, body.experience,
event.pathParameters.id], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    default:
      throw new Error("Unsupported route: " + event.routeKey);
  }
};
```

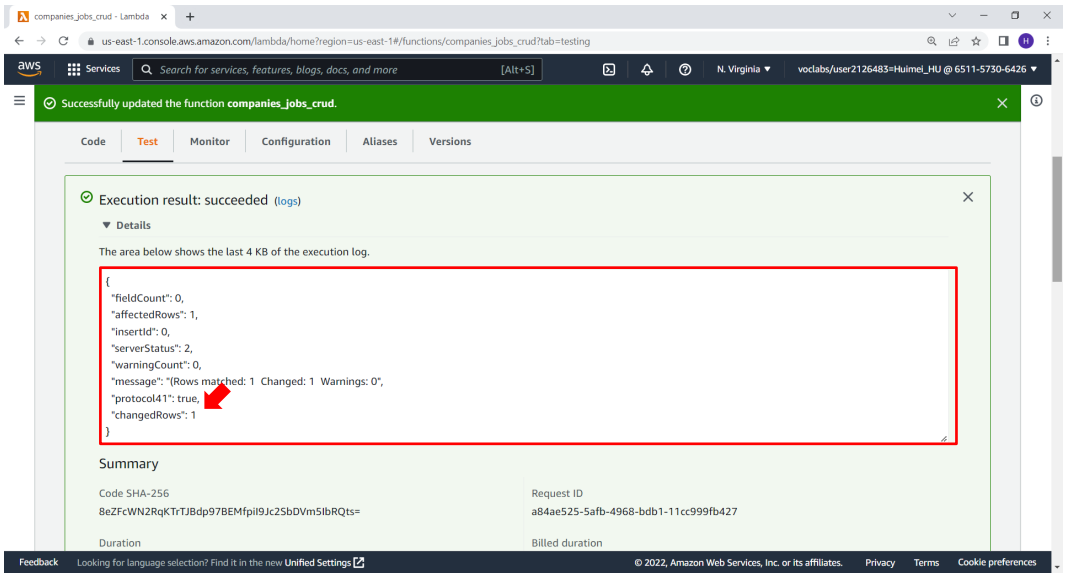
2. Click on **Deploy** to save and deploy the code.

3. Let's test the "Edit selected job" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey**, **pathParameters** and **body** for our event:

```
{
    "routeKey": "PUT /jobs/{id}",
    "pathParameters": {"id": 15},
    "body": "{ \"title\": \"Associate Engineer\", \"type\": \"Permanent\",
      \"sector\": \"IT\", \"salary\": \"2,200 SGD - 3,000 SGD/Month\",
      \"role_description\": \"You will support a team of Senior IT
      Consultants to deliver solutions to company's clients. You will learn
      about the latest industry cybersecurity products and how it can be
      used to solve customer's problems. Our experience staff will show you
      the ropes, teach and guide you so you can be successful in your role.
      This job is exciting and challenging as you come to interact with
      customers across various industries and you help tackle challenges at
      various levels. The realm of cybersecurity is highly regarded, now is
      the time to join us and be part of the exciting team that tackles
      real world problems!\", \"qualification\": \"Diploma\",
      \"experience\": \"Not Required\" }"
}
```

Click on the **Test** button.

4. Expand **Details**. You should see the value of **changedRows** is 1, indicating that one row is updated.

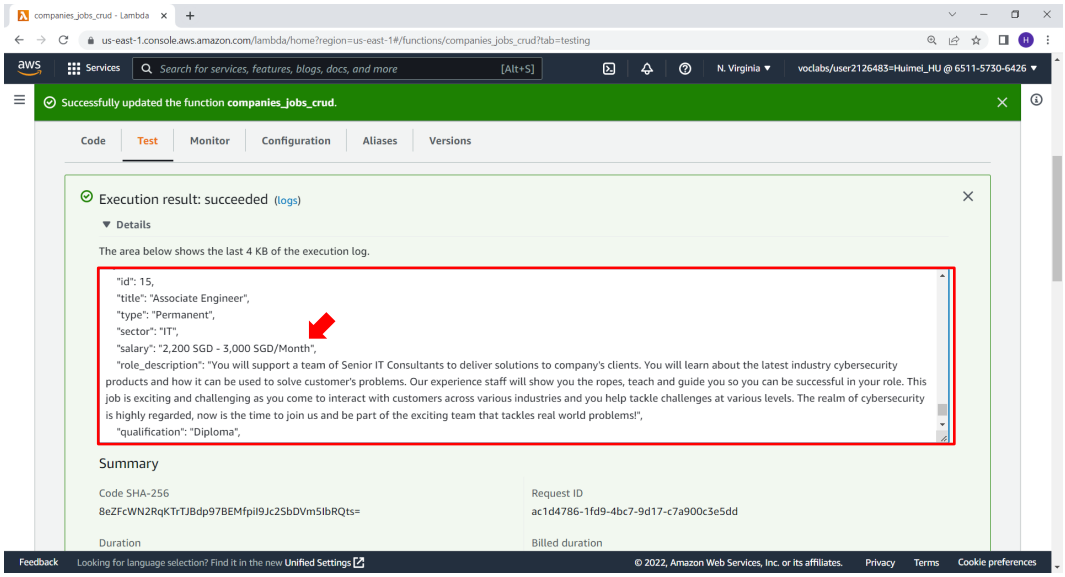



5. Let's verify that the **salary** value for **job id 15** is updated using the "View all jobs" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey**:

