ASSIGNMENT – 4 PART A

GitLab Link:

https://git.cs.dal.ca/umatiya/csci5410-f23-b00899642/-/tree/A4

I followed steps below to solve the part A of the assignment:

The common steps I took is given below:

Step 1:

- Create an Account and login to the AWS Academy as shown in fig 1 & 2.

Fig 1: AWS Academy account login page

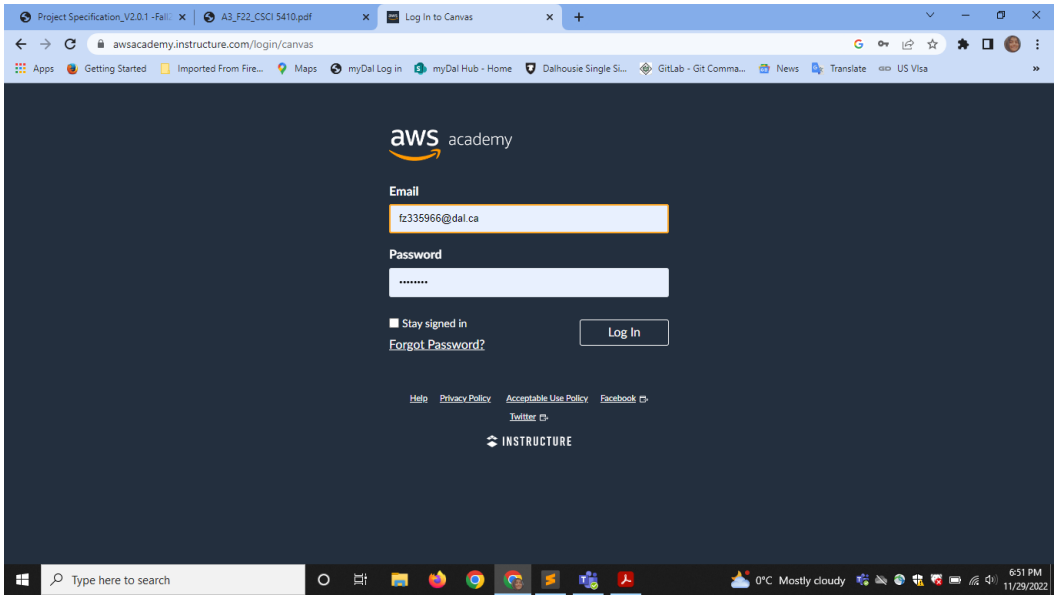


Fig 2: Student Account login page

- After login to the AWS academy, go to AWS Academy Learner Lab as shown in the fig 3.

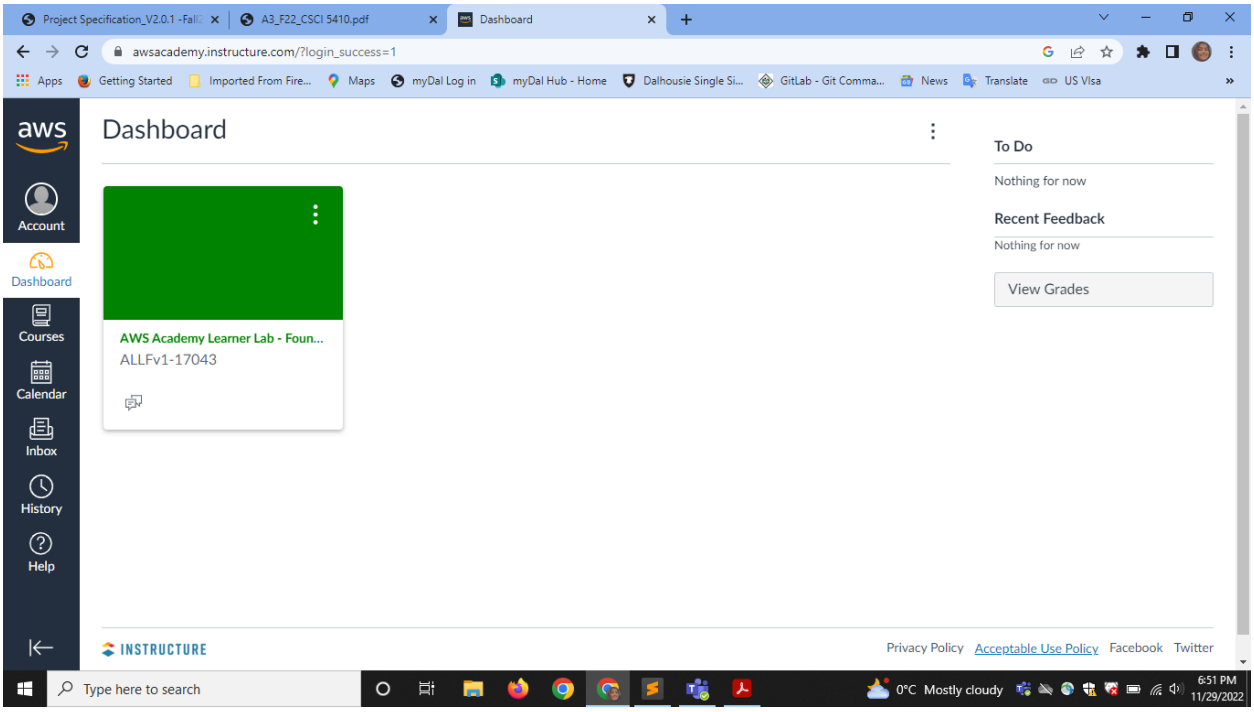


Fig 3: AWS Dashboard

- The first window you'll see open is the AWS Dashboard as shown in the fig 3.
- Click on the AWS Academy Learner Lab shown in the fig. 3 and the new window will be open which is shown in the fig 4.

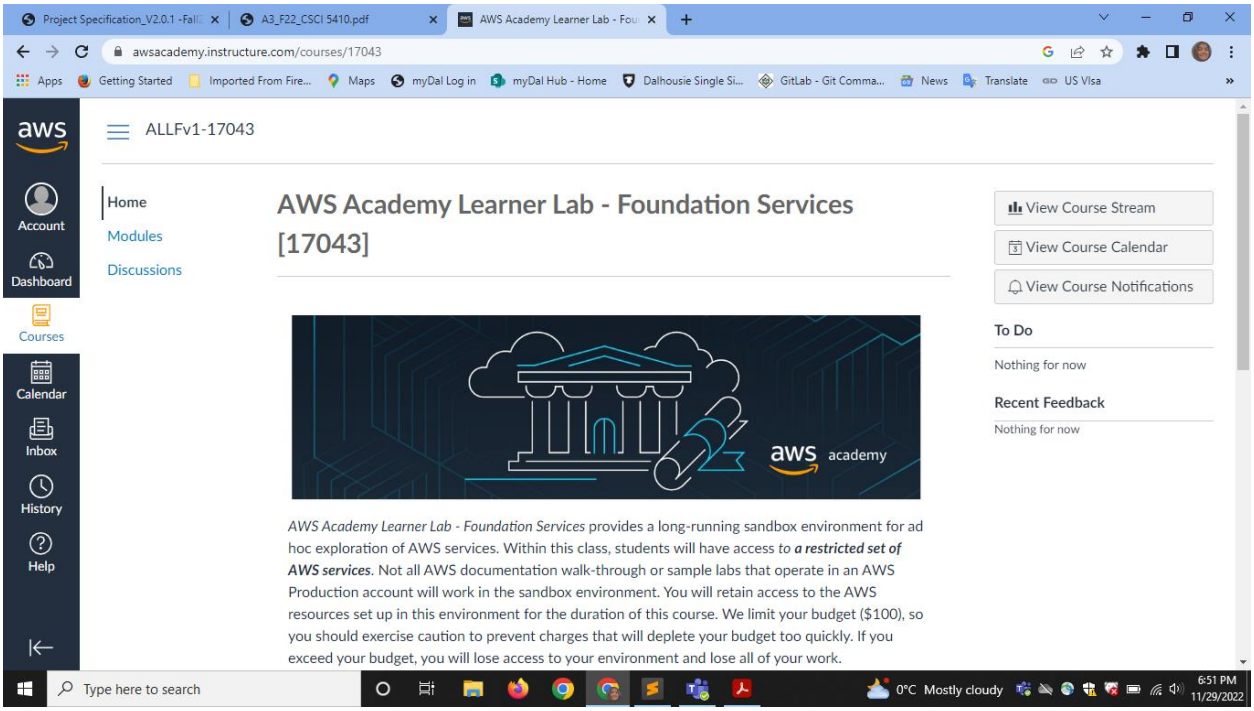


Fig 4: AWS home page

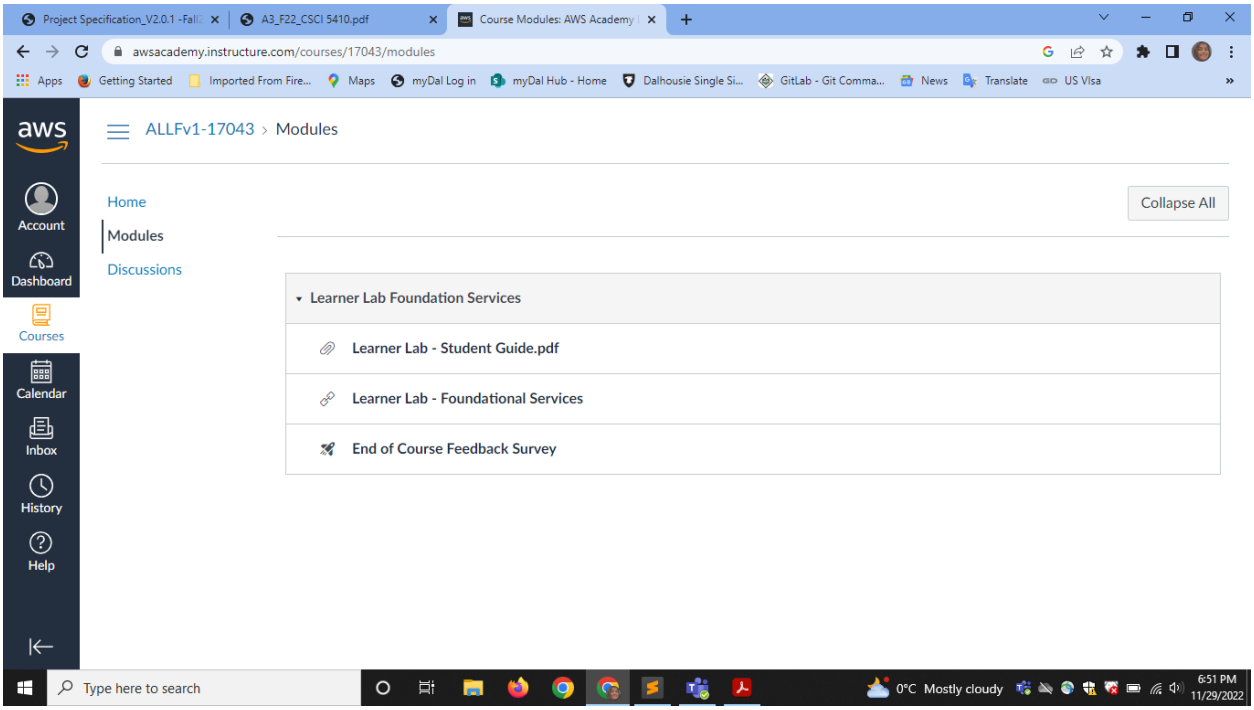


Fig 5: AWS Modules Page

- Go to modules as shown in the fig 4. After clicking on the modules, AWS modules page will open as shown in the fig 5.

- Go to “Learner Lab – Foundational Services” as shown in fig 5. After clicking on it, the start lab and end lab page will open as shown in the fig 6.
- Click on the Start Lab to start the lab (The red dot represents the lab is not started yet) as shown in the fig 6.

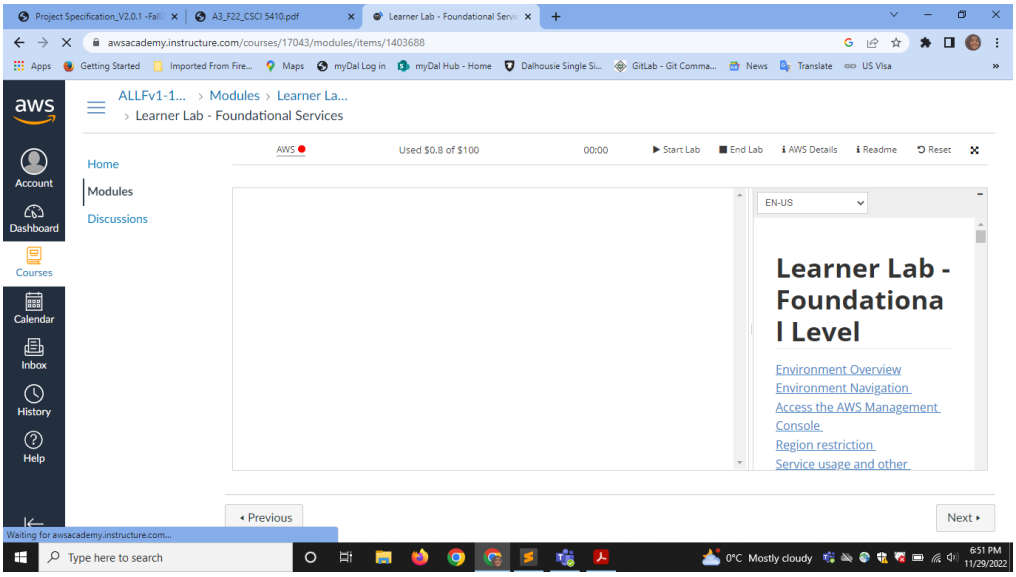


Fig 6: AWS End lab

- Lab has started because the green dot indicates the lab has started as shown in the fig 7.
- On clicking the AWS green icon which is shown in the fig. 7, it will redirect the user to the AWS Console Home as shown in the fig. 7.1. It contains all different services like Amazon Lex. DynamoDB, S3, Lambda and many more.

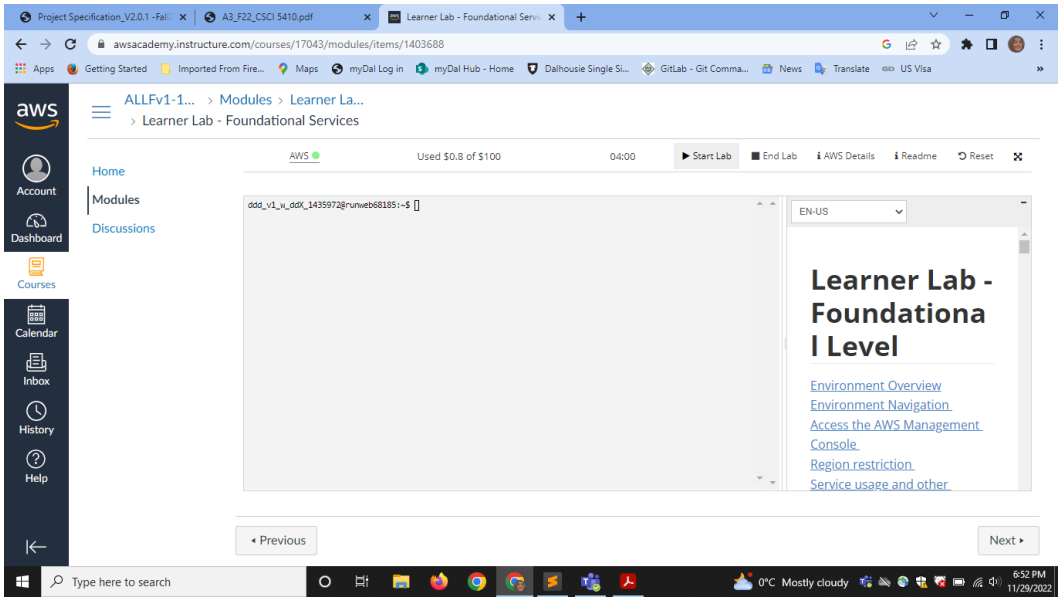


Fig 7: AWS start lab

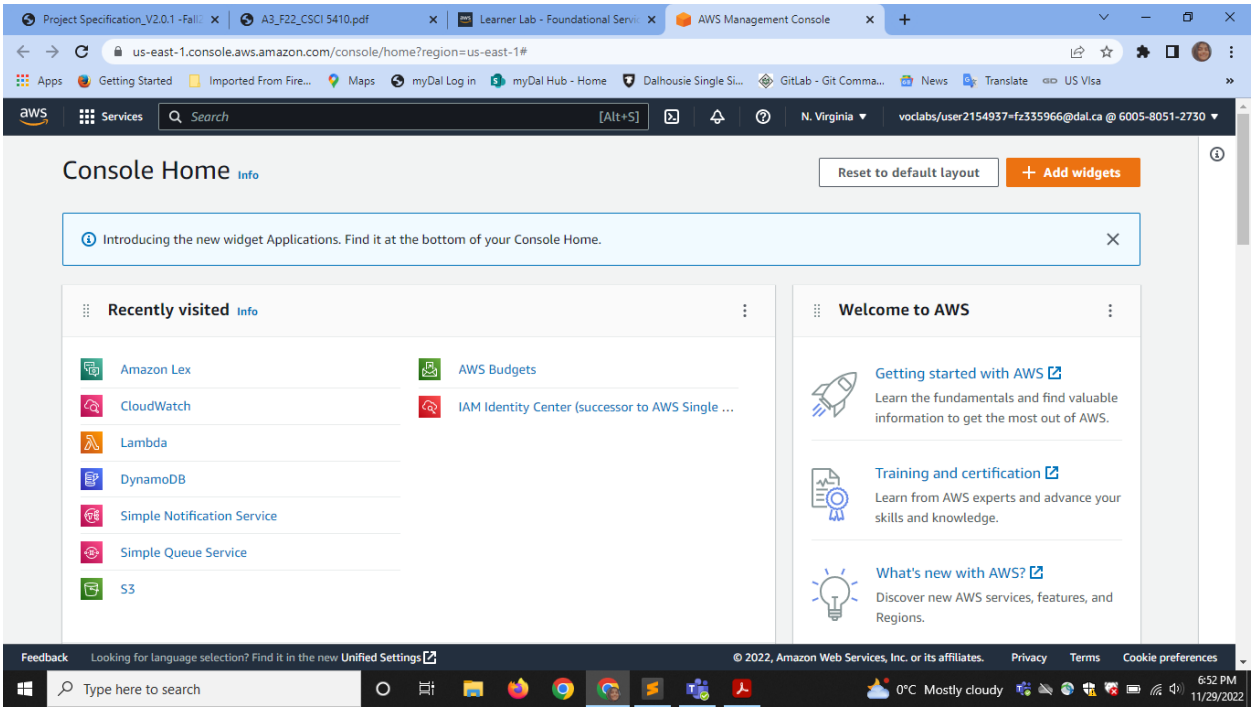


Fig 7.1: AWS Console Home

Step 2: Create SQS

- Go to AWS Console (follow step 1 to get here) and search for Amazon SQS as shown in the fig 8.
- Click on the “Create queue” to create queue as shown in the fig 9 or 10.
- Fill the details in the “Create queue” form as shown in the fig 11 – 17.
- The queue “PartA_SQSqueue” has been created and its details has been mentioned as shown in the fig 18.

