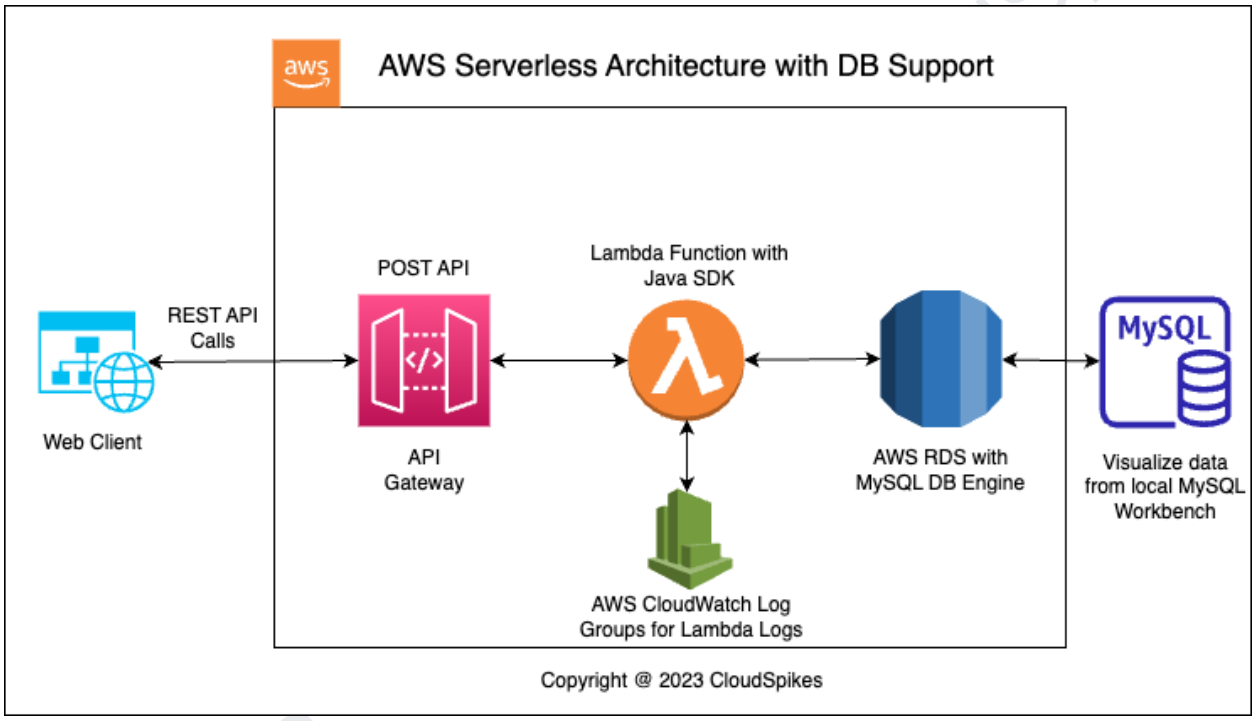
#10 Task: Create a POST API Gw API with Lambda that has a Java function to store data directly to the RDS DB Instance with MySQL DB Engine with the latest stable version. Visualize the data stored in the DB using MySQL Workbench or a similar tool.



1. Java App Code - Create a Lambda Function with Java SDK.

- Open Eclipse and install AWS Toolkit to prepare AWS Lambda specific Java code using Maven build tool configurations using this AWS Official document.
- Now, from the menu bar, click on New → Project → Seach for “AWS Lambda” → Select **AWS Lambda Java Project** → Provide the name of the project and select the Input Type as **Stream Request Handler** → Hit Finish.
- This will create a new project in the project explorer section on left hand side with the project name folder you provided in the previous step.
- Now, refer to the source code from this GitHub Repository and prepare Java class files as well as pom.xml dependency file accordingly.

Note: To skip the above steps, you can also simply import this GitHub Repository in to Eclipse IDE and compare the code with the GitHub code base to verify the Java and pom.xml files.