```
{
    "routeKey": "GET /jobs"
}
```

Click on the **Test** button.

6. Expand **Details**. Scroll to the bottom of the result. You should see the updated information.




## H. Creating a Lambda Function – Delete

In this section, we will complete the implementation of the **Delete** functionality. We will make use of **pathParameters** to retrieve the job id.

| Functionality | routeKey | SQL Command |
|---|---|---|
| Delete selected job | `DELETE /jobs/{id}` | `DELETE FROM `companies_jobs`.`jobs` WHERE id = ?` |

1. Switch back to the **Code** tab and modify the code in **index.js**, as shown:

```
...

exports.handler =  (event, context, callback) => {
  ...

  switch (event.routeKey) {
    ...
    case 'DELETE /jobs/{id}':
      sql = "DELETE FROM `companies_jobs`.`jobs` WHERE id = ?;";
      con.query(sql, [event.pathParameters.id], function (err, result) {
            if (err) throw err;
            return callback(null, result);
            });
            break;
    default:
      throw new Error("Unsupported route: " + event.routeKey);
  }
};
```

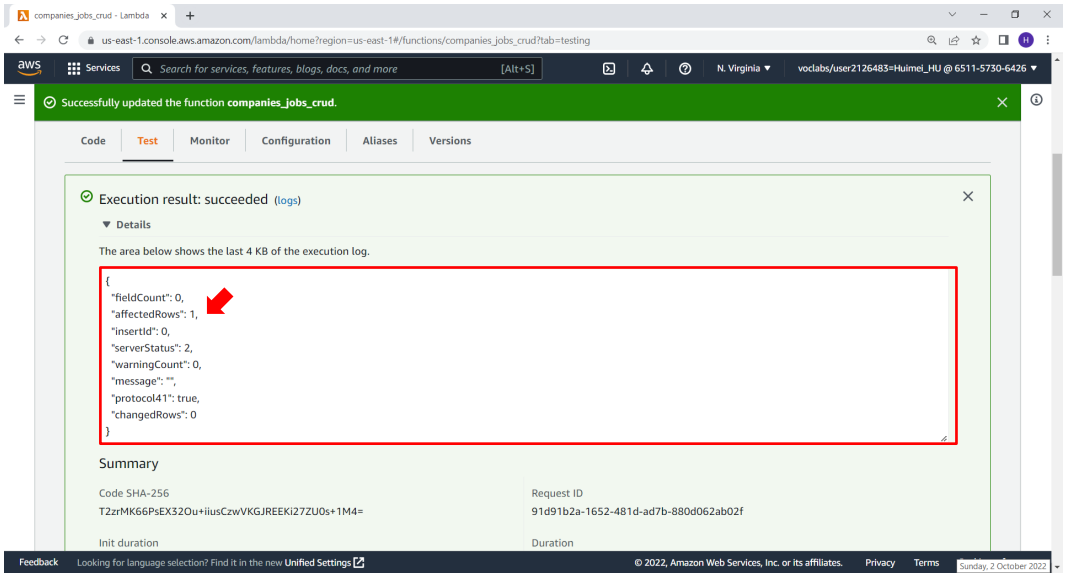
2. Click on **Deploy** to save and deploy the code.

3. Let's test the "Delete selected job" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey** and **pathParameters** for our event:

```
{
    "routeKey": "DELETE /jobs/{id}",
    "pathParameters": {"id": 15}
}
```

Click on the **Test** button.

4. Expand **Details**. You should see the value of **affectedRows** is 1, indicating that one row is deleted.

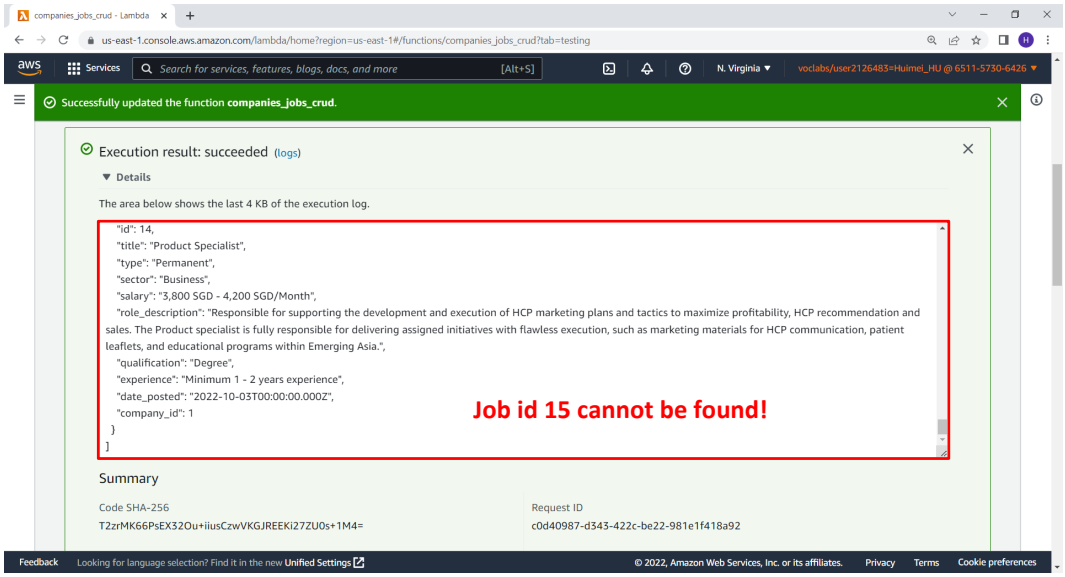



5. Let's verify that **job id 15** is deleted using the "View all jobs" functionality. Click on the **Test** tab. Scroll to the **Event JSON** section and enter the following JSON data to include a **routeKey**:

```
{
    "routeKey": "GET /jobs"
}
```

Click on the **Test** button.

6. Expand **Details**. Scroll through the result, you should see that job id 15 is removed.



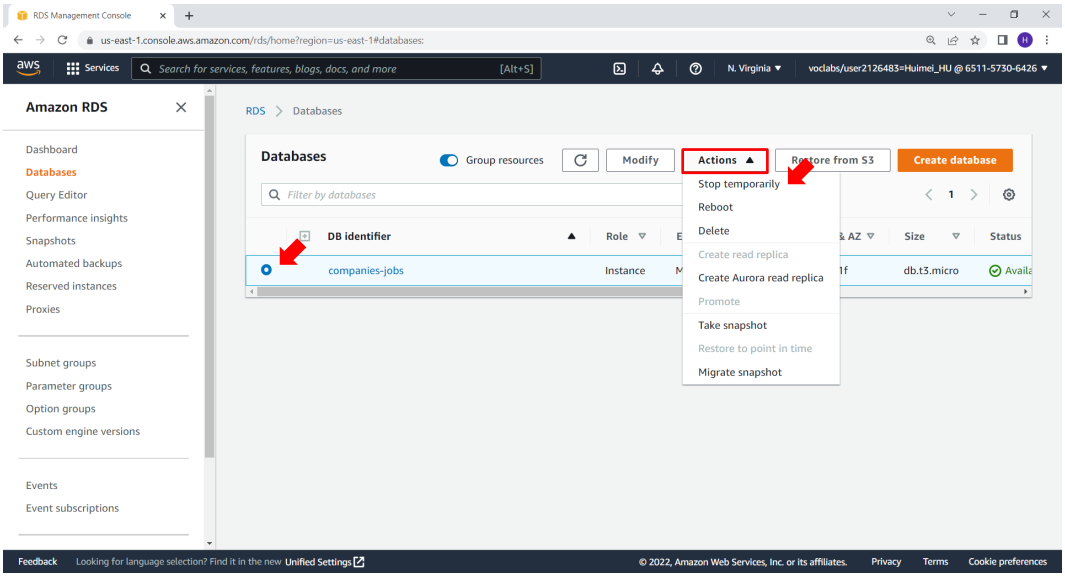

## I. Lab Cleanup

To avoid incurring steep charges, we will stop the DB instance temporarily when it is not in use.

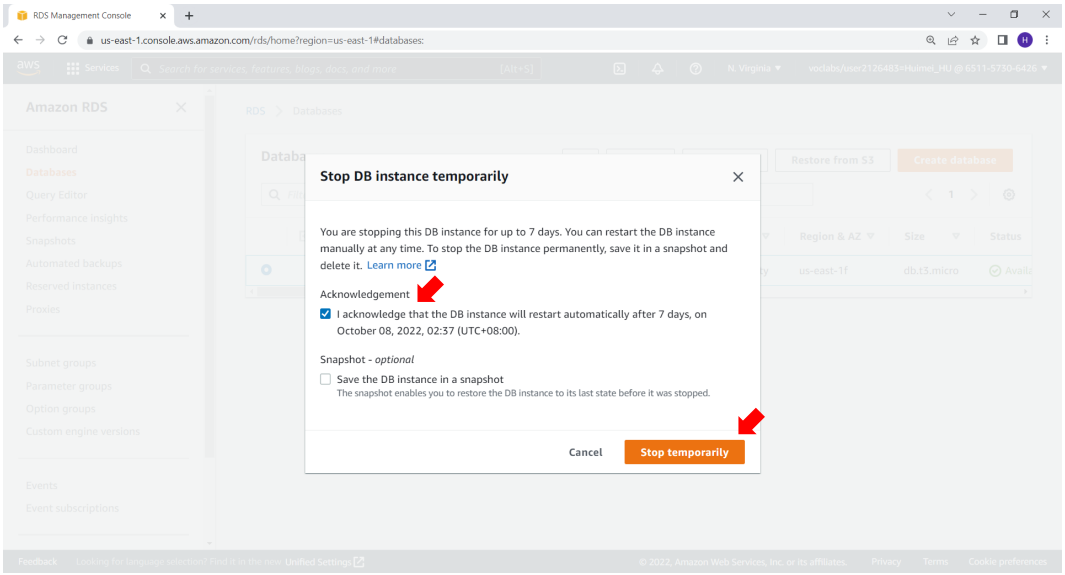
1. Click on **Services** → **Database** → **RDS** to switch to the RDS Dashboard.

2. In the RDS **Dashboard**, select **Databases** in the menu on the left-hand side.

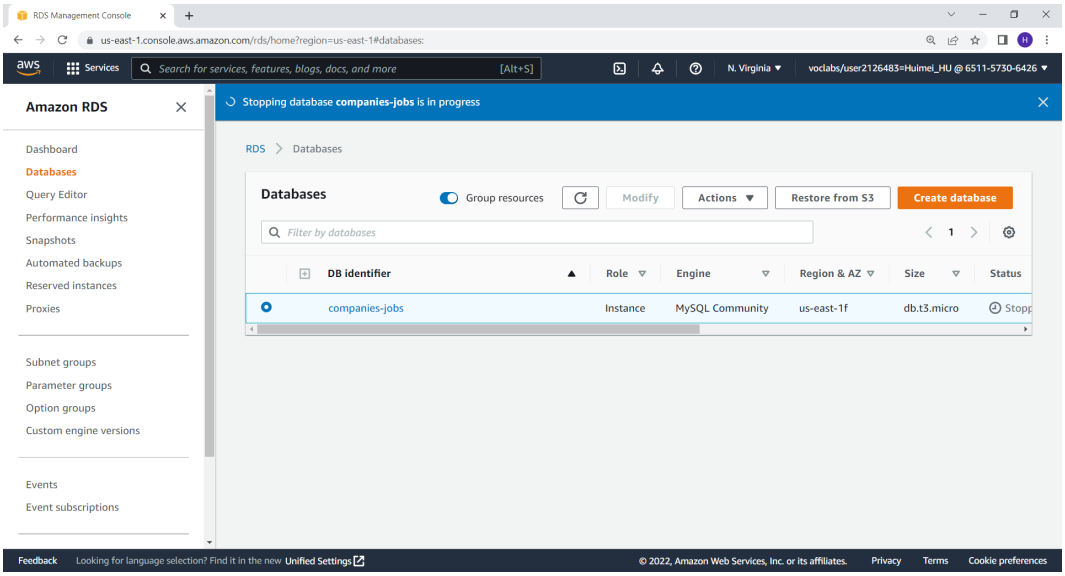
3. Select on the **radio button** beside **companies_jobs** → Click on **Actions** dropdown menu → Select **Stop temporarily**.



4. **Acknowledge** that the DB instance will restart after 7 days by checking on the checkbox and click on **Stop temporarily**.




5. Allow a **few minutes** for the DB instance to stop temporarily.




Set a **reminder** to stop the DB instance again **every 7 days** to avoid incurring steep charges.

End of Lab