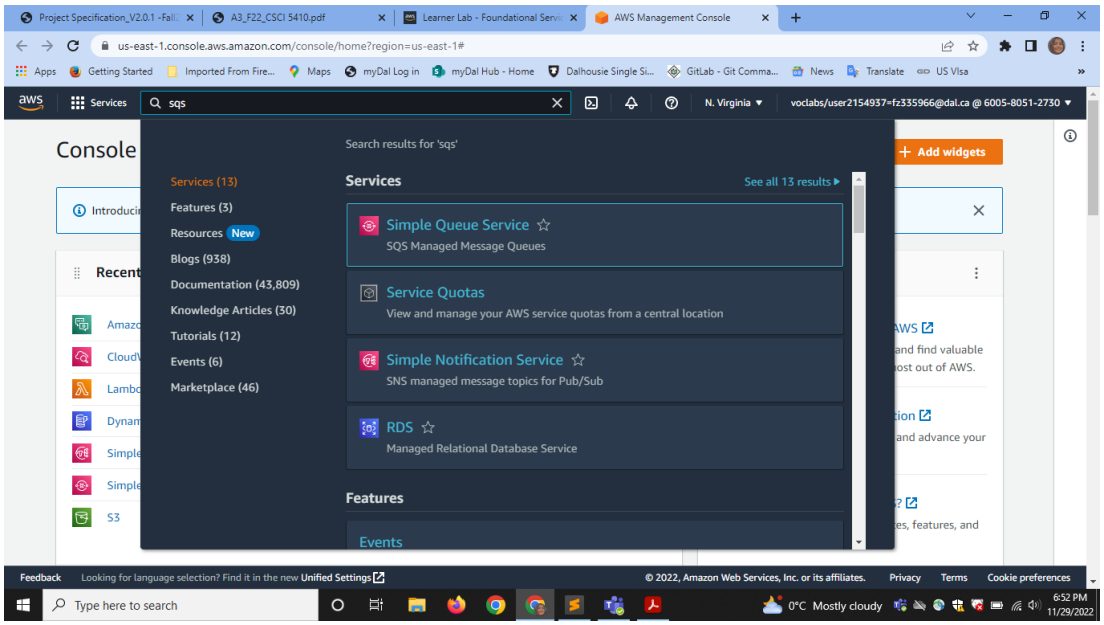


Fig 8: Search for Amazon SQS from search bar

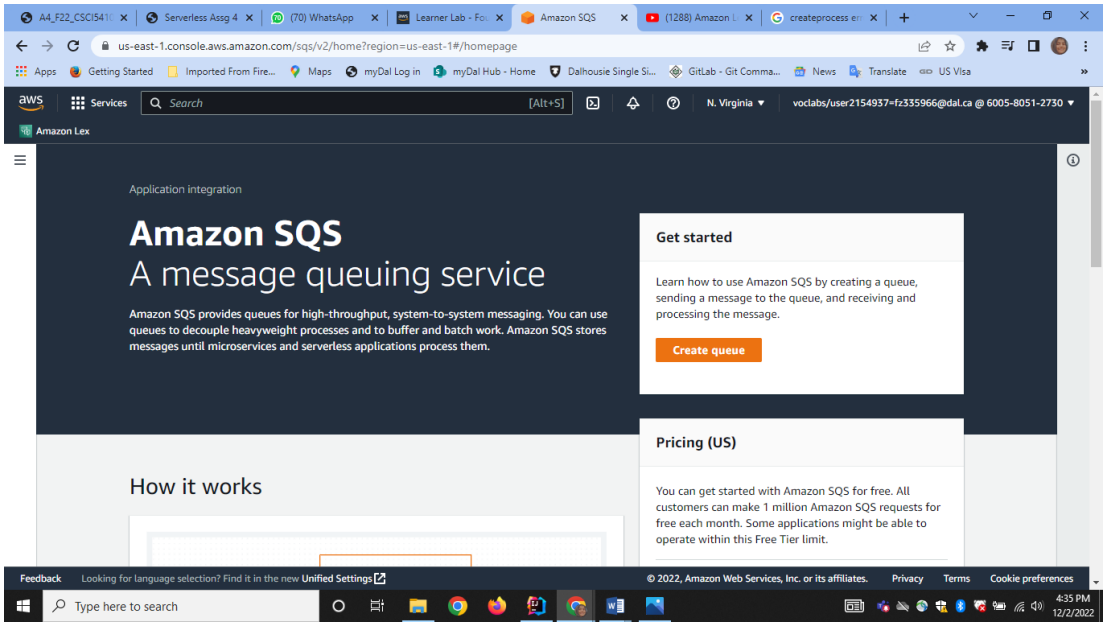


Fig 9: Amazon SQS home page

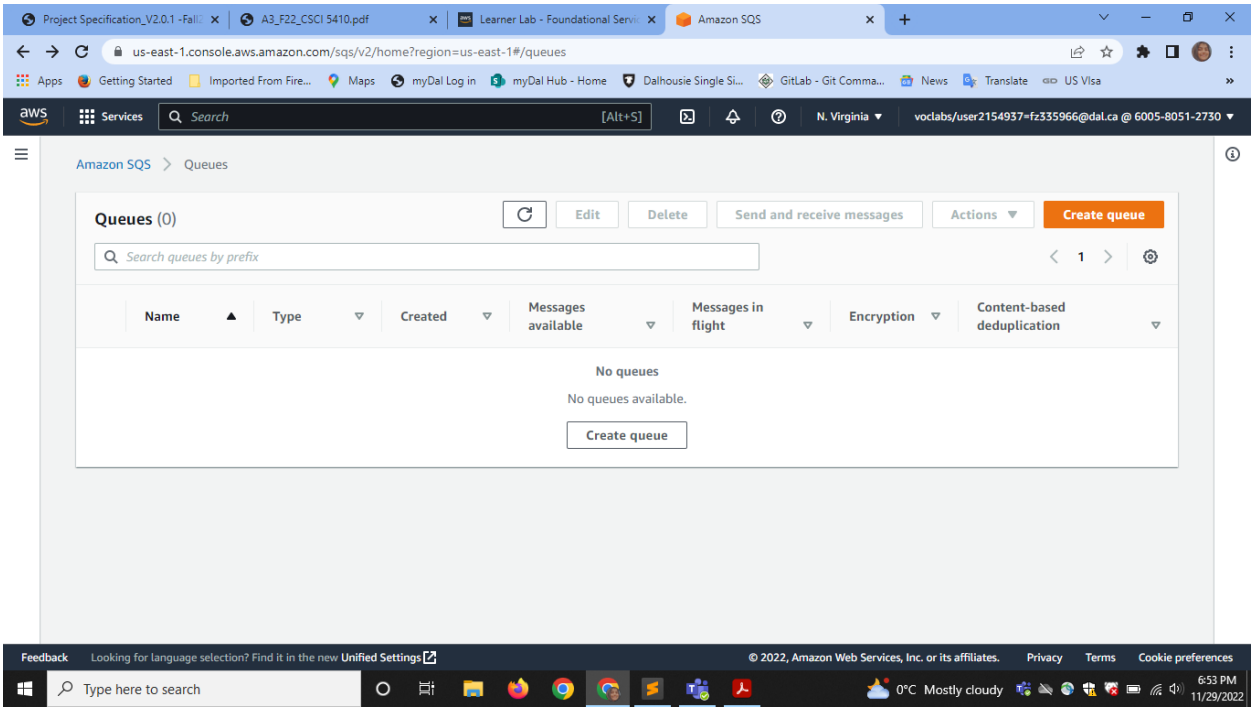


Fig 10: Queues page

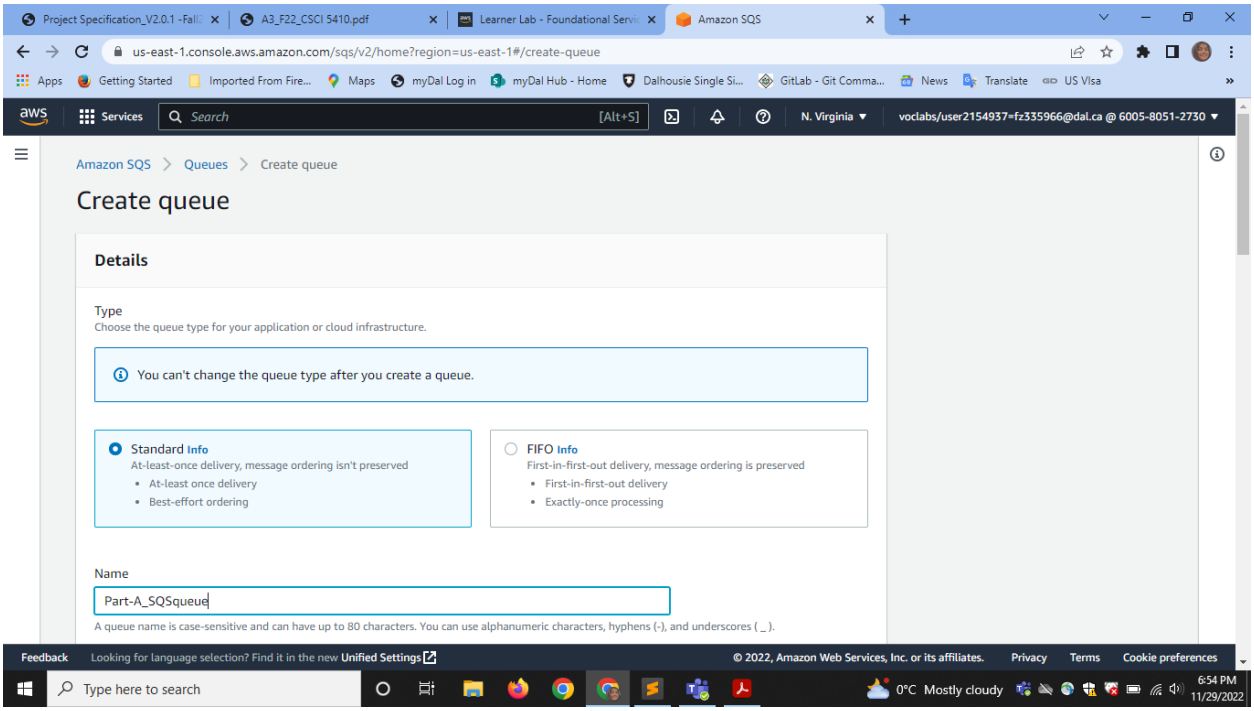


Fig 11: "Create Queue" details

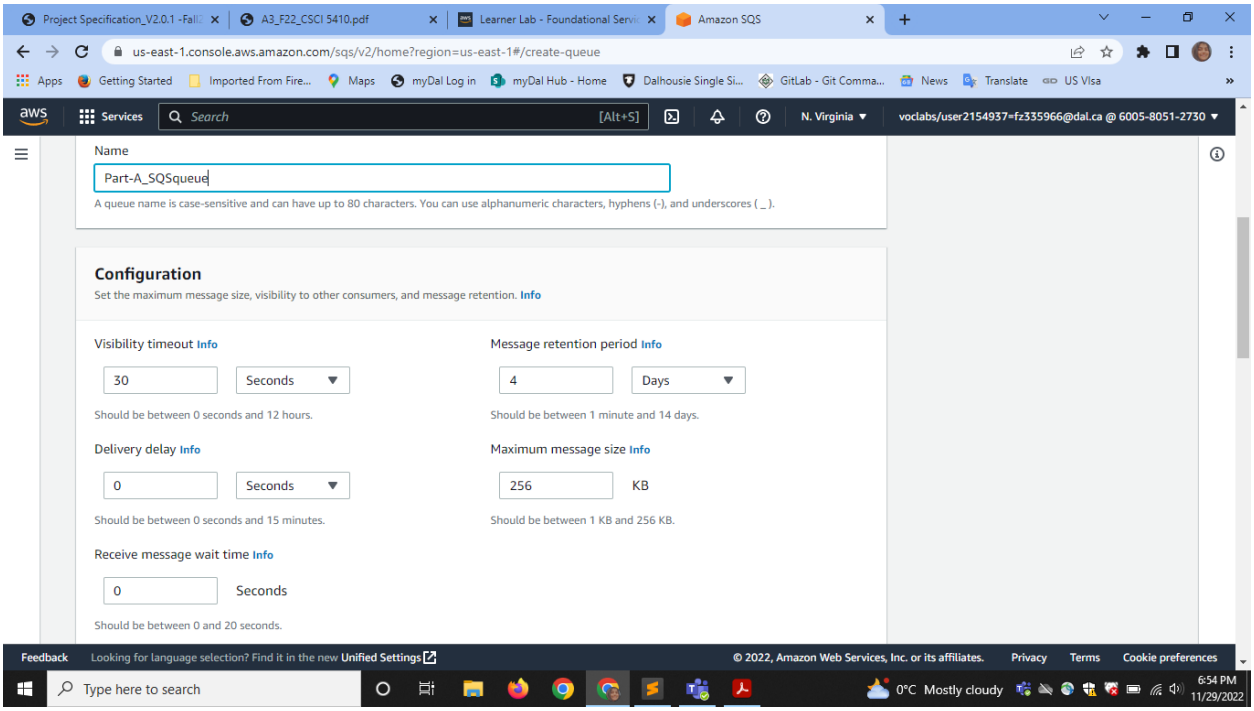


Fig 12: "Create Queue" details

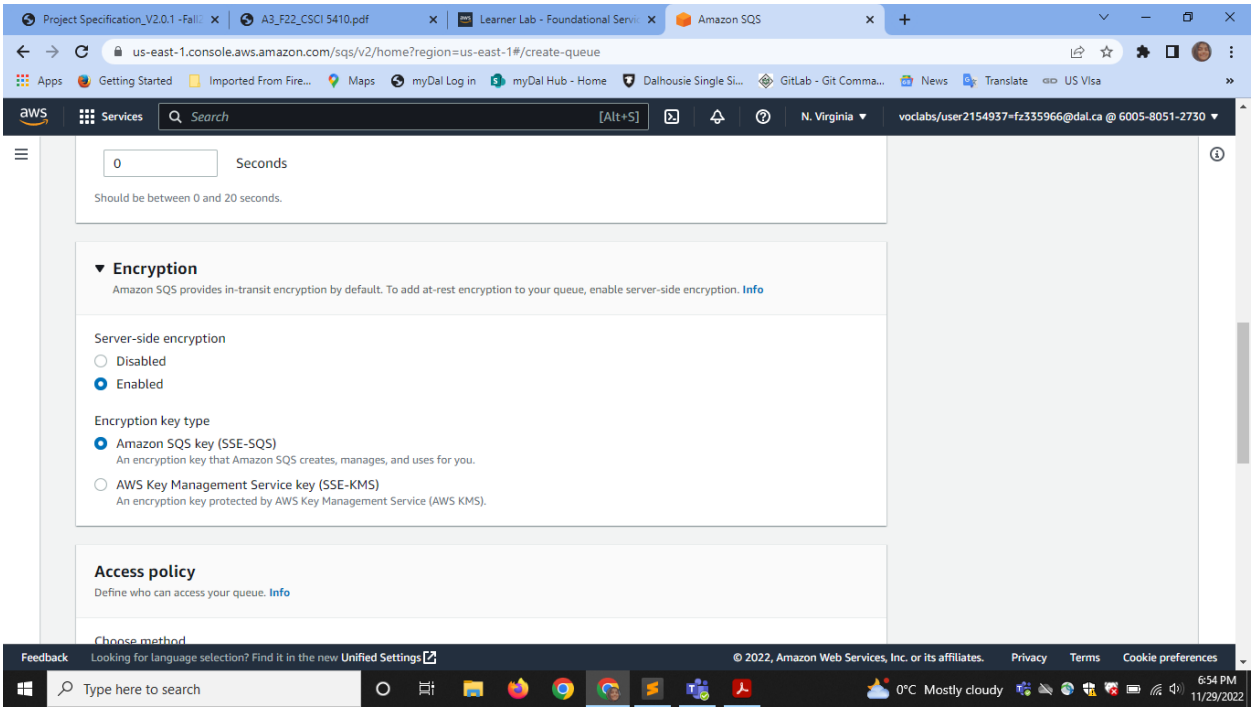


Fig 13: "Create Queue" details

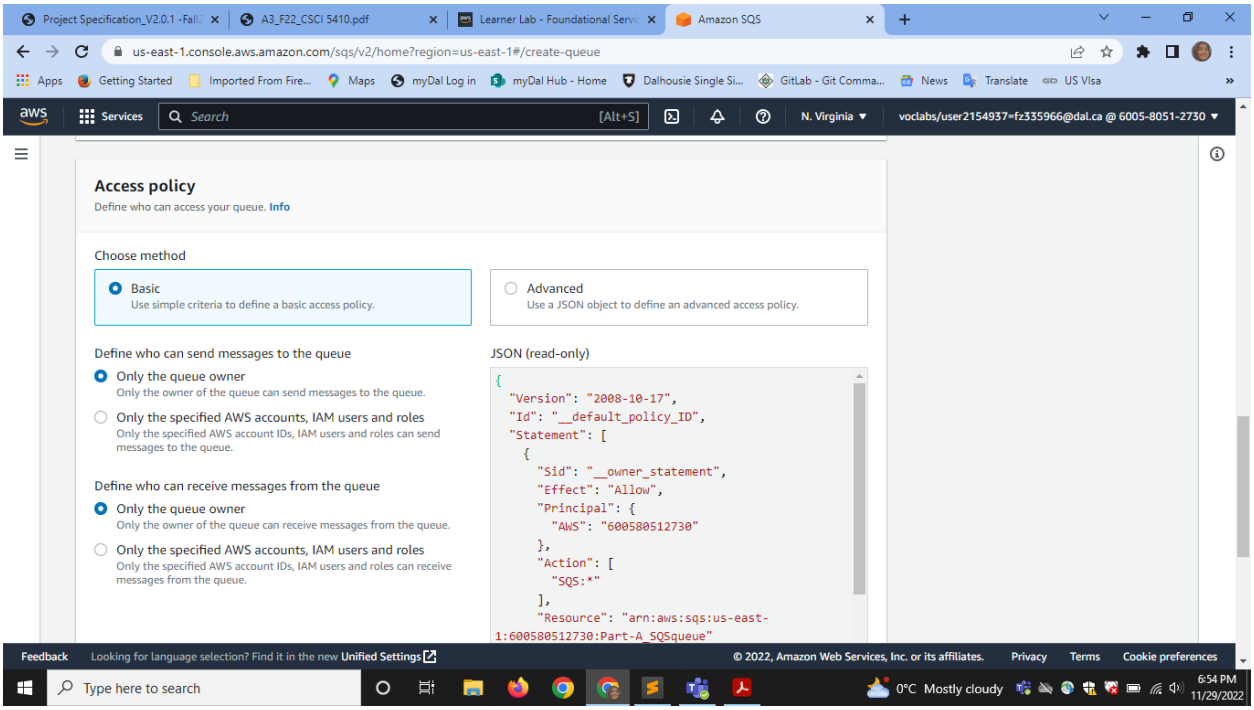


Fig 14: "Create Queue" details

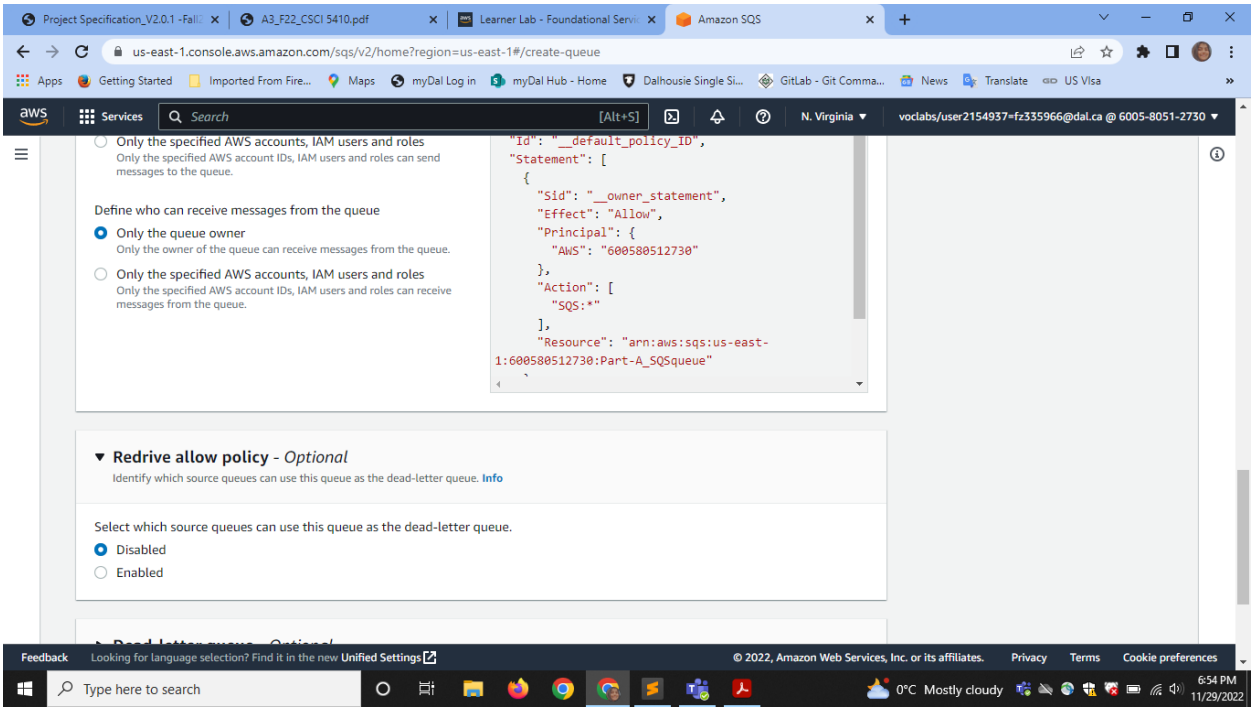


Fig 15: "Create Queue" details

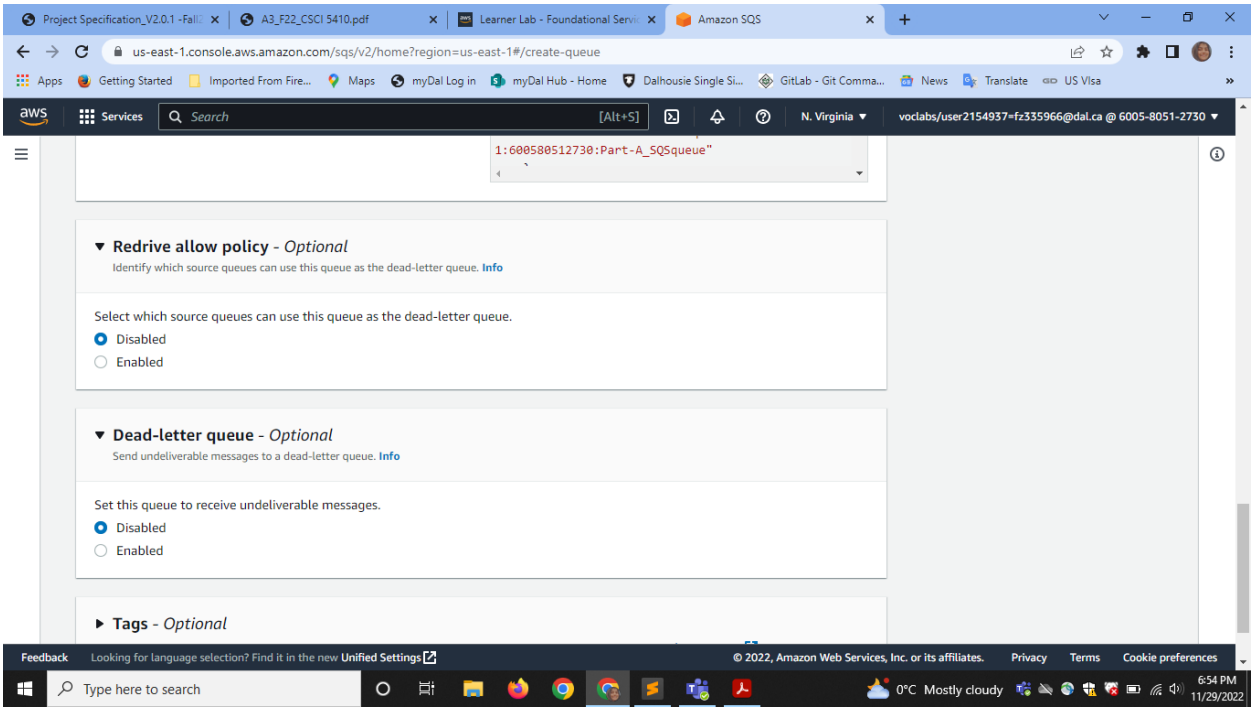


Fig 16: "Create Queue" details

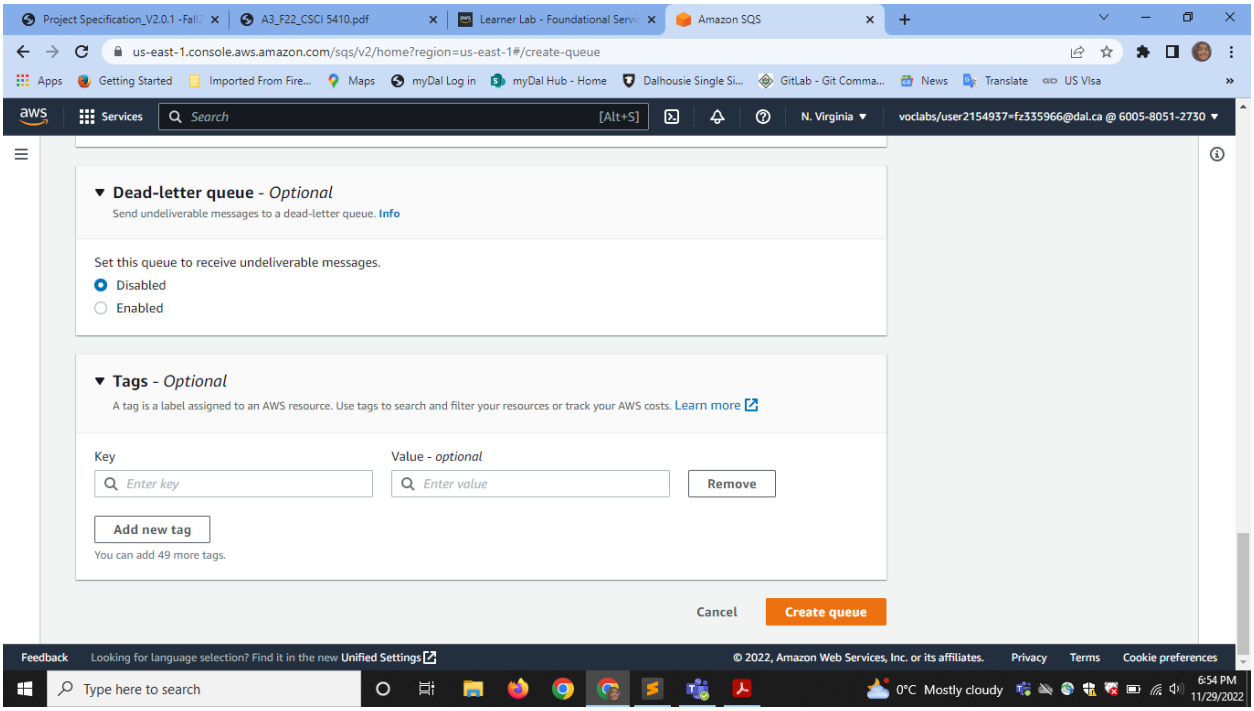


Fig 17: "Create Queue" details

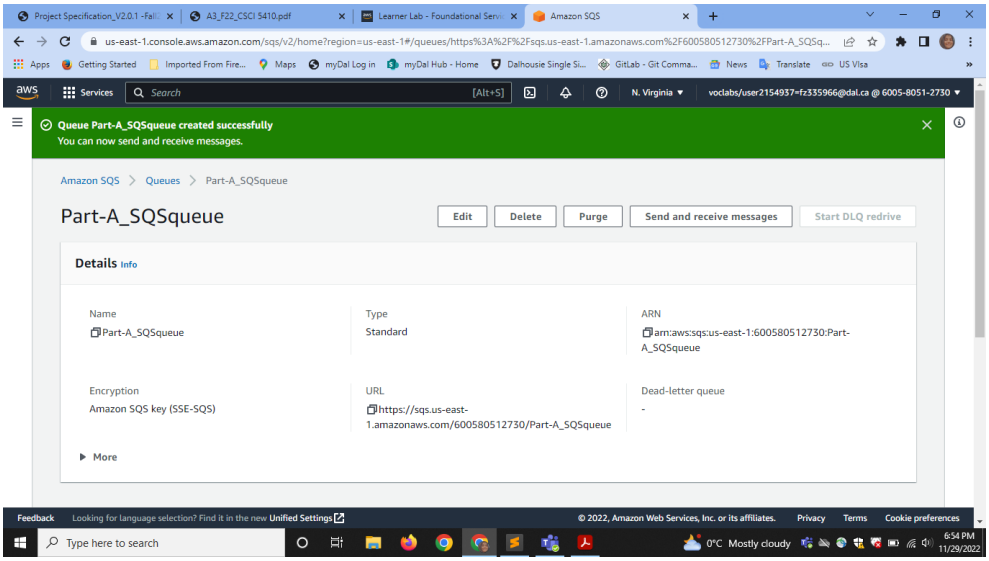


Fig 18: Queue has been created

Step 3: Create SNS

- Go to AWS Console (follow step 1 to get here) and search for Amazon SNS as shown in the fig 19.
- Click on the "Create topic" to create topic as shown in the fig 20.
- Fig 21 represents the dashboard for the SNS.
- Fill the details in the "Create queue" form as shown in the fig 22 – 32.

- The queue “Part-A_SNS” has been created and its details has been mentioned as shown in the fig 33.