- FYI, this Java code will capture the input JSON data from the API Gateway POST API, process the input data as well as execute the SQL Queries on RDS MySQL Instance.
- Finally, to prepare the executable JAR build for this project, go to the terminal and navigate to the location of the AWS Lambda Java Project. Then. execute "**mvn clean install**" command. This will prepare the JAR file under **target** directory which will be uploaded to the AWS Lambda Function while deployment process will be executed. ***Note***: You can download the Java source code for this Lab work from [this public GitHub Repo](#).

1. Infra Setup - Setup RDS DB Service on AWS for MySQL DB Server.

- Create a RDS DB Instance by navigating to RDS service and clicking on **Create database** button.
- A web form will appear on the screen where you need to fill up all the configurable value to create a RDS Instance.
- Select a database creation method as **Standard create** and **DB Engine** type as MySQL (Not Aurora (MySQL Compatible) option).
- Leave rest of the values such as DB Version as latest with default values and choose template as **Free tier**.
- In the settings tab, leave DB Identifier as default value and enter **DB User** name and **Password** (Do not select Manage master credentials in AWS Secrets Manager and Auto generate a password check boxes).
- In the instance type selection, choose **db.t2.micro** to save cost in the practice lab work.
- Leave **Storage** configs as default values and in **Connectivity** select **Public access Yes** and select **Availability Zone** as per your preference.
- In the **Additional configuration** under **Database options** - set **Initial database name** as **rds_init_db**.
- Keep rest of the settings as default values and hit **Create database** button to finish creating the MySQL DB Instance.

Post RDS DB Creation Steps:

- Once the RDS DB Instance is up and running, you can navigate to the RDS Instance → **Connectivity & security** tab under **Endpoint & port** copy the Endpoint value.
- Go to your local MySQL workbench tool and test the created DB Instance by clicking on Database option from tool bar → Connect to Database…
- Paste endpoint value in the Hostname text box and DB user credentials as per the values provided while creating the RDS DB Instance.
- You can execute SQL queries in the MySQL Workbench Query editor tool and test the RDS DB Connectivity by executing a test SQL query as below:
  SELECT * FROM sys.sys_config;
- Once DB Connectivity testing is done successfully, you need to replace these DB Configuration valuesin the MySQL Connection Java file in your Lambda source code.

2. Infra Setup - Create a Lambda Function with Java SDK.

- Create a Lambda function and provide the basic details like Function name, Runtime as Java 11 (Corretto), Architecture type as x86_64 and Change default execution role option as Create a new role with basic Lambda permissions.
- Once, the Lambda is created with the basic details, go to Runtime settings and set Handler value as
  com.amazonaws.lambda.demo.LambdaFunctionHandler::handleRequest
- Here, **com.amazonaws.lambda.demo** is the Package value for the Java App, **LambdaFunctionHandler** is the Java Class name where Handler Function (App Java Code Entry Point from where Lambda execution code will start reading Java code from Line 0 to Line N - Last line of code) is defined and to define the Lambda Handler Function is distingueshed using :: (Double column). So, here, **handleRequest** is the Java Lambda Handler Function name.
- Furthermore, configure the environment variables needed for the Lambda function such as AWS_AK, AWS_SK, AWS_DEFAULT_REGION_VALUE and AWS_ACCOUNT_ID by navigating to **Configuration tab** → **Environment Variables** section.
- Deploy the JAR build of the Java App on the prepared AWS Lambda function by navigating to **Code tab** → **Code Source** section and selecting **Upload from** → **.zip or .jar file** and upload the built JAR file using Maven build tool from the build server/machine. In this case, we prepared JAR file on local env, so upload it from the local machine.

3. Infra Setup - Create a REST API with required Resource and Method.

- Create a REST API by going to the API Gateway AWS service.
- Define an API Resource like **test-rds-db-ops** by clicking on Actions button and selecting the root / Resource.
- Once the Resource is defined, click on the created **test-rds-db-ops** Resource and click on Create Method button via Actions button.
- Select a POST method for this API resource and then select integration type Lambda and map it with the created Lambda function in step 1.
- Finally, select the root / Resource and click on Actions button. Select Deploy API option and create/use the Stage to deploy the APIs defined.