**Steps For creating My project:**

*Step 1:* First in AWS an account is created and then logged in through using root user.

*Step 2:* Then A flow Chart is created for creation and understanding for my project.

*Step 3:*

So, for the above project

which is

**Personal Finance Tracker**

of

which includes the following

permissions policies:

[**AdministratorAcces**](https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAdministratorAccess)

[**s**](https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAdministratorAccess)

**AmazonAPIGatewayAdministrator**

**AmazonAPIGatewayInvokeFullAccess**

**AmazonS3FullAccess**

**CloudWatchFullAccess**

**CloudWatchFullAccessV2**

IAM roles

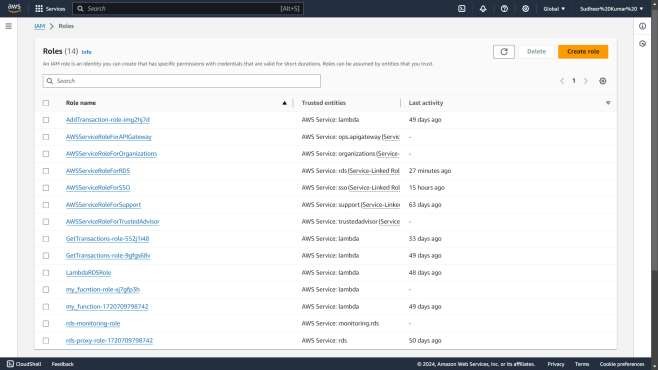
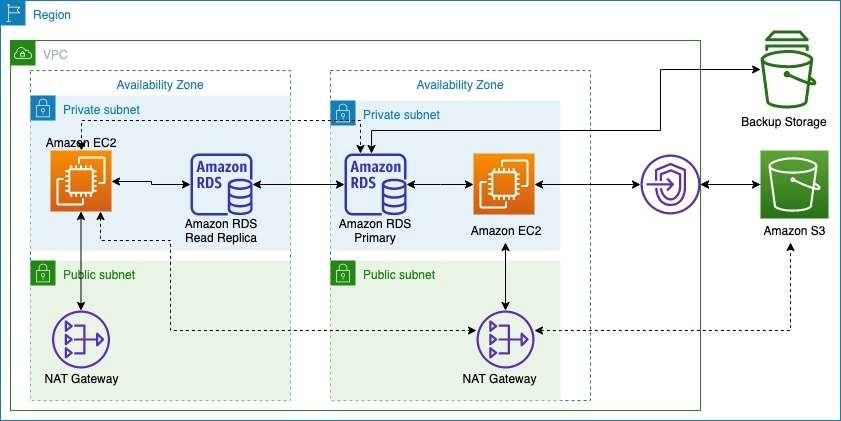
where

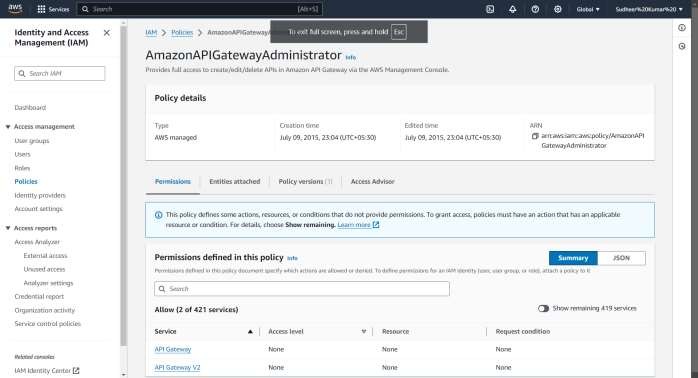
**AmazonAPIGatewayAdministrator**

i

s

made.





*Step 4:* Created an RDS Database name ‘**database-1’.**

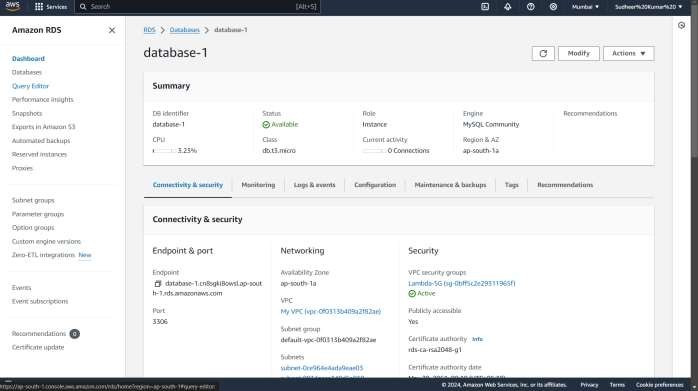


 **-** **-** **-**

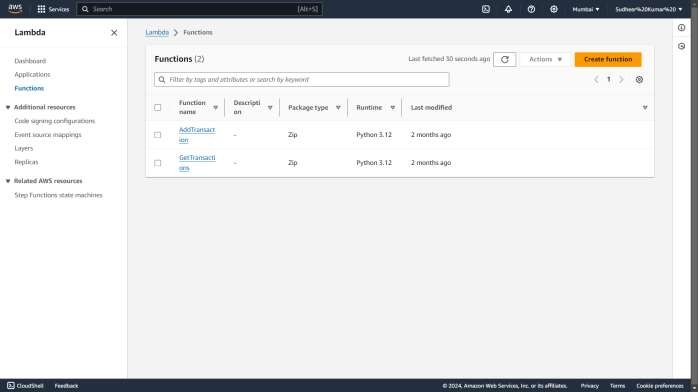


**-**



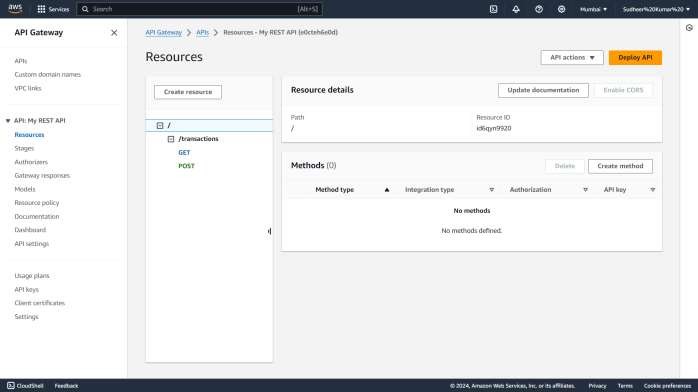


Here is my RDS database.



*Step 5:* Created two new Lambda functions. Configure the functions’ execution role to allow it to access the RDS instance. Attach the necessary IAM policies.

*Step 6:* Create a new **REST API** then create resources and methods to handle different operations and integrate these methods with the corresponding Lambda functions.



*Step 7:* Deploy your API in API Gateway then I have use tools like Postman or CURL to send HTTP requests to your API endpoints to ensure everything works correctly.

*Step 8:* Ensure your RDS instance is not publicly accessible now use parameterized queries or prepared statements to prevent SQL injection then implement proper authentication and authorization mechanisms in your API.