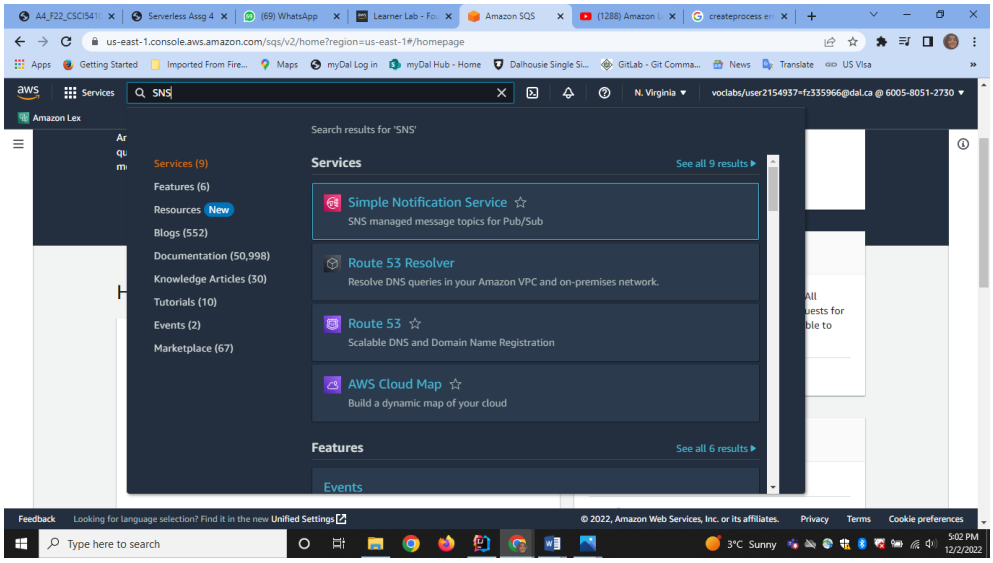


Fig 19: Search for Amazon SNS from search bar

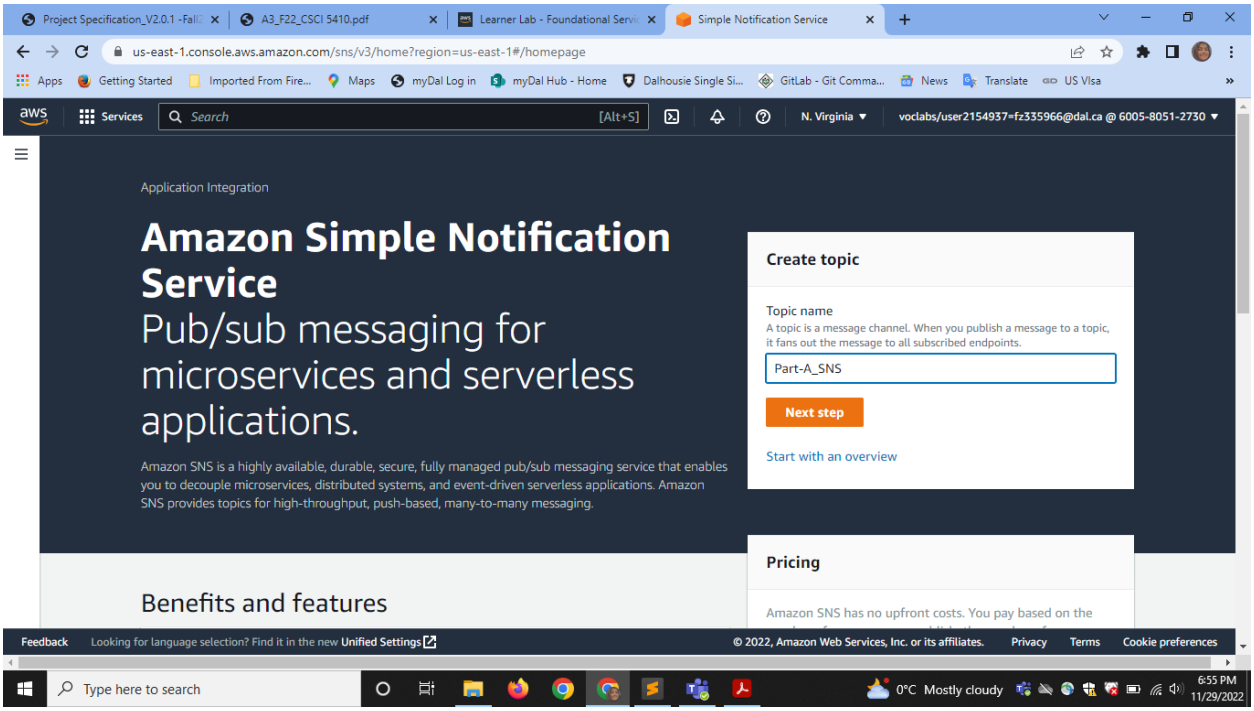


Fig 20: Amazon SNS home page

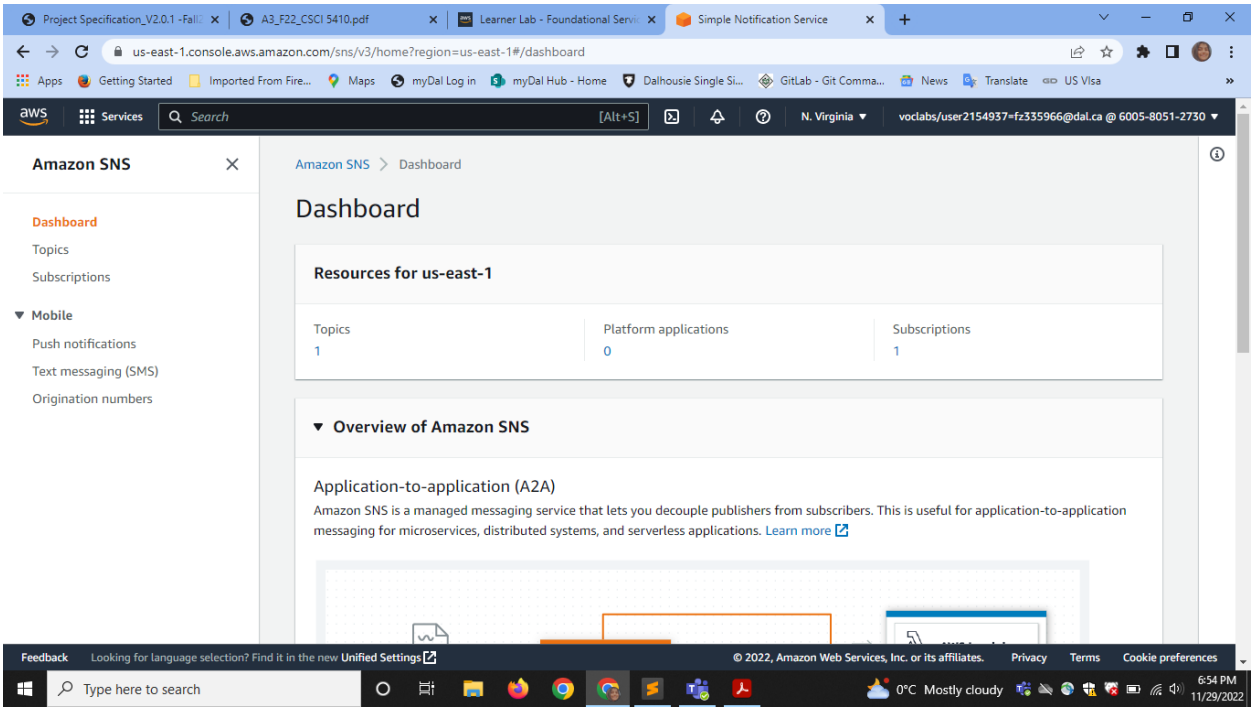


Fig 21: SNS Dashboard

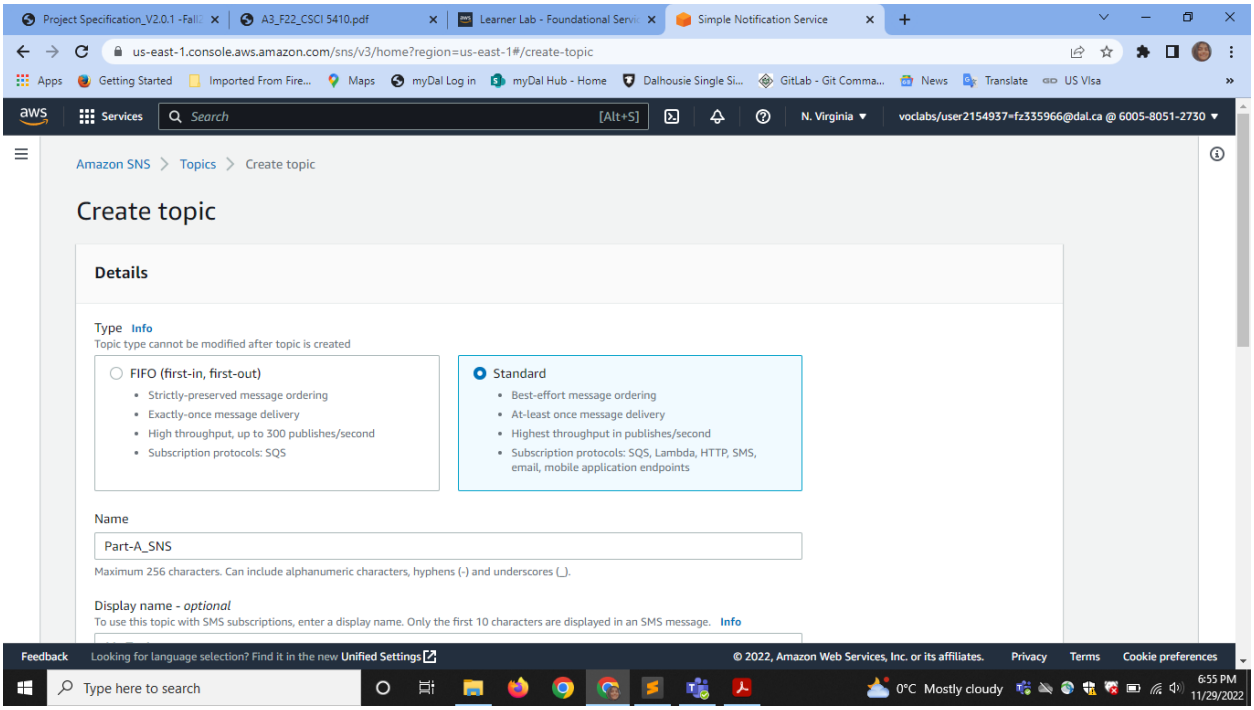


Fig 22: "Create topic" for SNS page

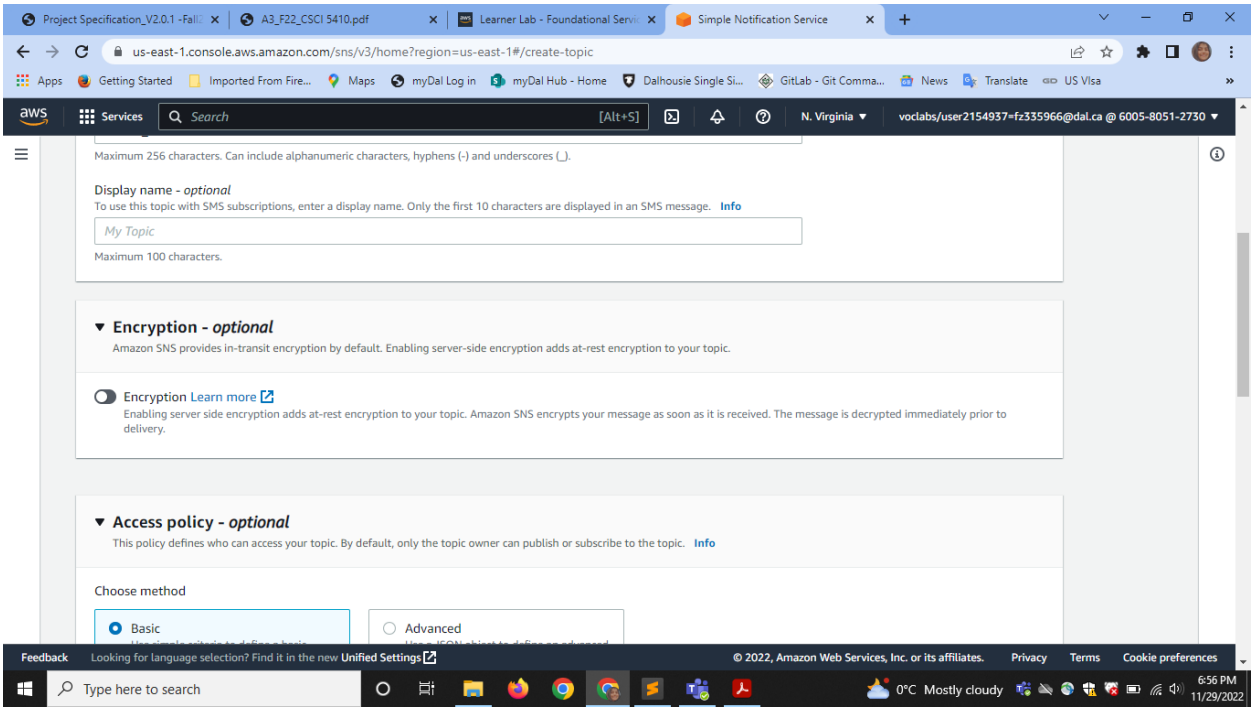


Fig 23: "Create topic" form

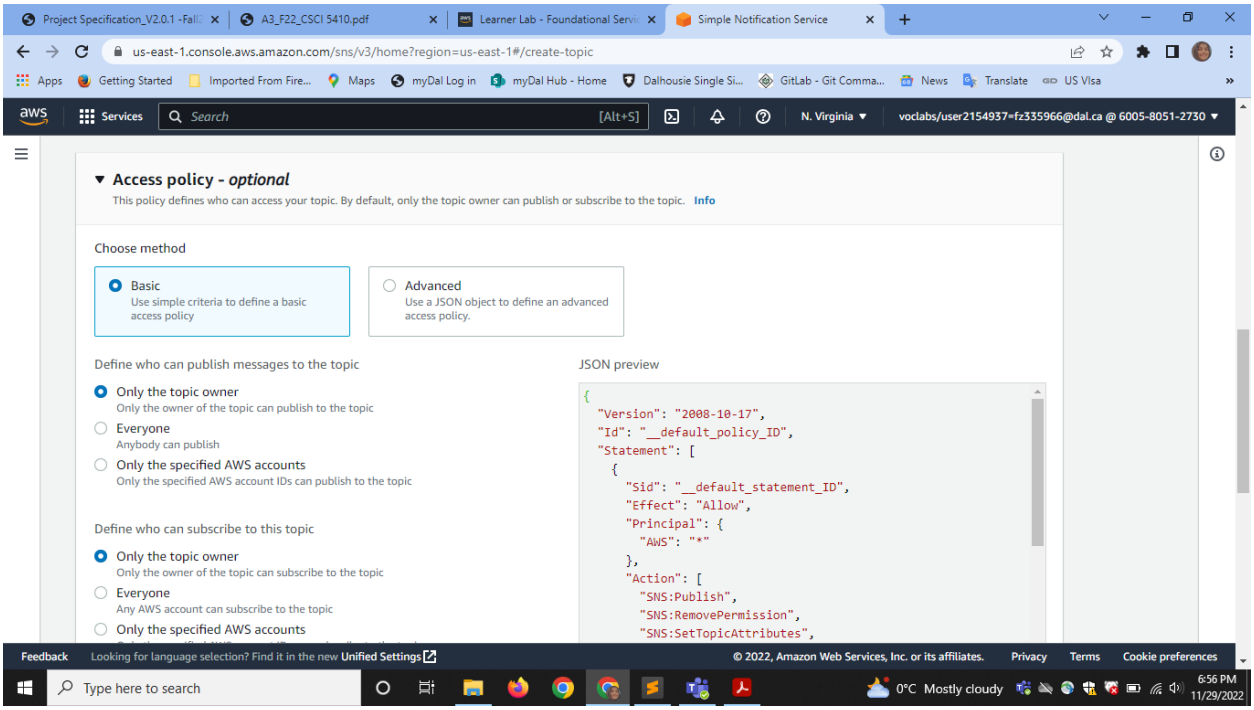


Fig 24: "Create topic" form

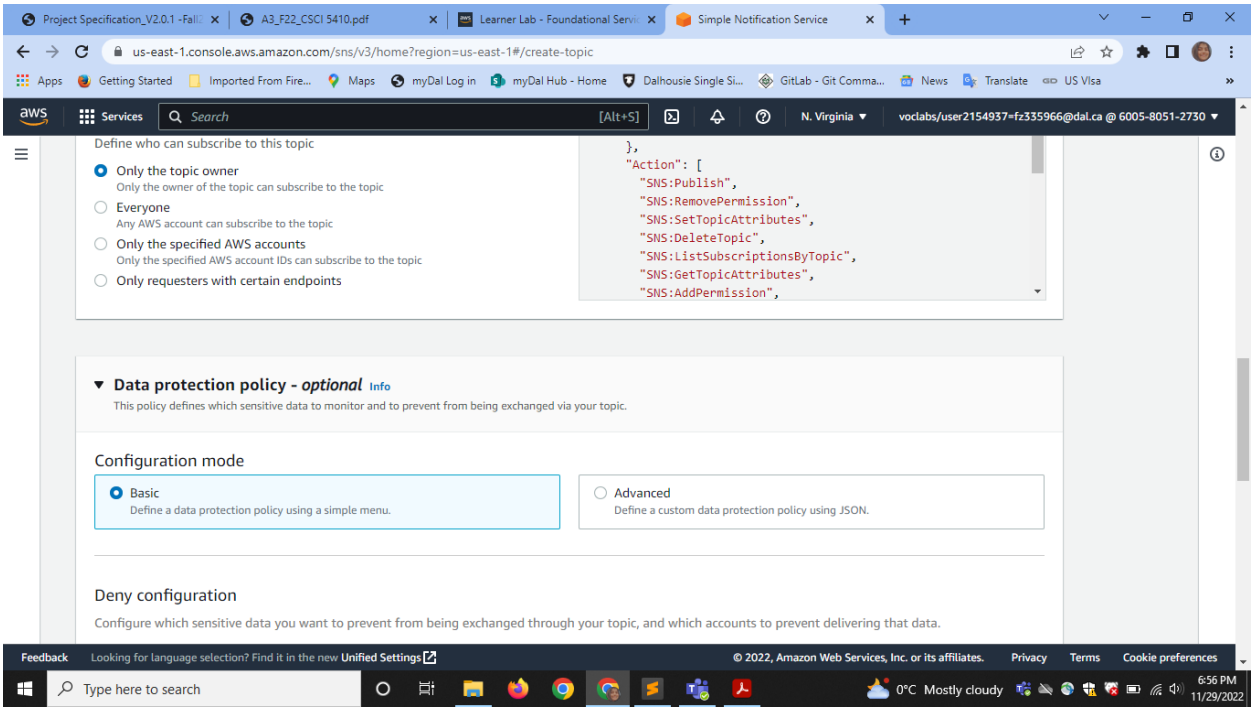


Fig 25: "Create topic" form

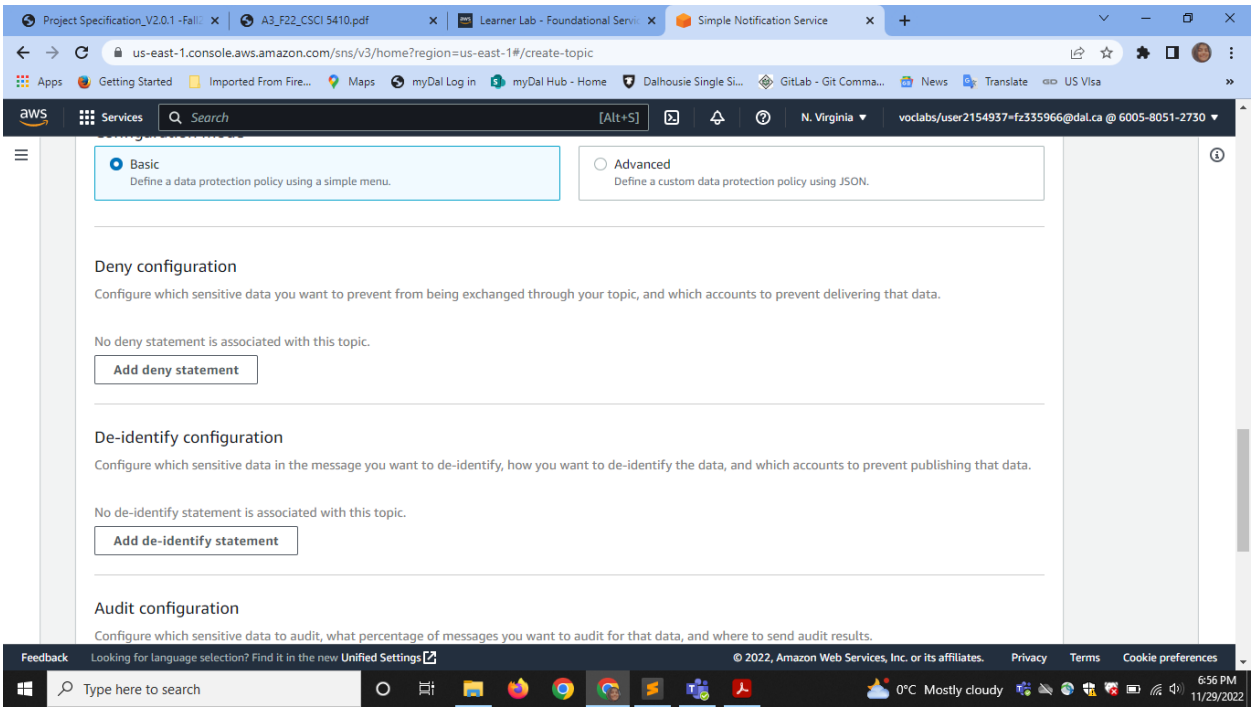


Fig 26: "Create topic" form

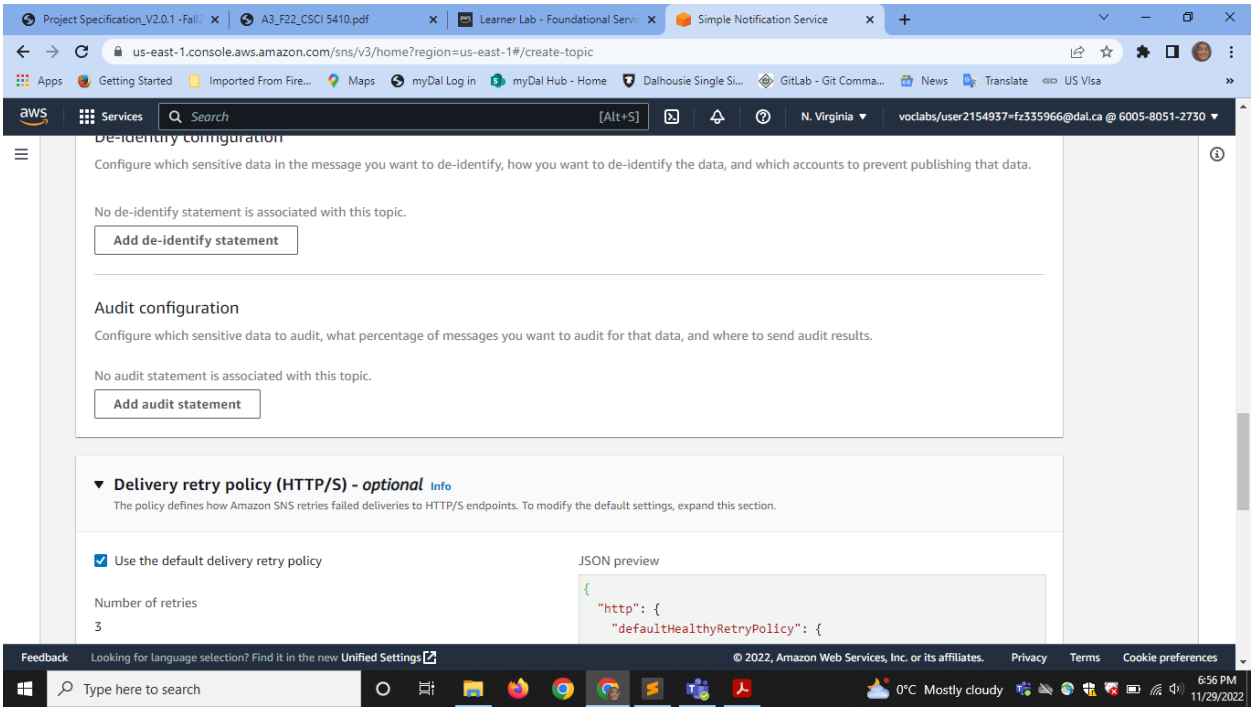


Fig 27: "Create topic" form

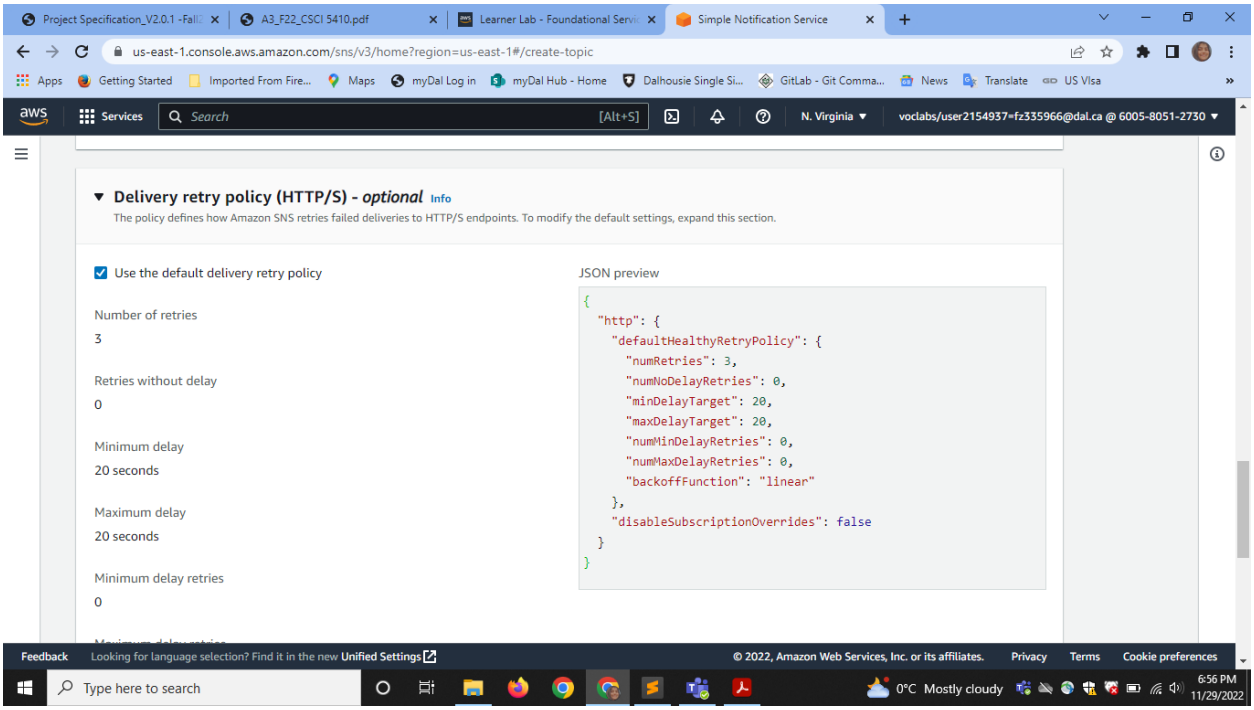


Fig 28: "Create topic" form

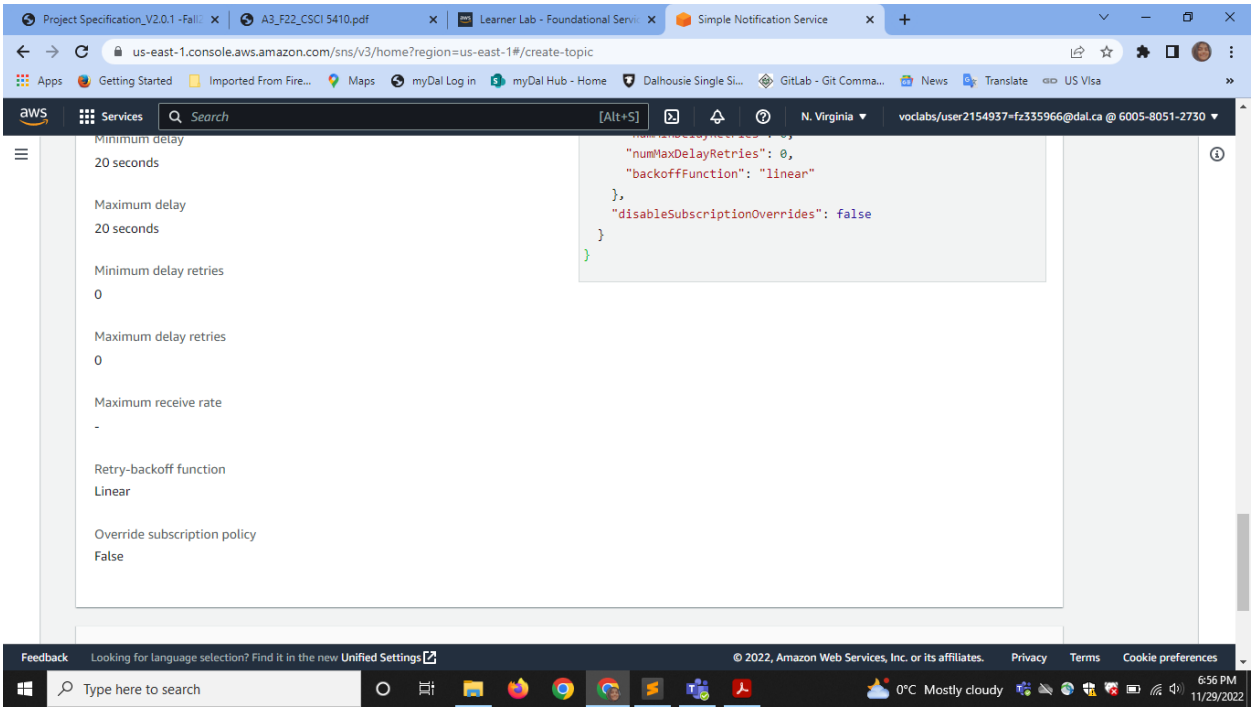


Fig 29: "Create topic" form

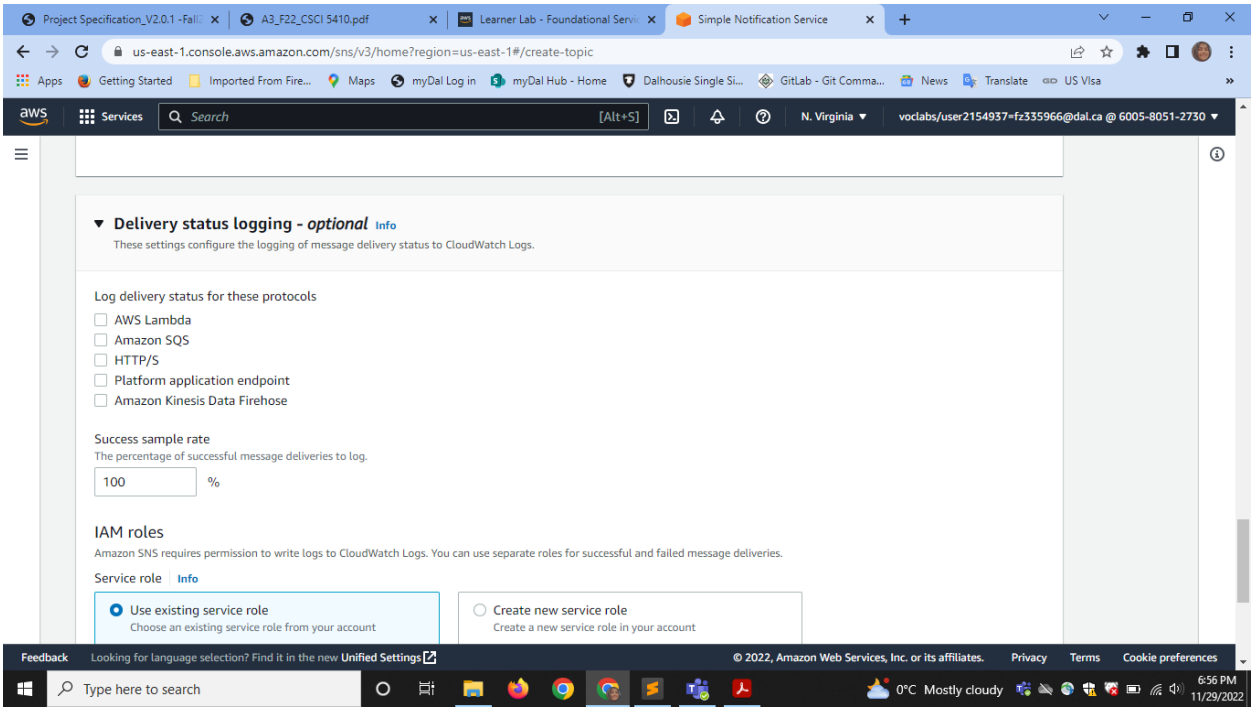


Fig 30: "Create topic" form

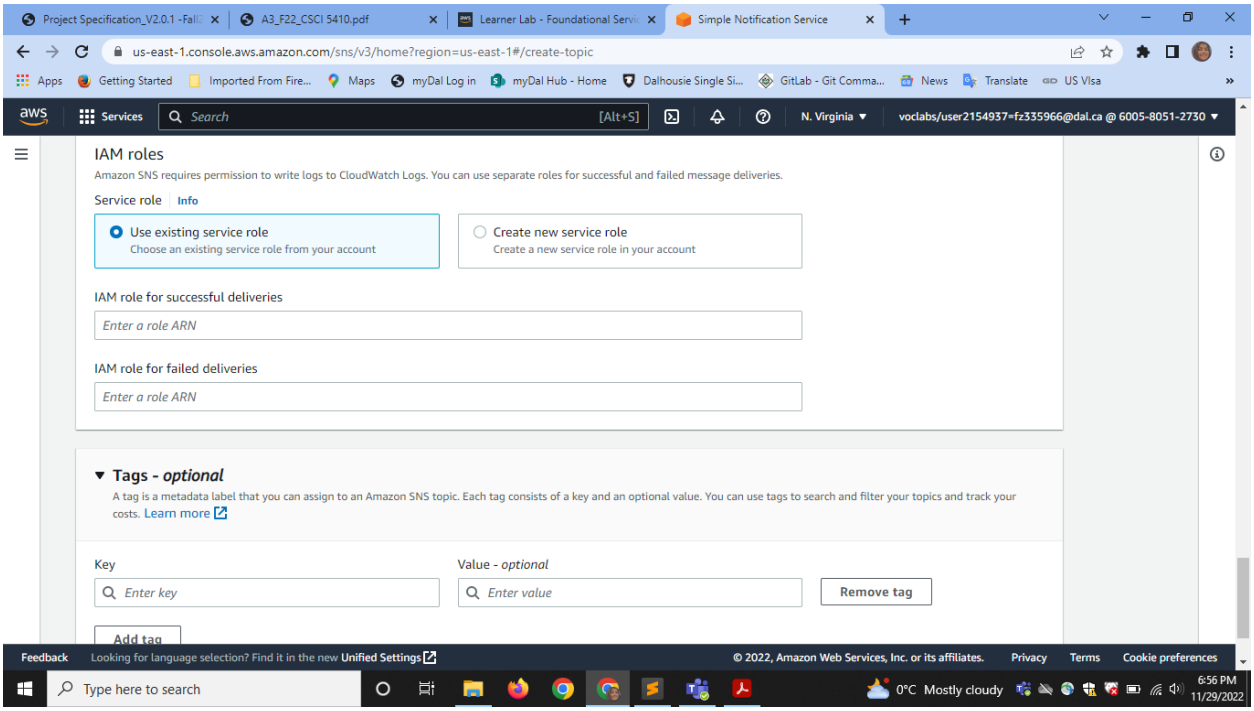


Fig 31: "Create topic" form

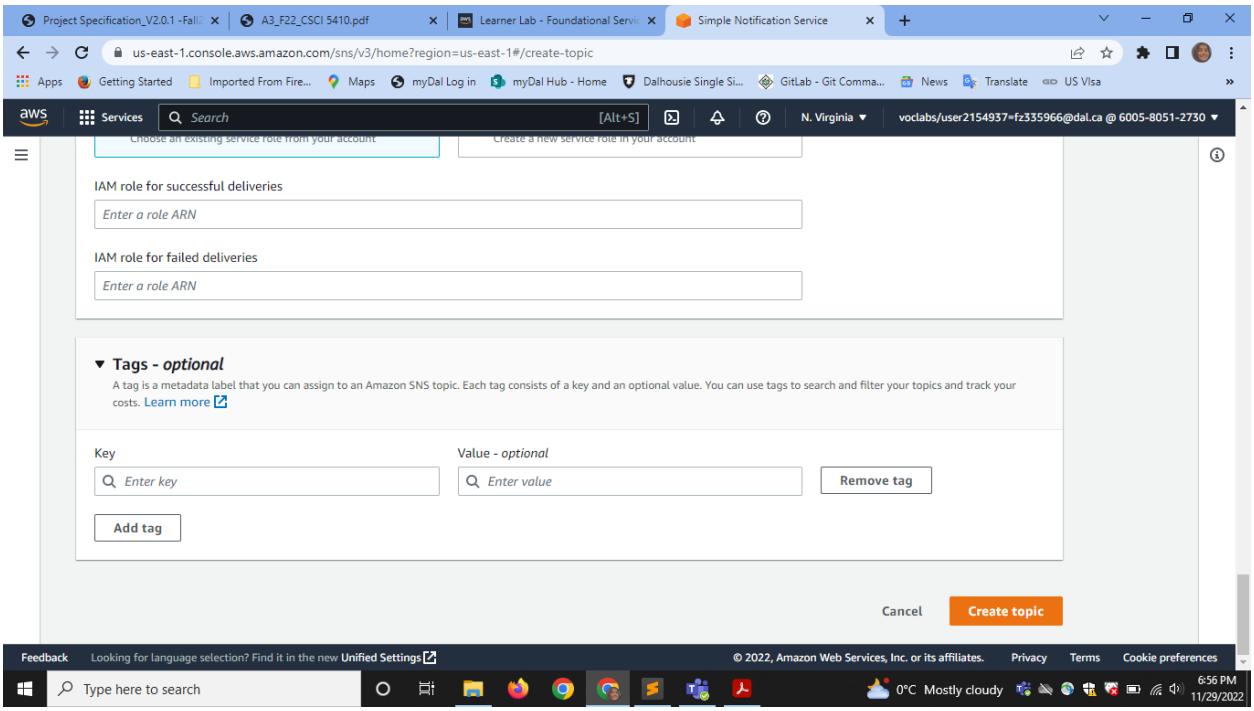


Fig 32: "Create topic" form

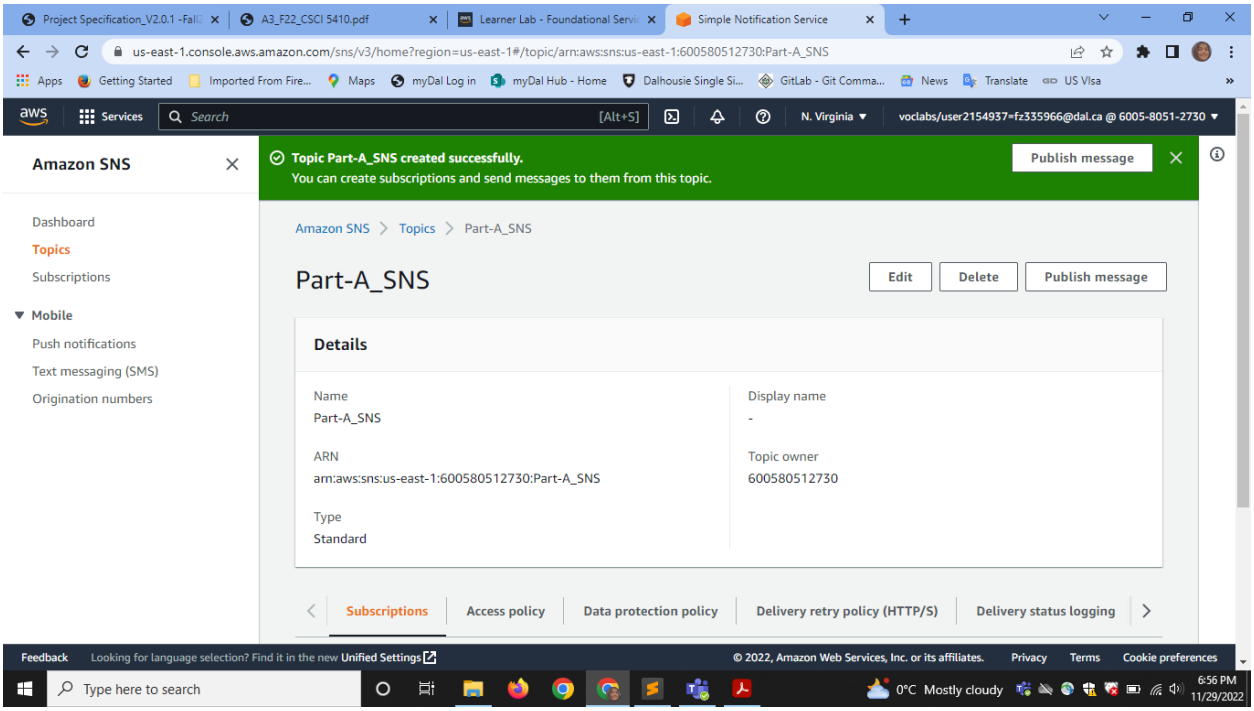


Fig 33: Topics "Part-A_SNS" has been created

Step 4: Create subscription

- Click on the "Create subscription" button as shown on the page as shown in the fig 34.
- Fill the details in the "Create subscription" form as shown in the fig 35-37.
- The subscription "Part-A_SNS" has been created and its details has been mentioned as shown in the fig 38 and 39.
- The subscription is created to Part-A_SNS as shown in the fig 38. The status for the subscription is currently showing "pending confirmation" as shown in the fig 39.
- Now check the email which you provided while creating subscription (Refer fig 36 and 40).
- Confirm the subscription you received on email by AWS Notifications as shown in the fig 40.
- Go to Subscription page AWS to see the status changes from pending to confirmed on confirming the subscription through email notifications from AWS (Refer fig 42).

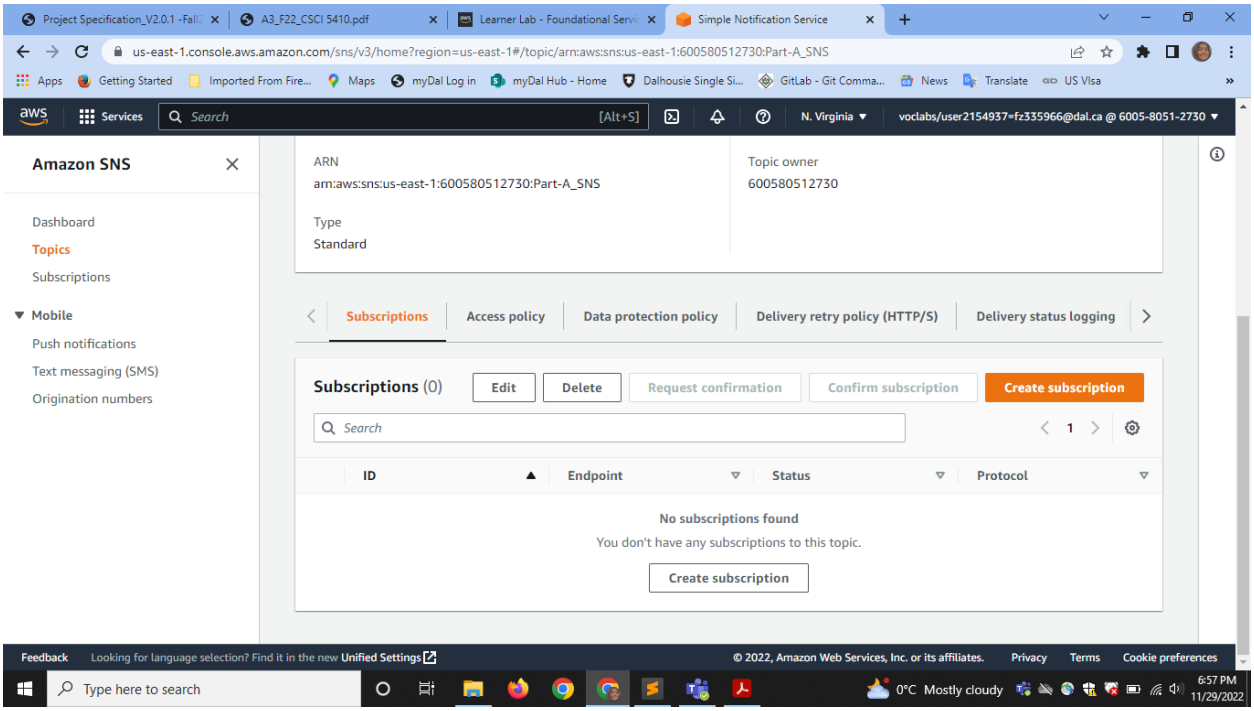


Fig 34: "Create subscription" page

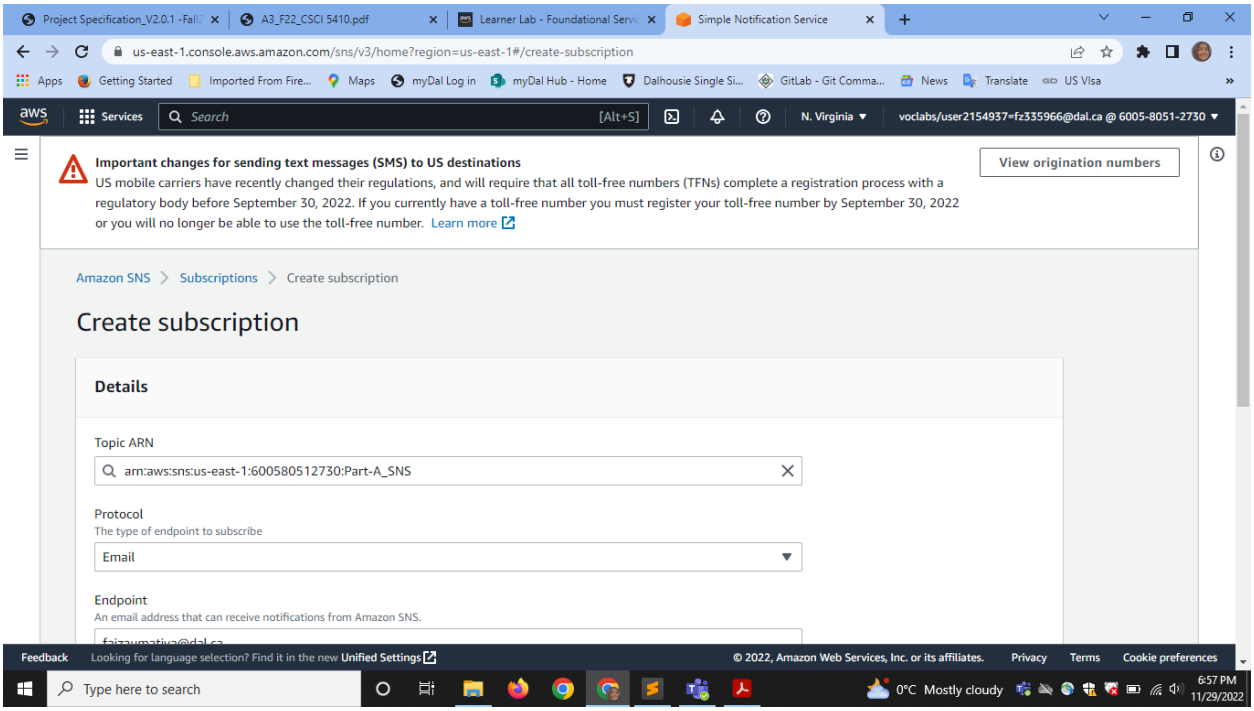


Fig 35: "Create subscription" form

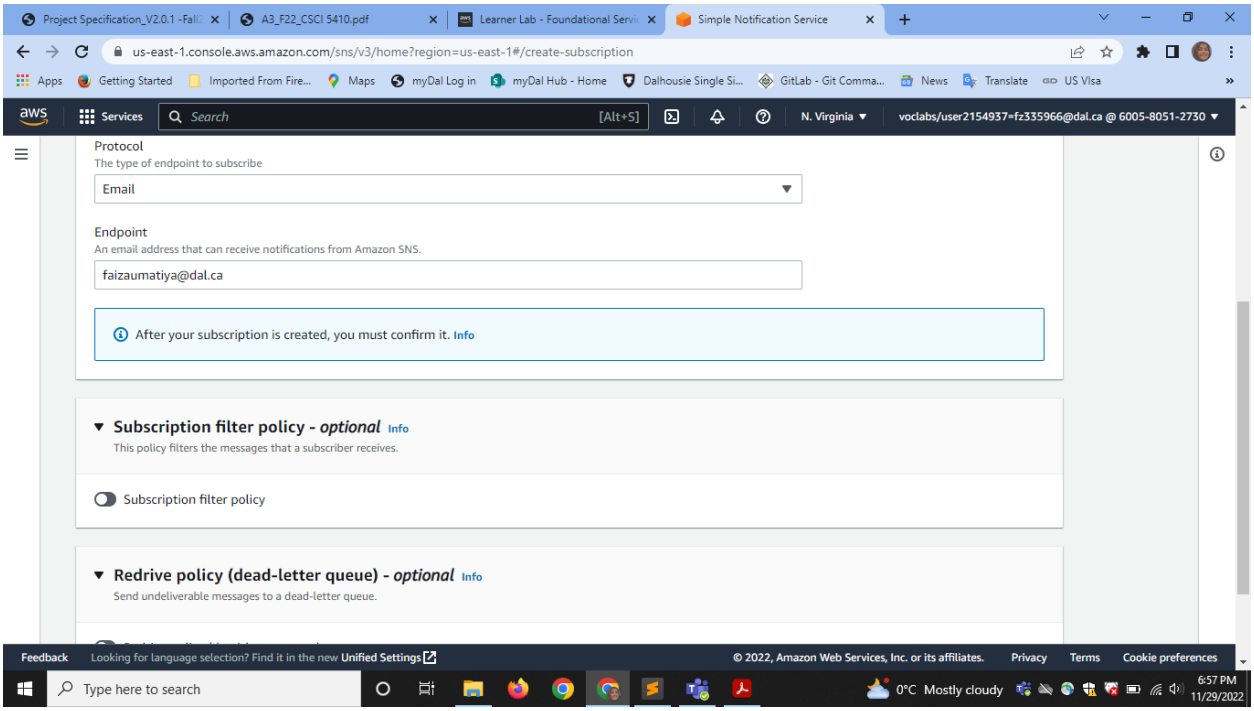


Fig 36: "Create subscription" form

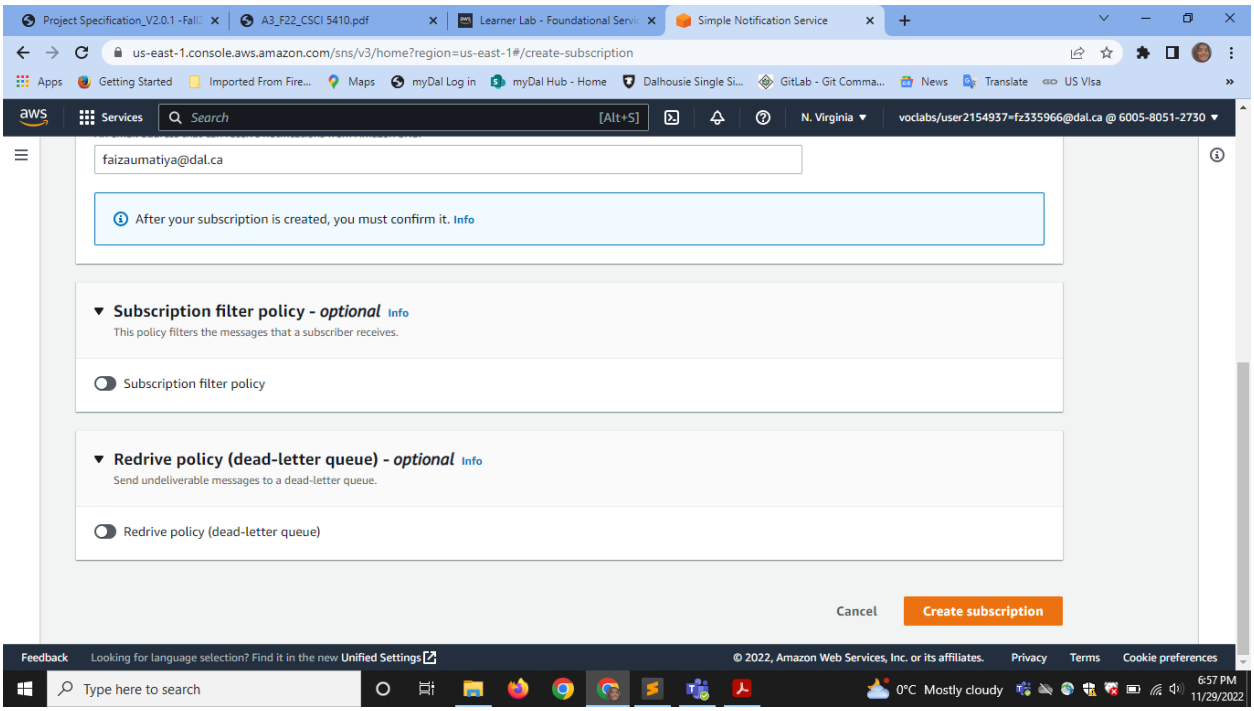


Fig 37: "Create subscription" form

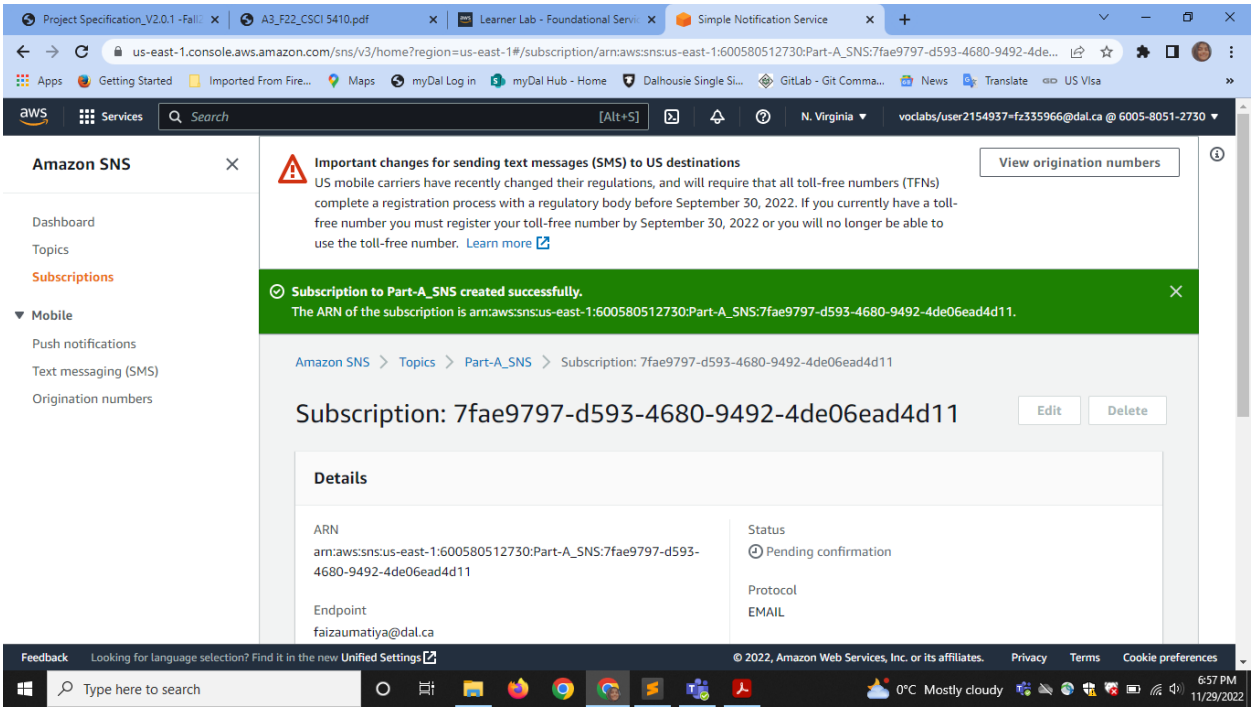


Fig 38: Subscription created successfully page

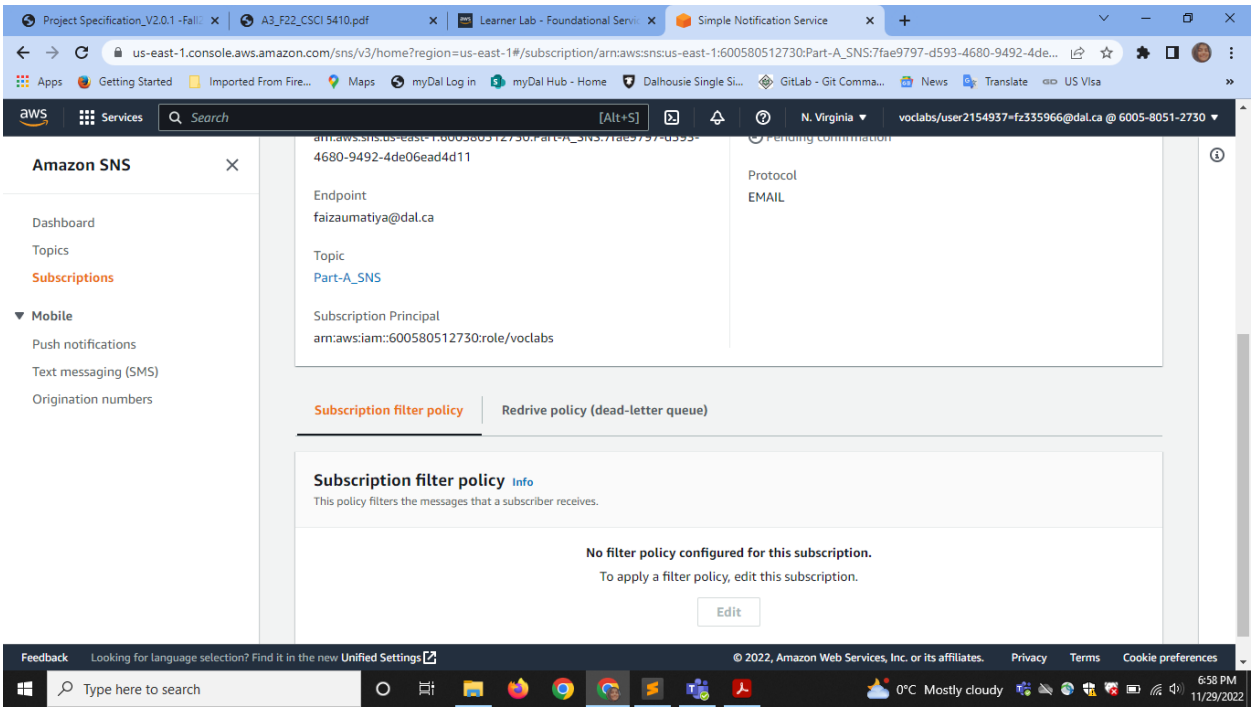


Fig 39: Subscription created successfully page

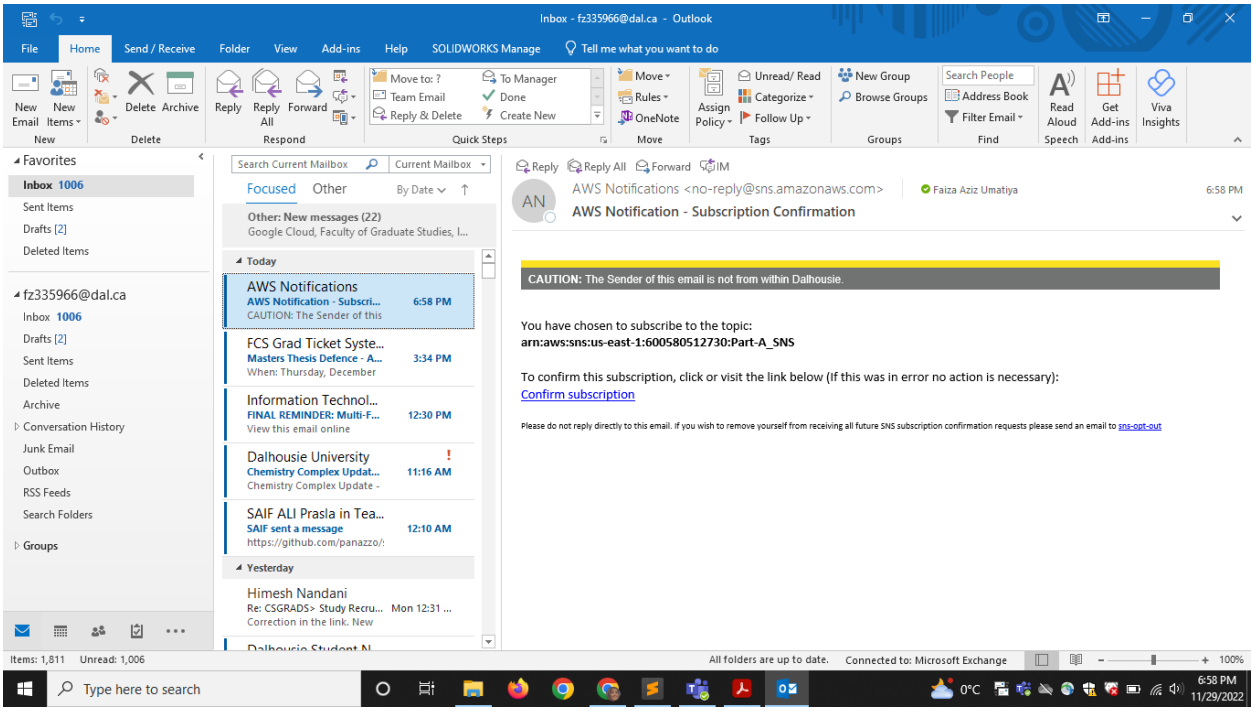


Fig 40: AWS notifications email for confirmation page

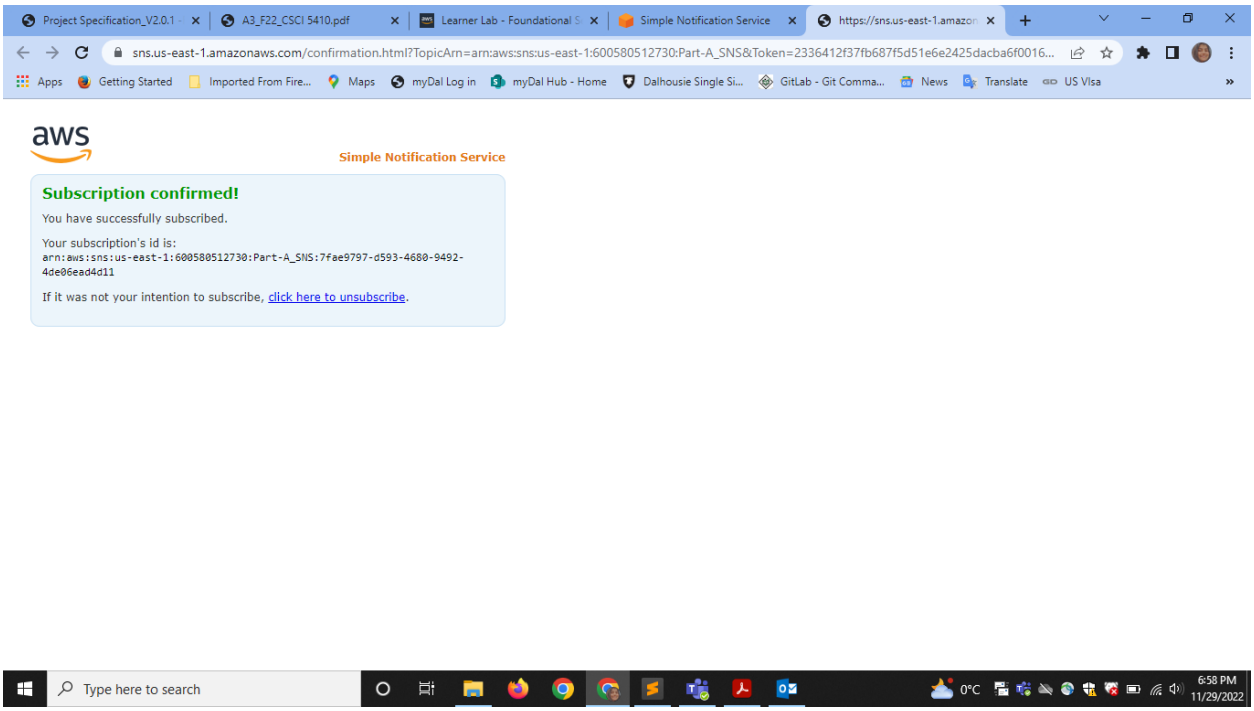


Fig 41: AWS subscription confirmation page

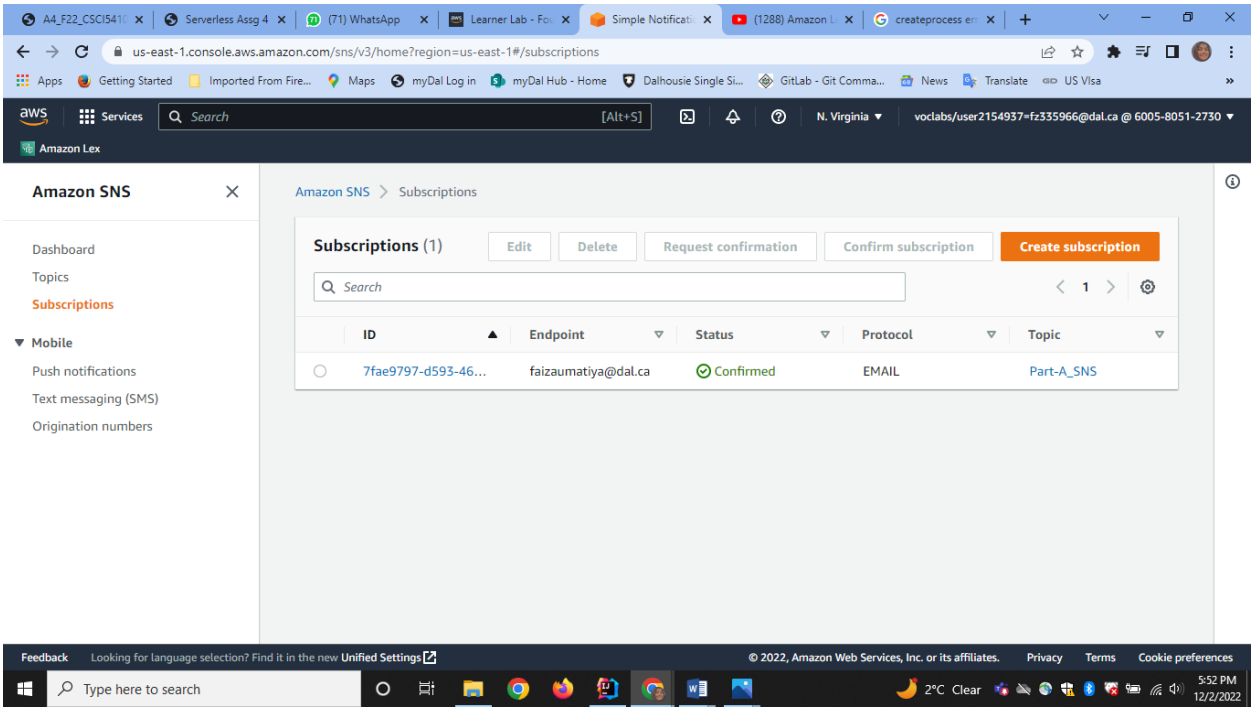


Fig 42: AWS subscription status confirmed

Step 5: Create a lambda function

- Go to search bar of the AWS console (Refer step 1 to get here) and search for AWS Lambda as shown in the fig 43.
- Fill the details to create lambda function as shown in the fig 44-47.
- Fig 48 shows lambda function “SQS_SNS” is created.
- Fig 49, 50, 51 and 51.1 shows the lambda function.

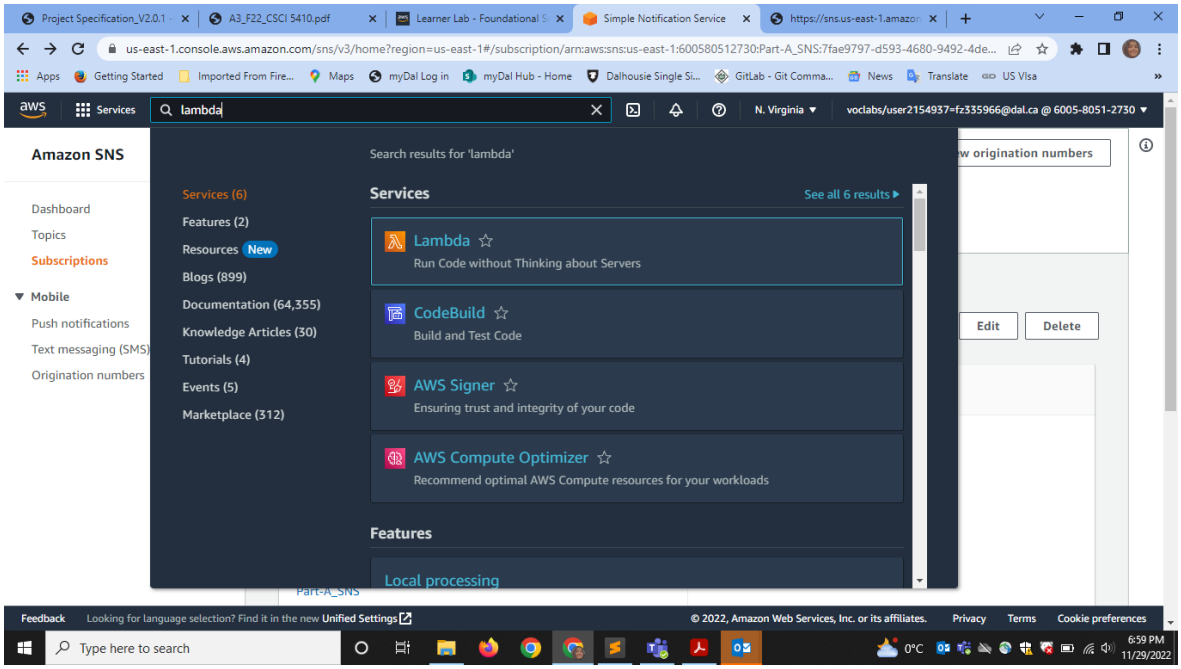


Fig 43: Searching for AWS lambda

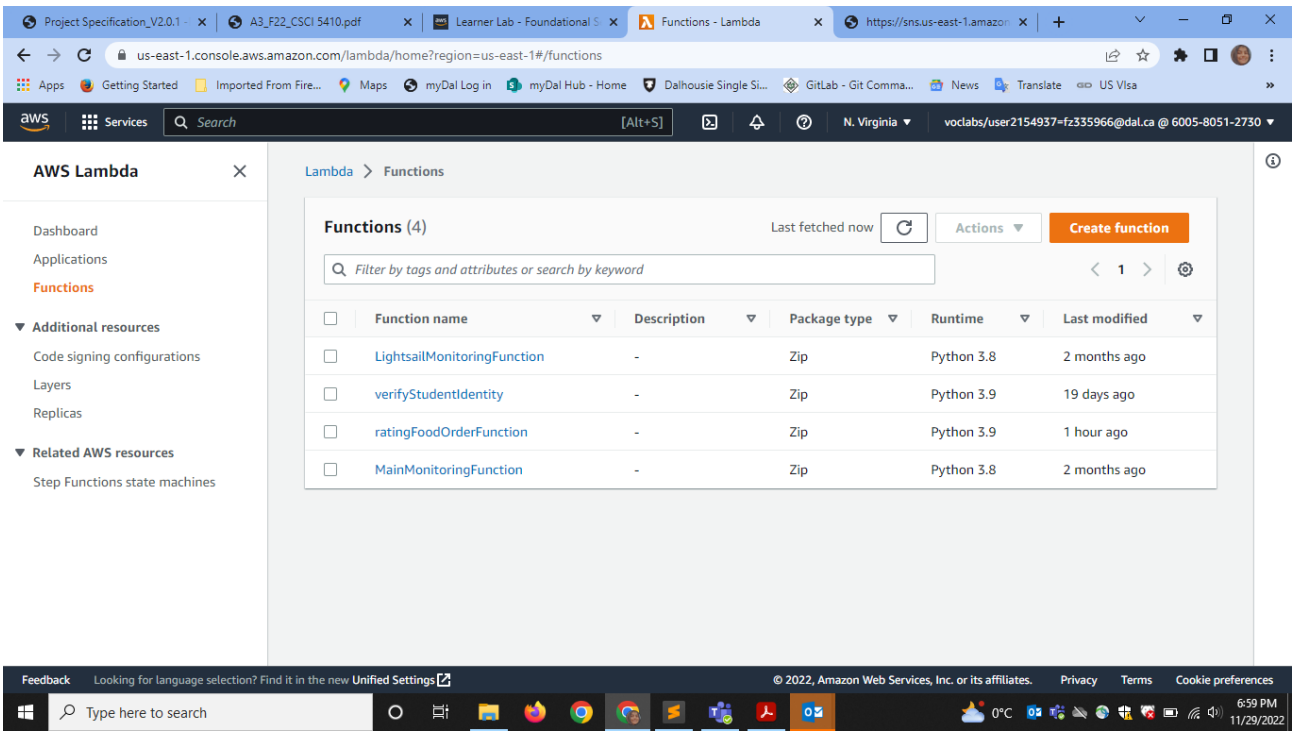


Fig 44: Create function page

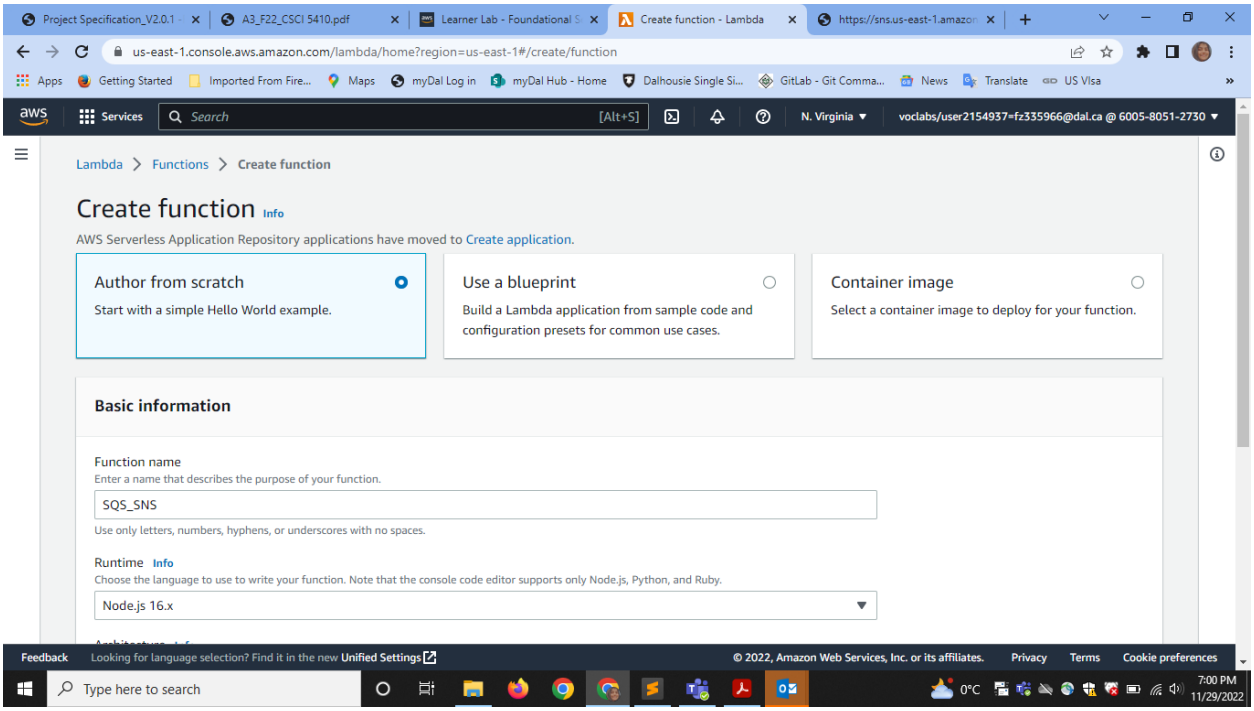


Fig 45: Create function form

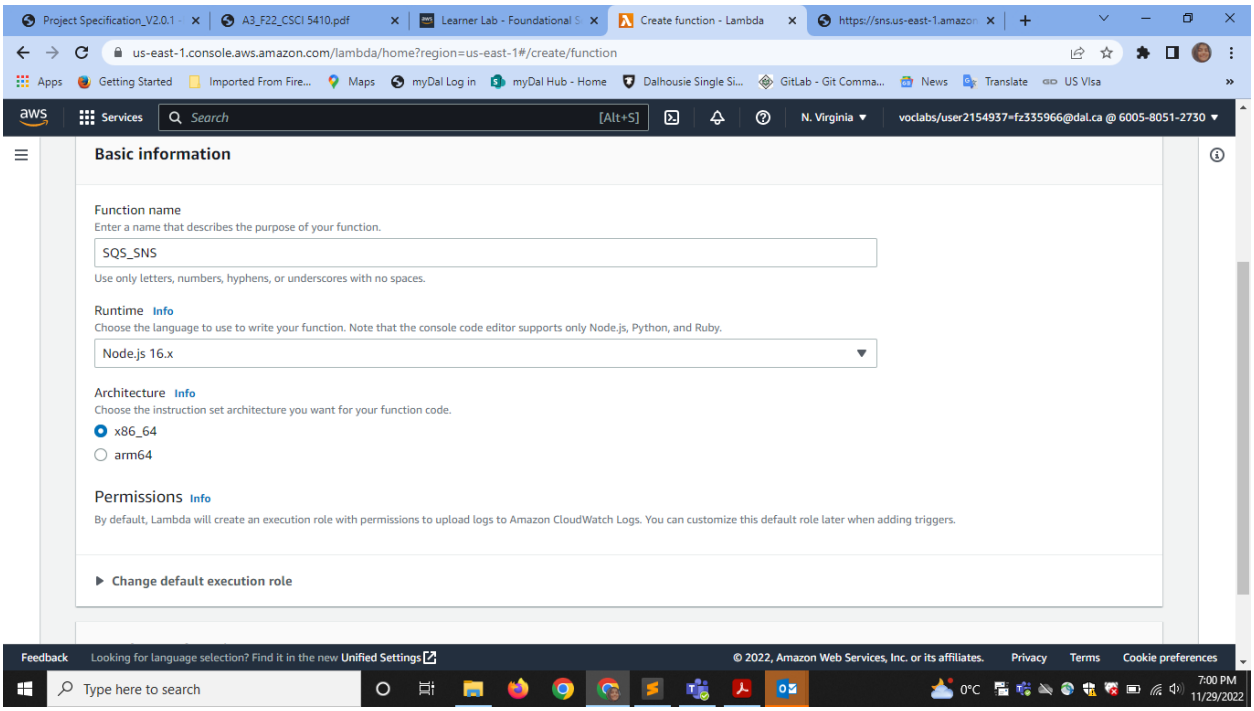


Fig 46: Create function form

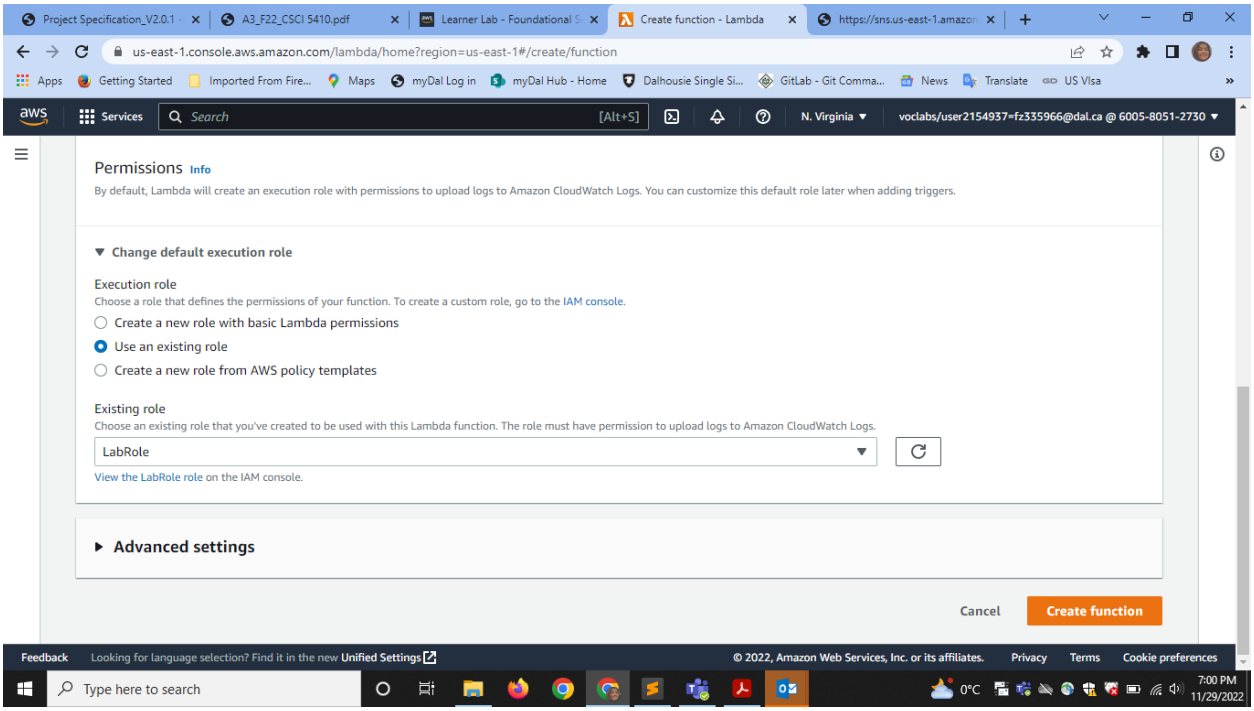


Fig 47: Create function form

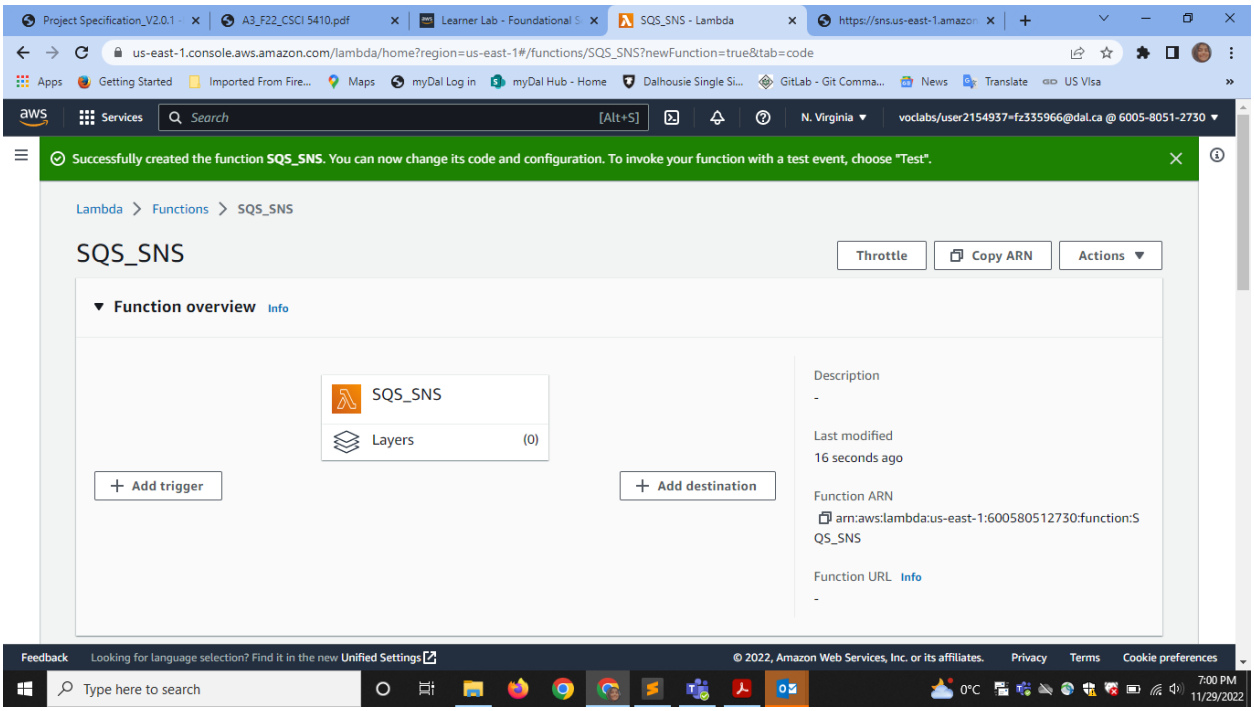


Fig 48: Lambda function "SQS_SNS" created

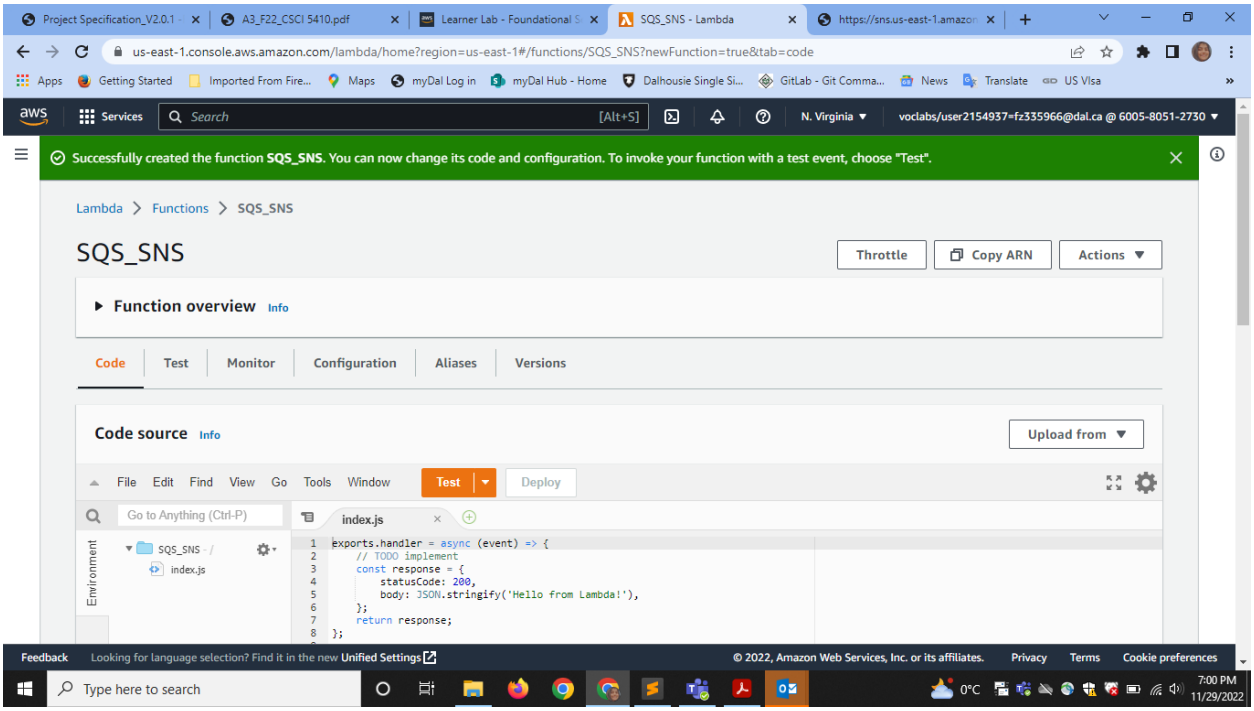


Fig 49: Lambda function "SQS_SNS" created

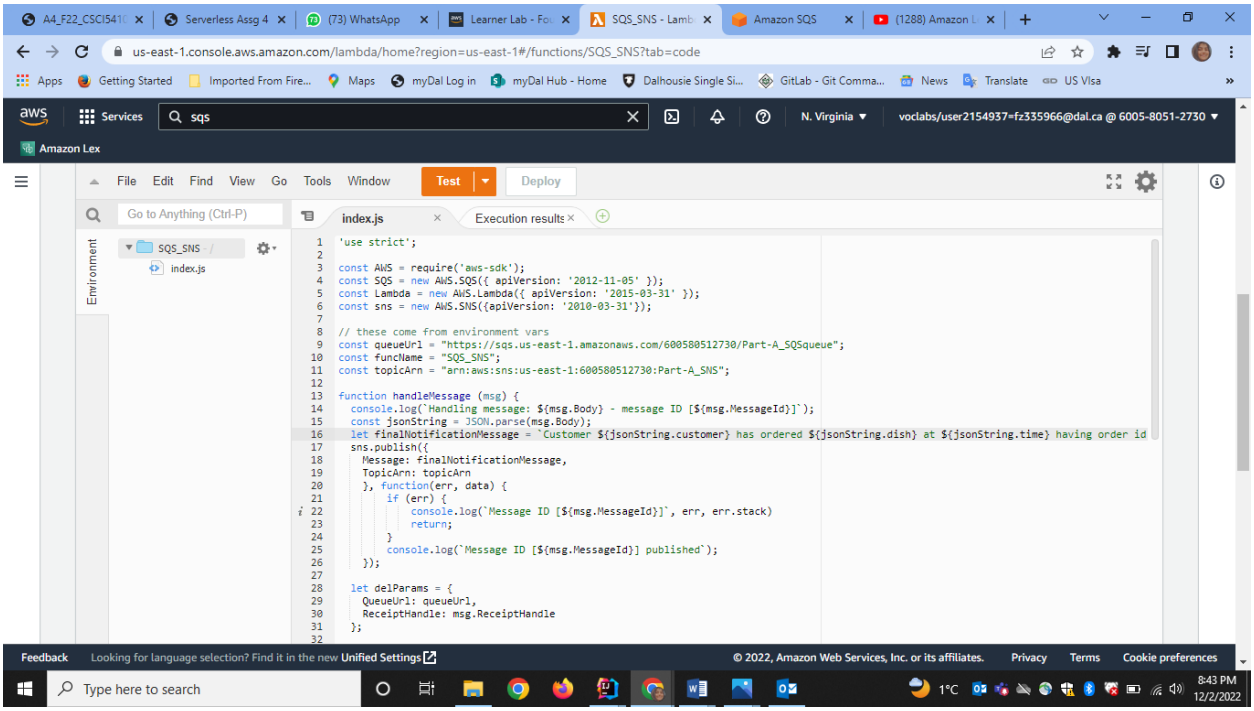


Fig 50: Lambda function

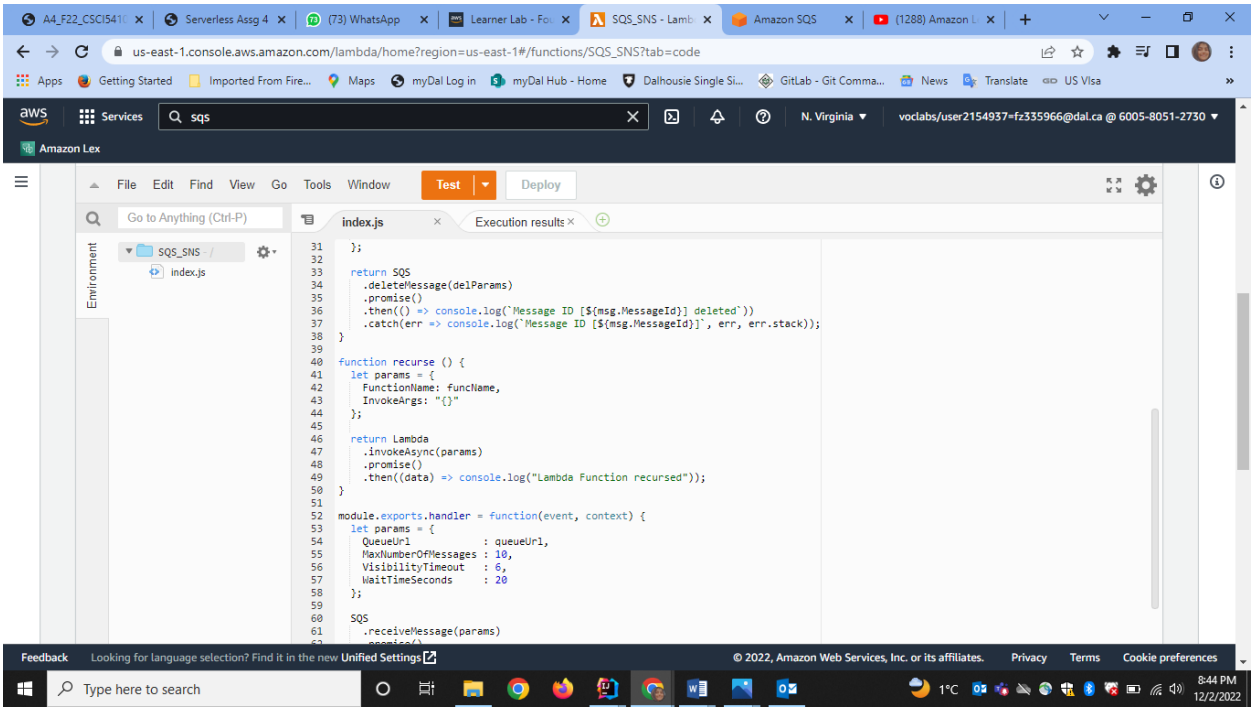


Fig 51: Lambda function

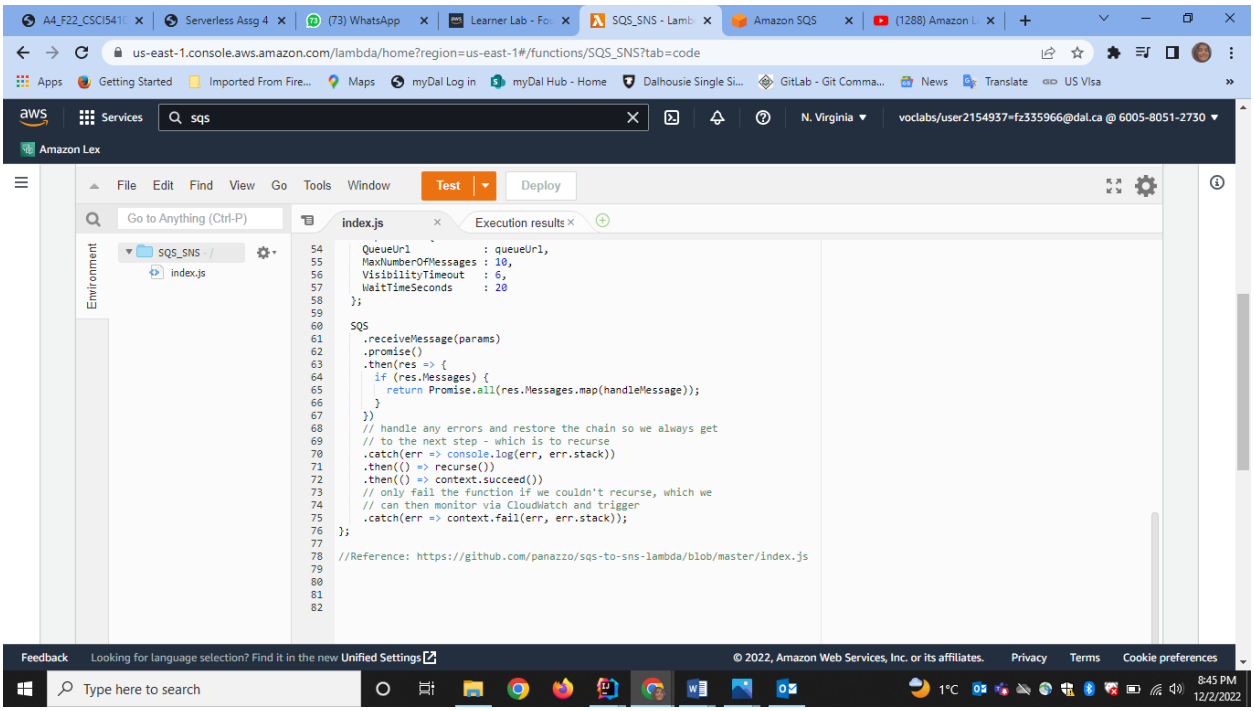


Fig 51.1: Lambda function

Step 6: To add trigger for receiving messaging at some time interval.

- To add trigger, click on the "Add trigger" option as shown in the fig 48.
- Fill in the details like adding rule name and schedule expression as shown in the fig 52 and 53.
- Click on "Add trigger" as shown in the fig 53.
- The EventBridge trigger has been added as shown in the fig 54.
- On clicking the "test" (Refer fig 60), I was prompt to "configure test event page" as shown in the fig 54.1 and 54.2.
- Fill the details as shown in the fig 54.1 and 54.2.

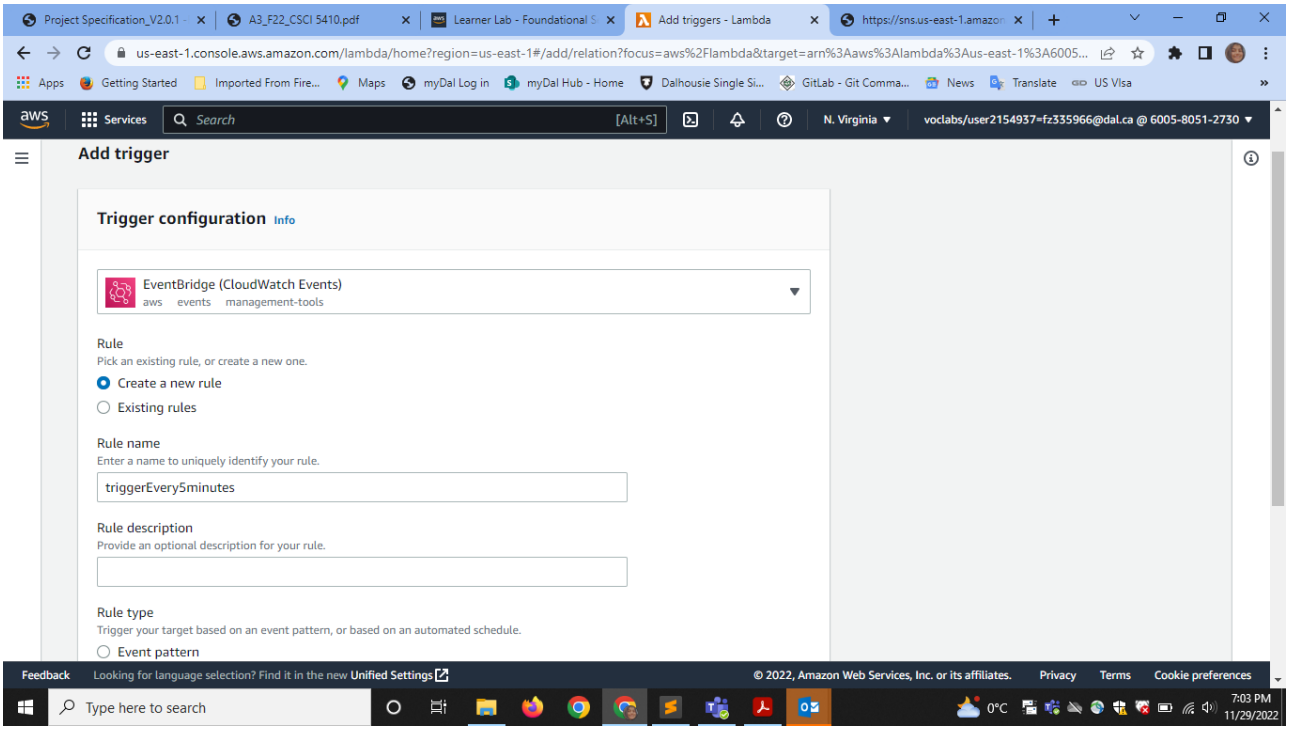


Fig 52: "Trigger configuration" form

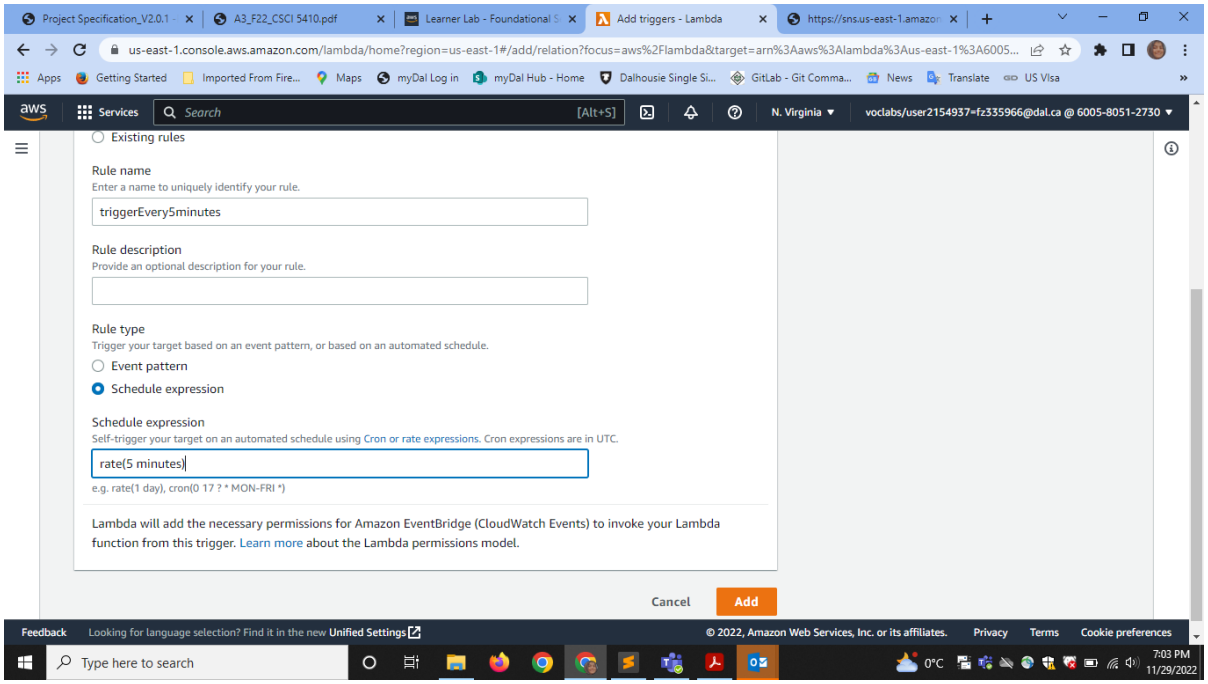


Fig 53: "Trigger configuration" form

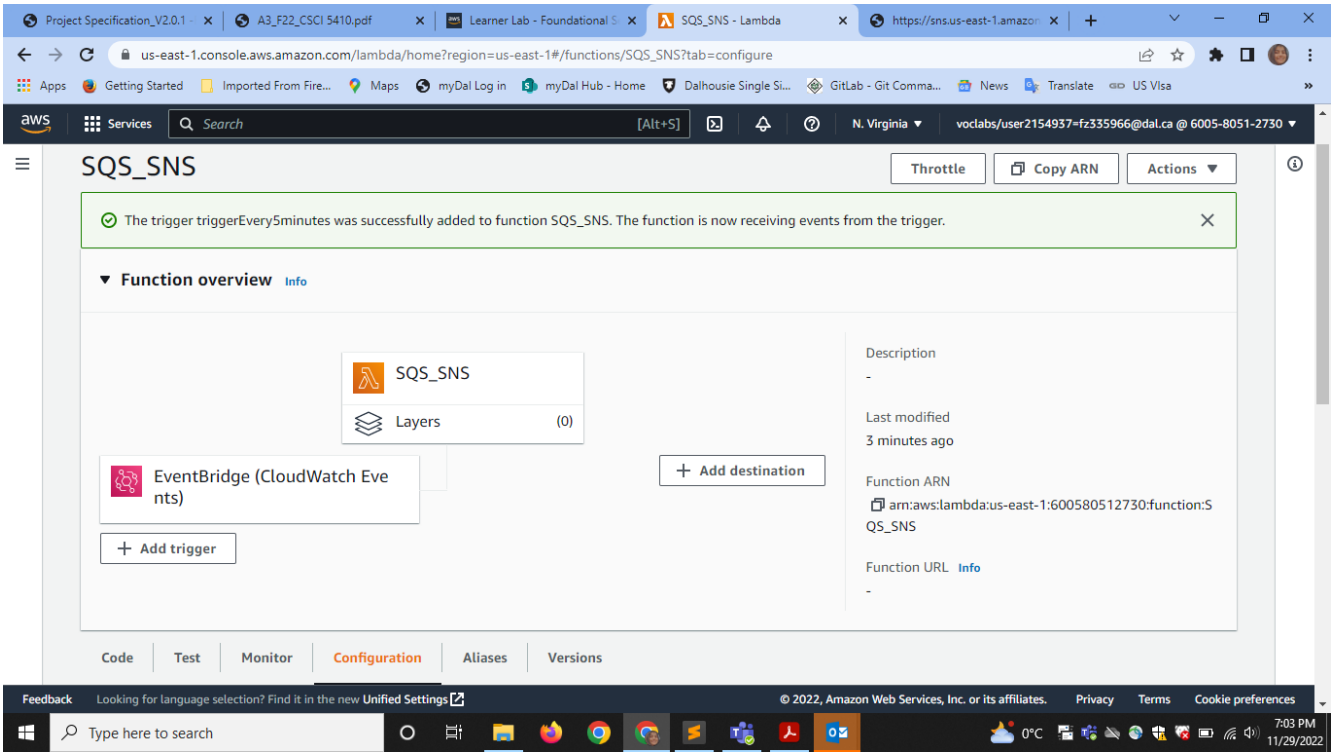


Fig 54: "EventBridge" trigger added

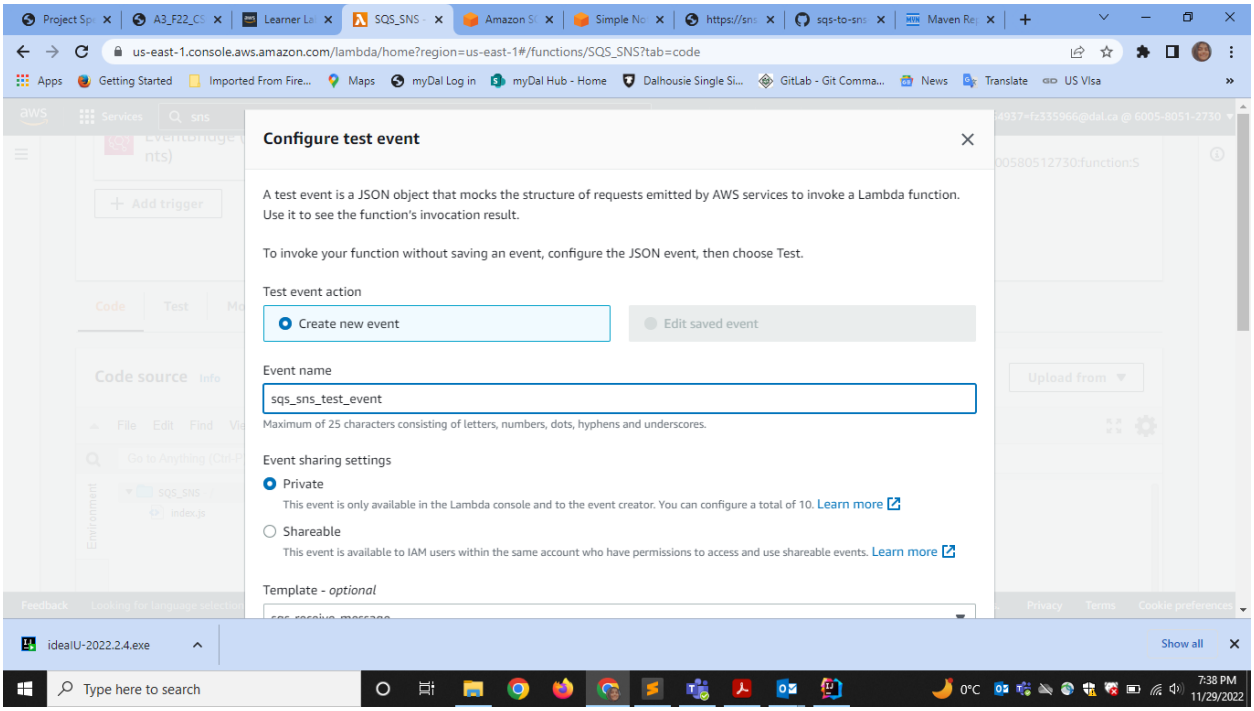


Fig 54.1: Configure test event page

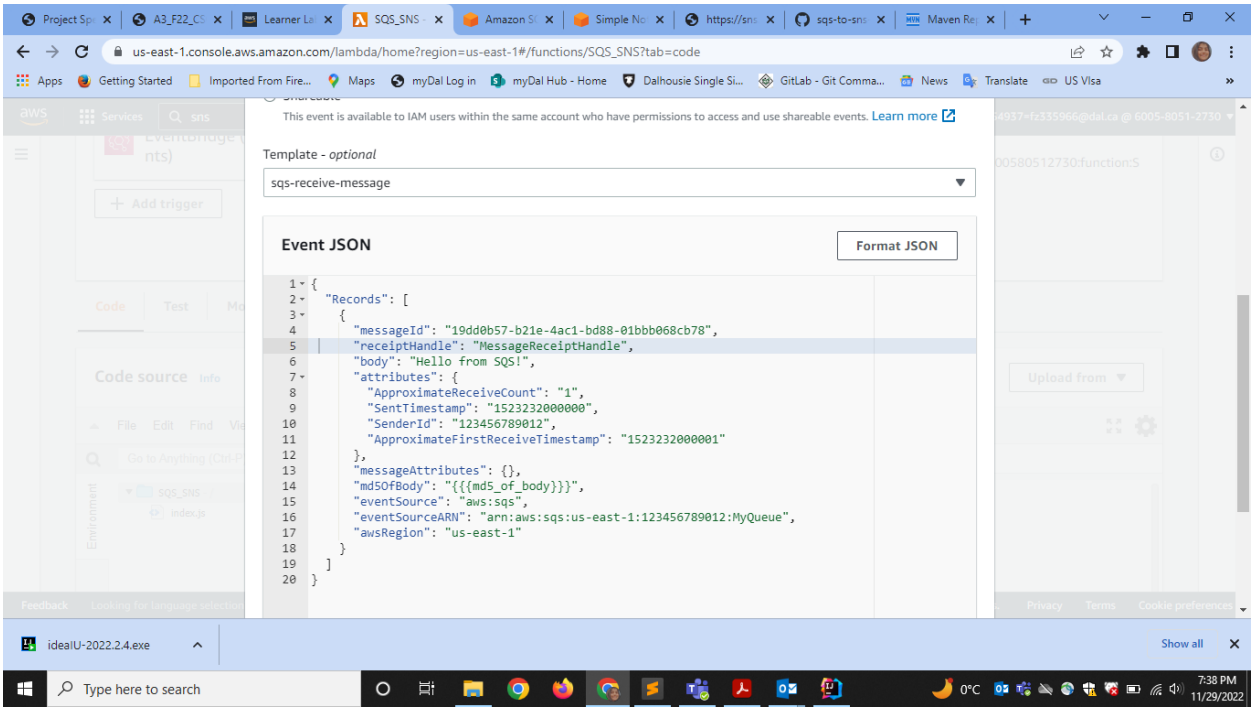


Fig 54.2: Template for Event JSON page

Step 7: Code Execution

- Write the java code and run the code as shown in the fig 55.
- Go to “Part-A_SQSqueue” (Refer fig 56) and click on “send and receive message” option as shown in the fig 56.
- To check whether we received message or not which is been sent, click on “Polling message” option which is shown in fig 57 and 58.
- On clicking the “polling message”, it shows the status of message receiving as shown in fig 58.
- After receiving the message (Refer fig 58) click on the message and you will see the actual json format message which is shown in fig 59.
- Go to the lambda function and test and deploy to get the message on your provided email as shown in the fig 60.
- Fig 61, shows that the message is been received after every interval of minutes with the order message i.e. previous mail was at 8.10pm and the next mail was at 8.15 pm.

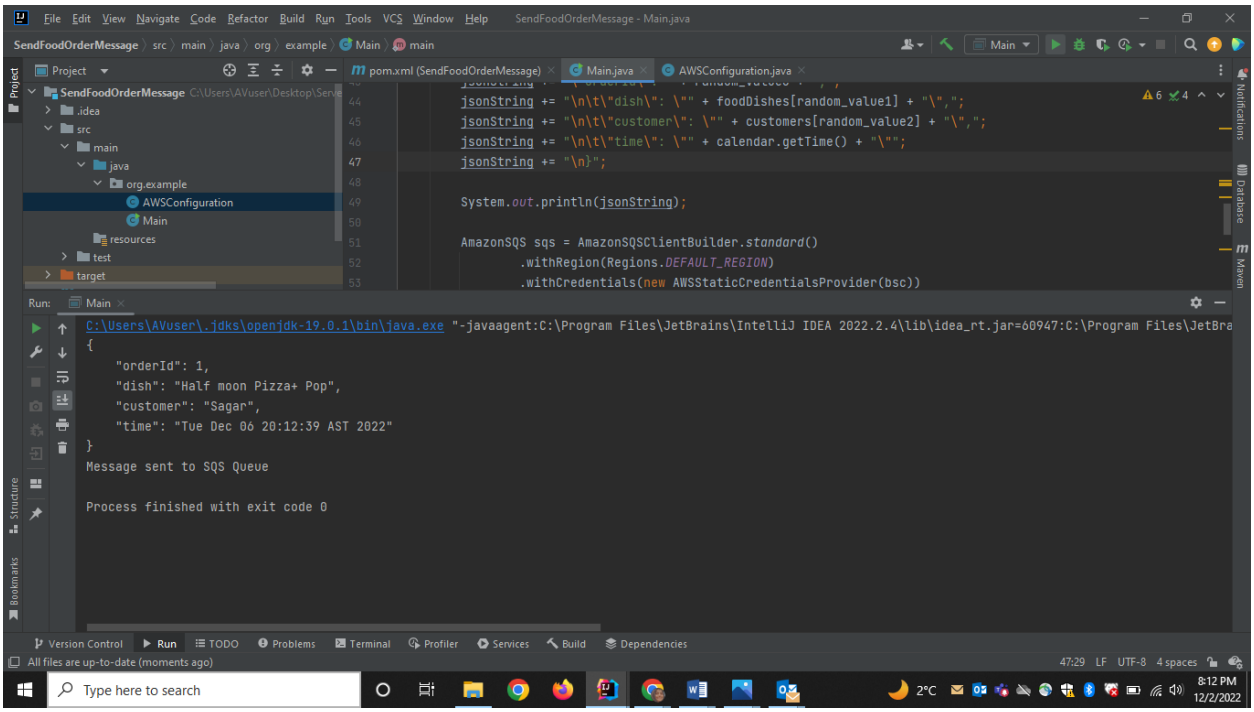


Fig 55: Terminal represents message sent to SQS

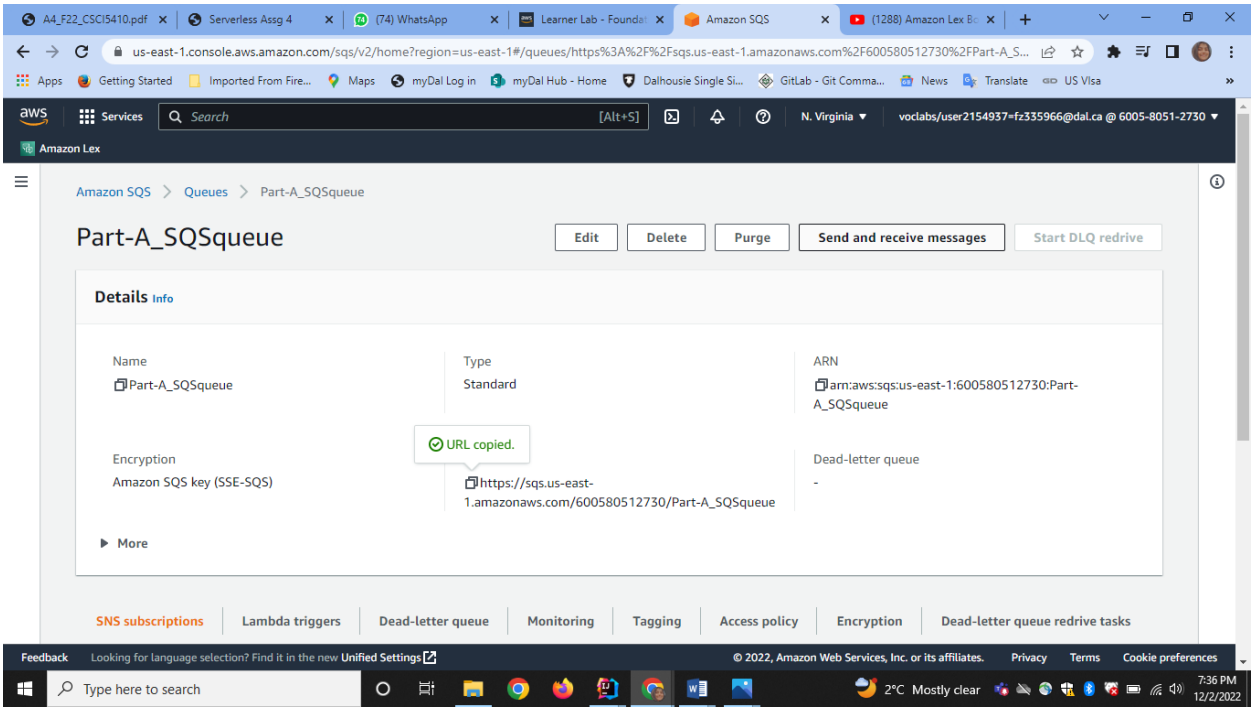


Fig 56: Part-A_SQSqueue page

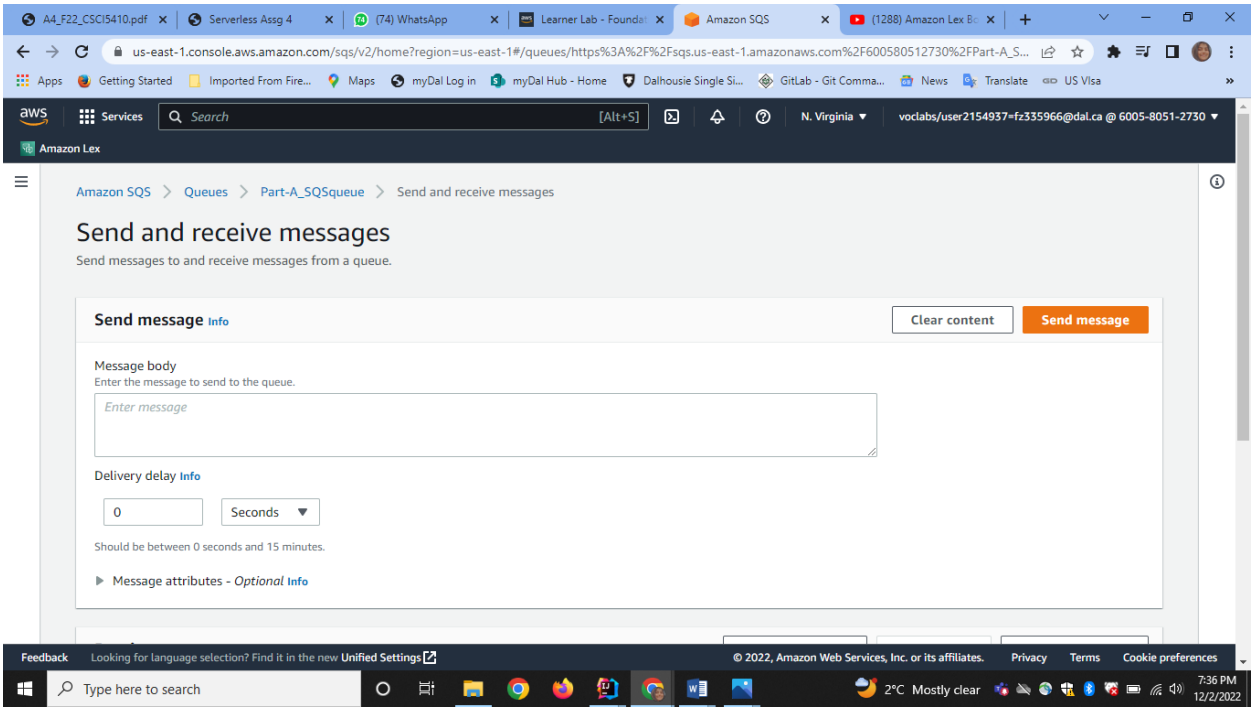


Fig 57: Send and receive message page

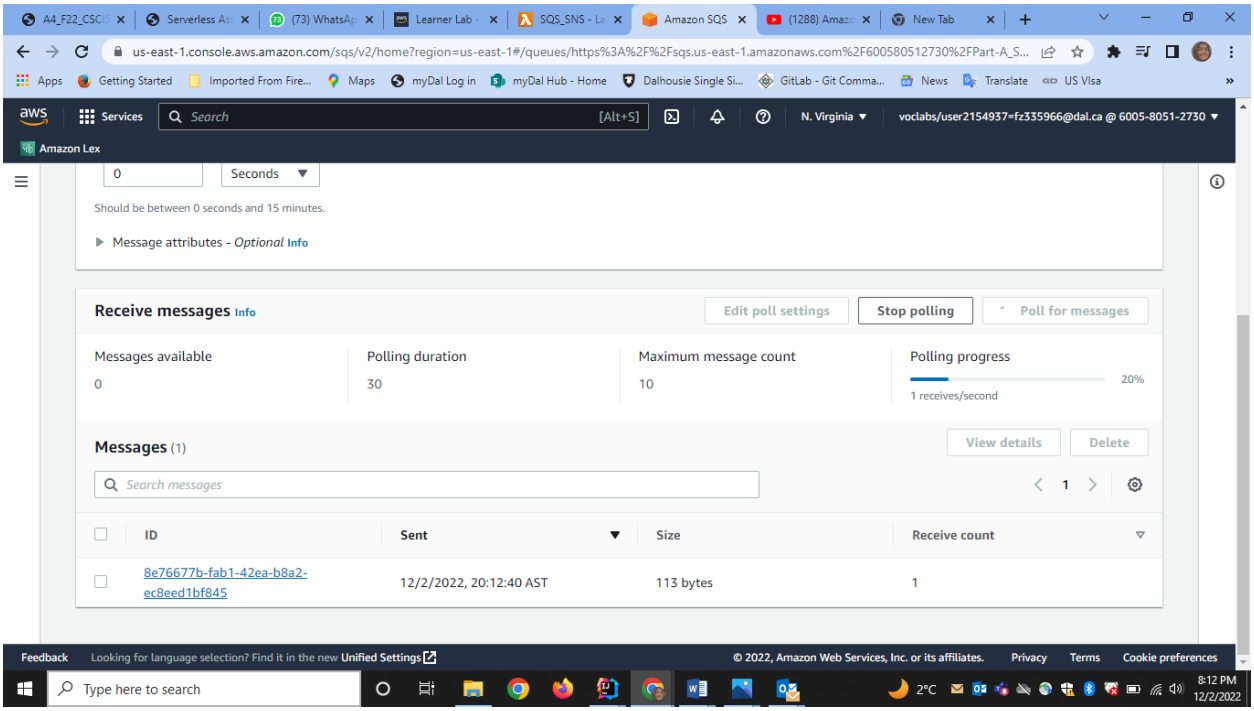


Fig 58: Send and receive message page – Message received

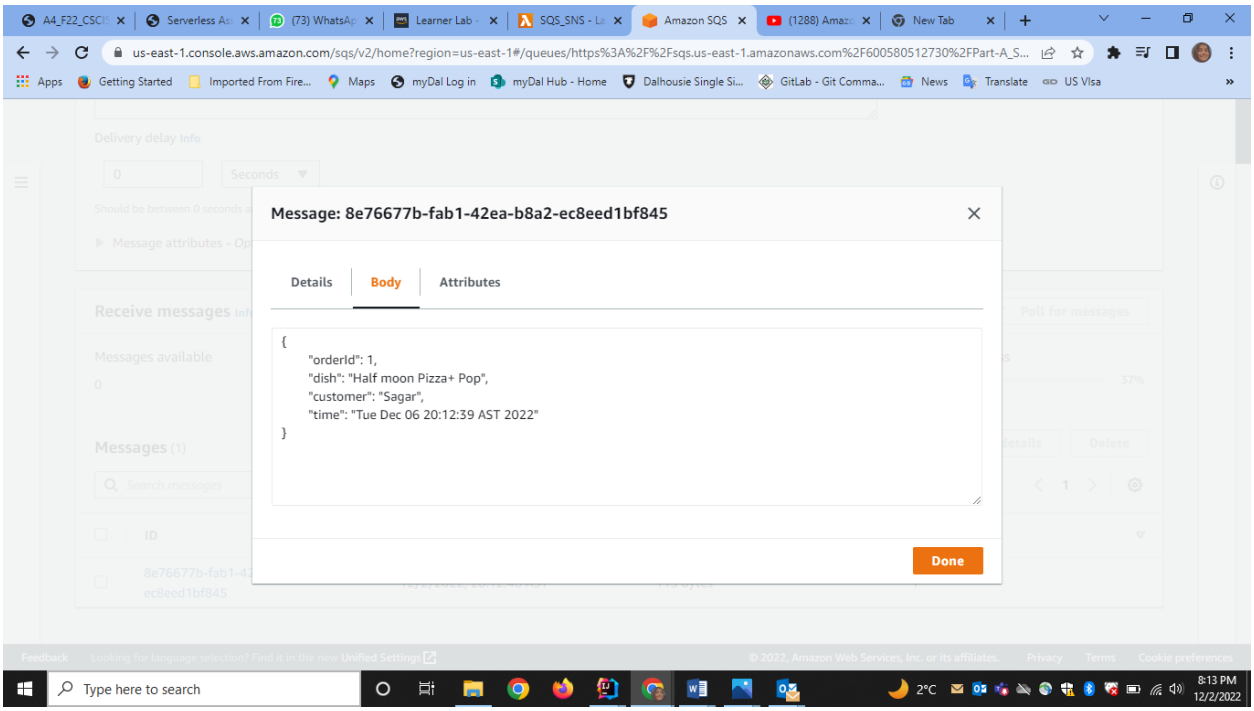


Fig 59: Send and receive message page – Message received

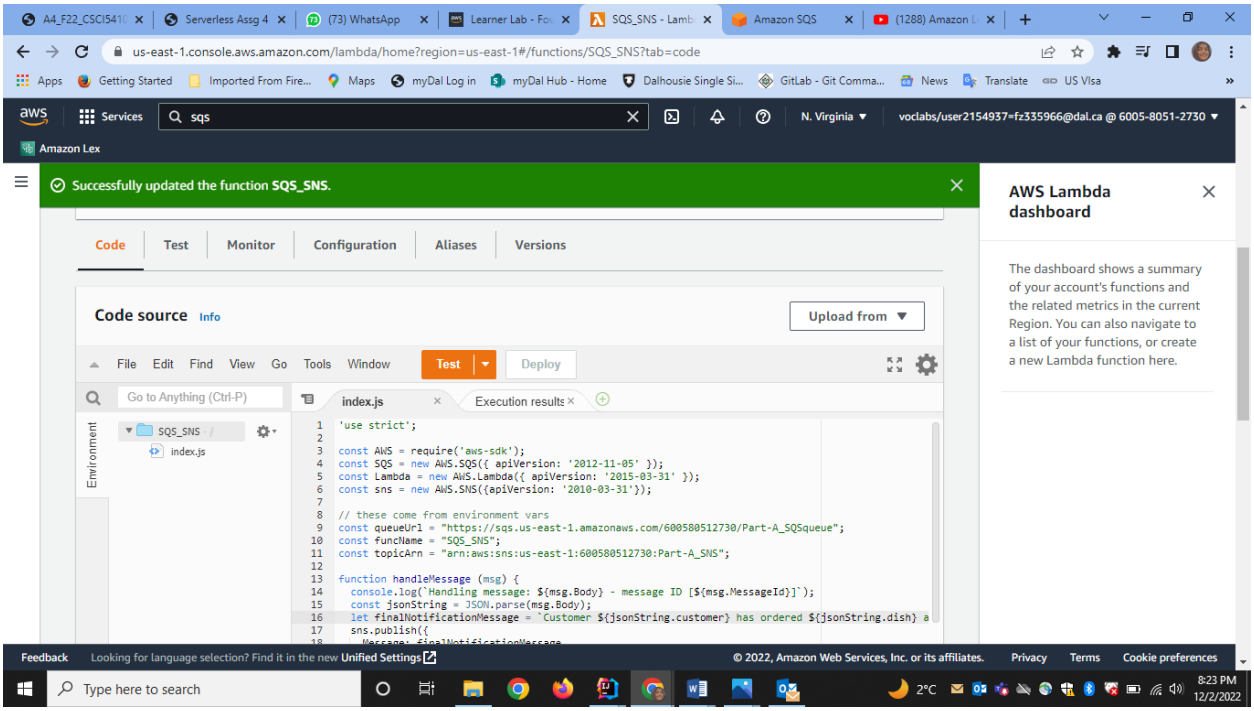


Fig 60: Test and deploy Lambda Function

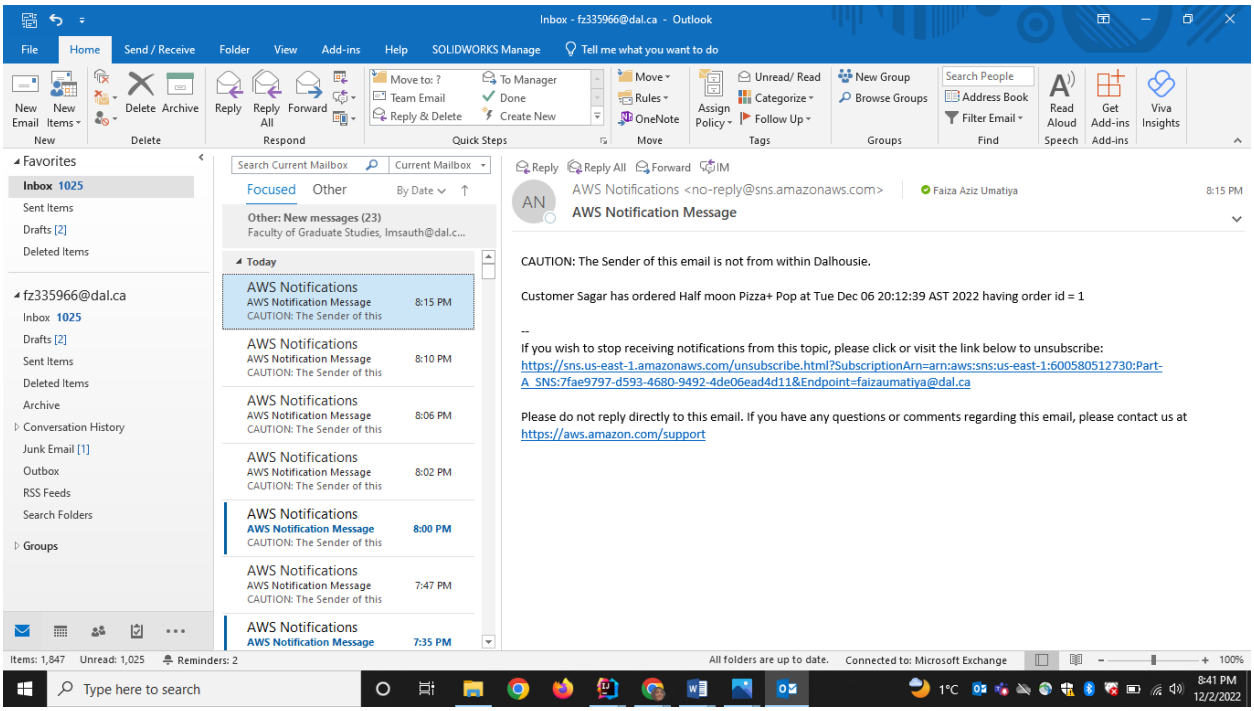


Fig 61: AWS notification having customer order message

Code Screenshots:

Main.java:

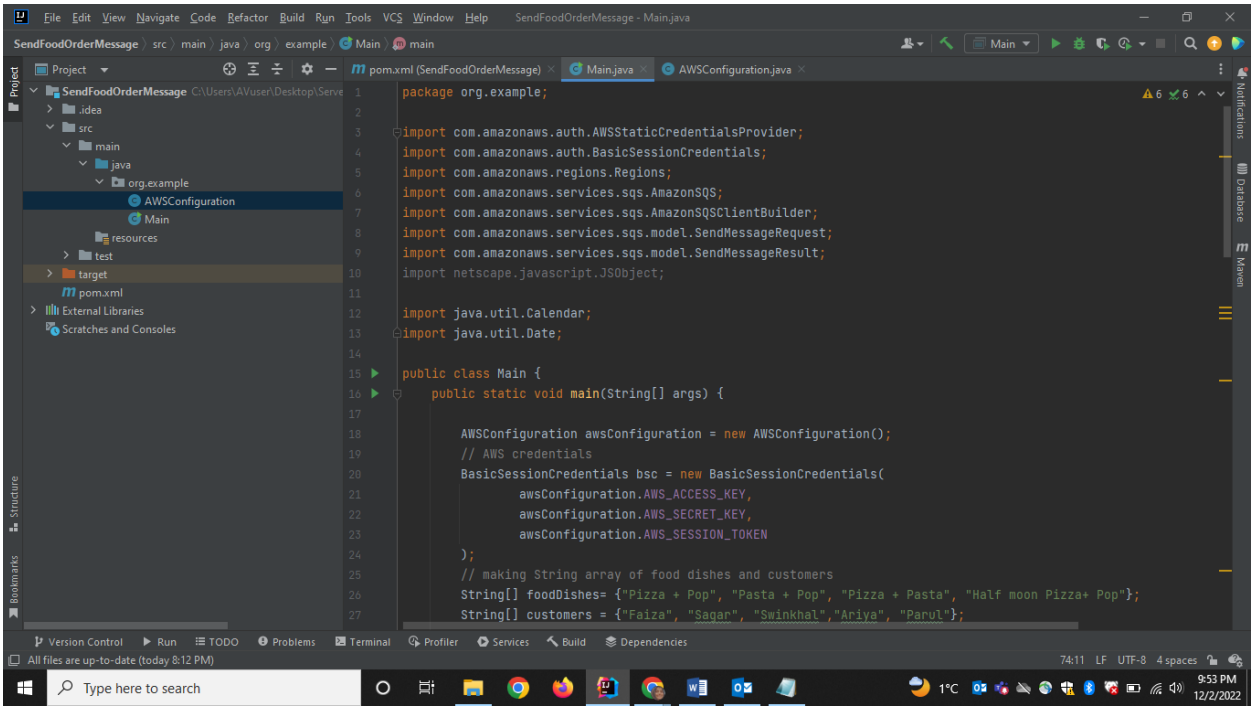


Fig 62: Main.java

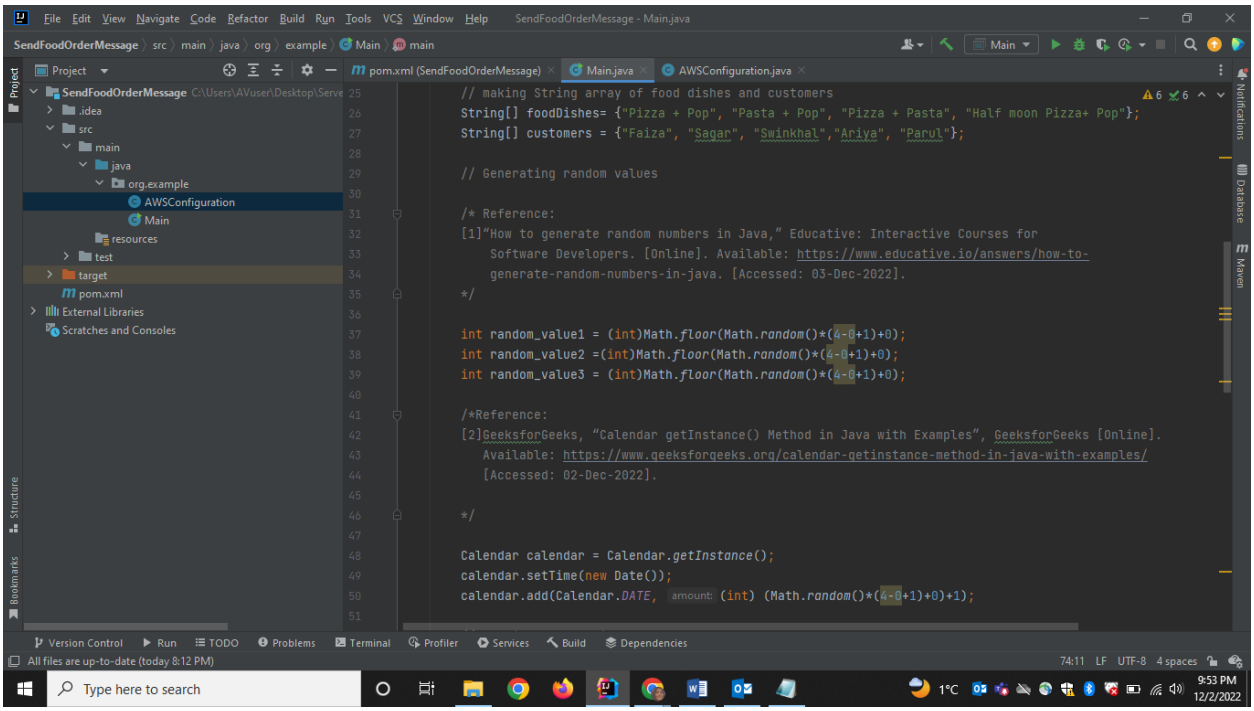


Fig 63: Main.java

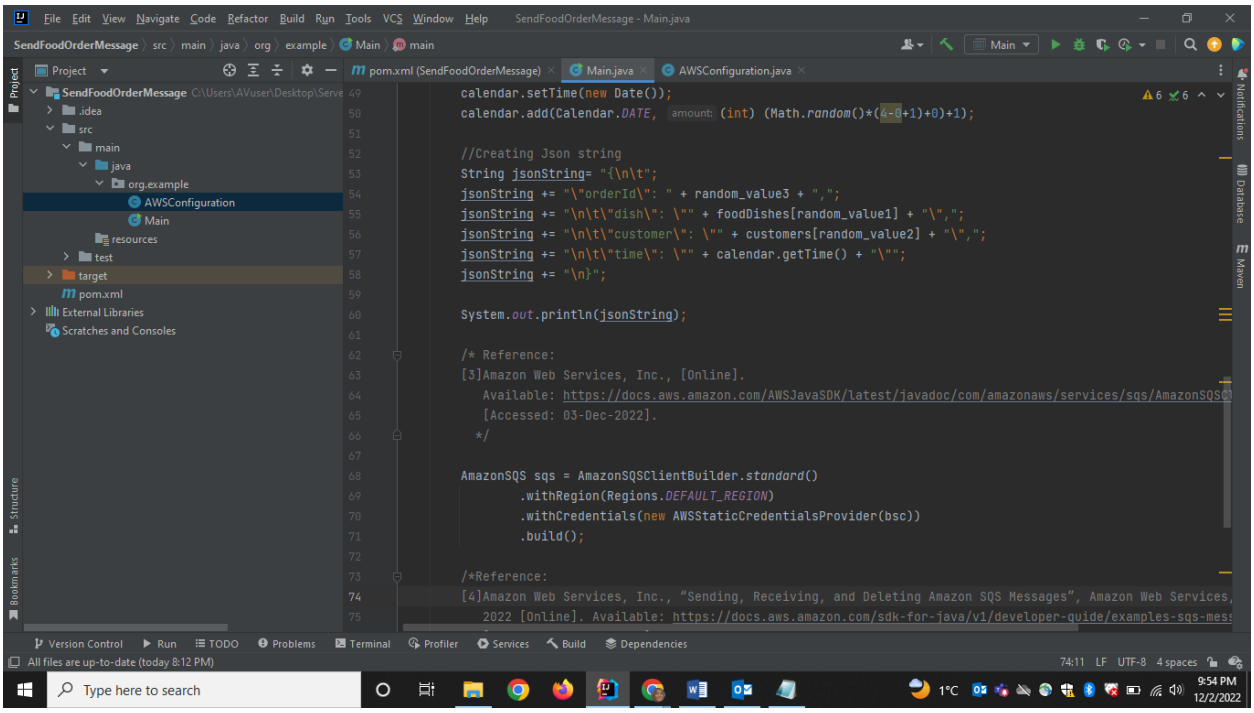


Fig 64: Main.java

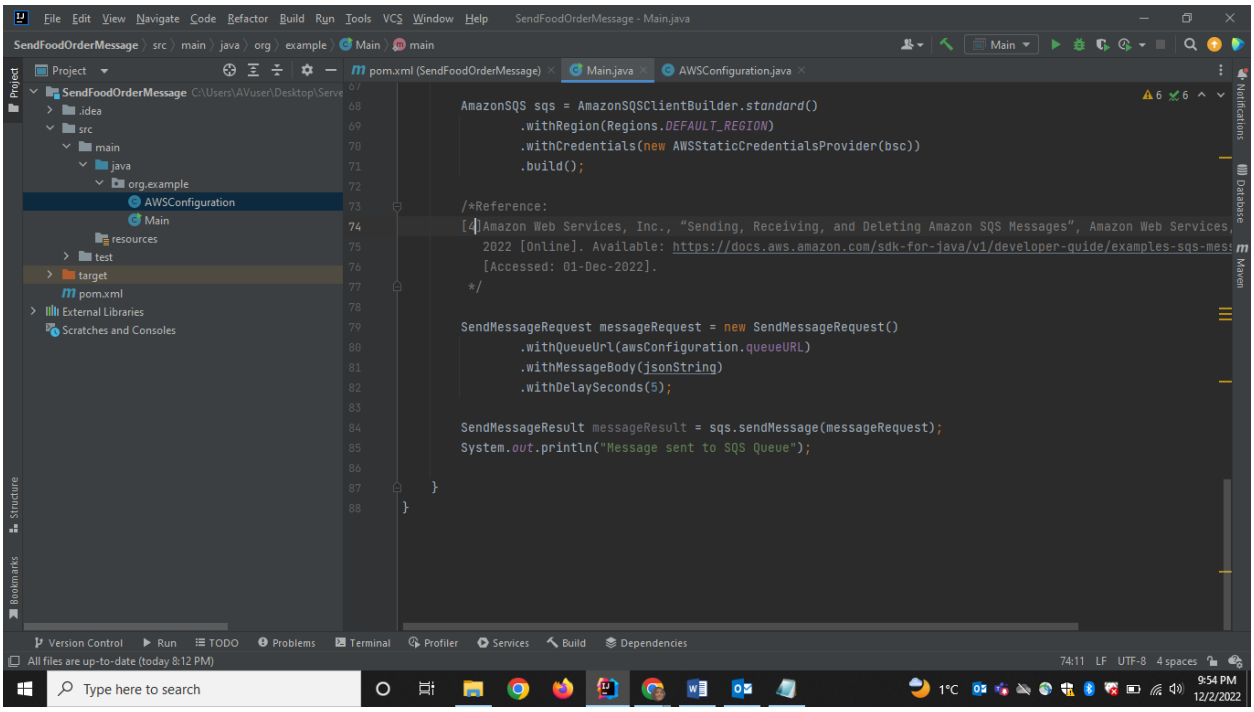


Fig 65: Main.java

AWSConfiguration.java:

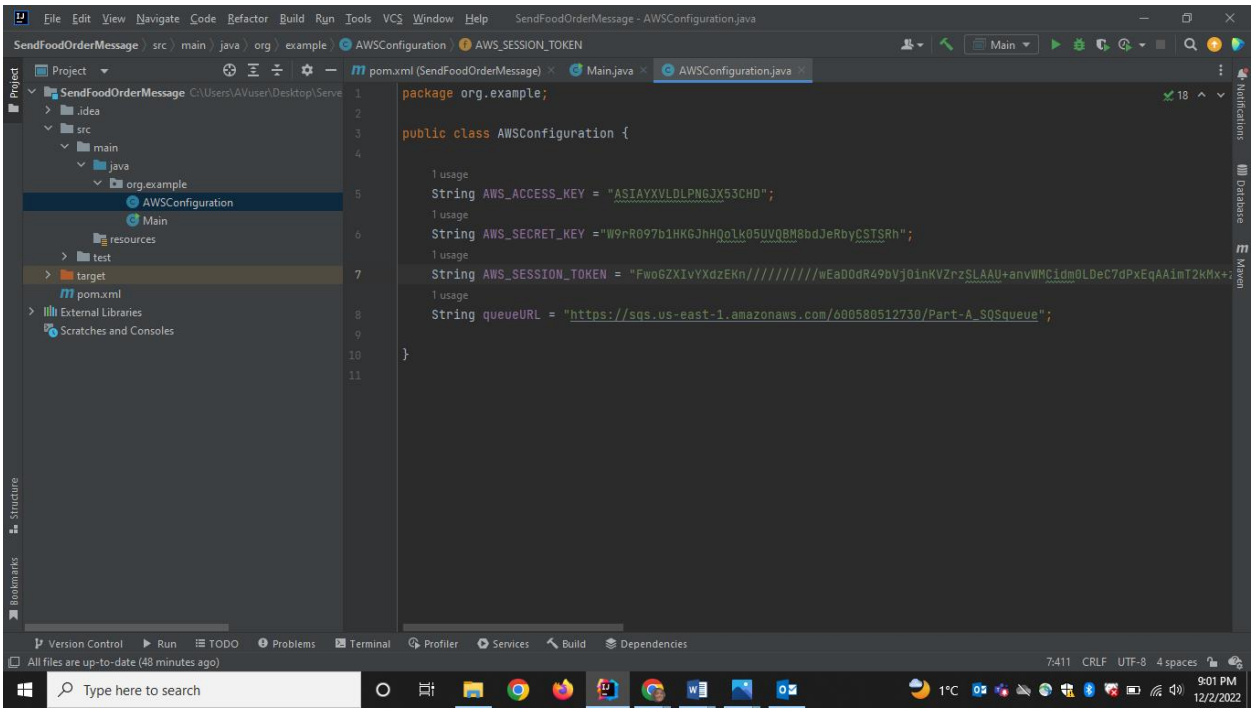


Fig 66: AWSConfiguration.java

Code Script:

I created 2 java class:

Main.java:

```
package org.example;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageResult;
import netscape.javascript.JSObject;

import java.util.Calendar;
import java.util.Date;

public class Main {
    public static void main(String[] args) {

        AWSConfiguration awsConfiguration = new AWSConfiguration();
        // AWS credentials
        BasicSessionCredentials bsc = new BasicSessionCredentials(
```

```
            awsConfiguration.AWS_ACCESS_KEY,
            awsConfiguration.AWS_SECRET_KEY,
            awsConfiguration.AWS_SESSION_TOKEN
        );
        // making String array of food dishes and customers
        String[] foodDishes= {"Pizza + Pop", "Pasta + Pop", "Pizza + Pasta", "Half moon Pizza+ Pop"};
        String[] customers = {"Faiza", "Sagar", "Swinkhal","Ariya", "Parul"};

        // Generating random values

        /* Reference:
        [1]"How to generate random numbers in Java," Educative: Interactive Courses for
           Software Developers. [Online]. Available: https://www.educative.io/answers/how-to-
           generate-random-numbers-in-java. [Accessed: 03-Dec-2022].
        */

        int random_value1 = (int)Math.floor(Math.random()*(4-0+1)+0);
        int random_value2 =(int)Math.floor(Math.random()*(4-0+1)+0);
        int random_value3 = (int)Math.floor(Math.random()*(4-0+1)+0);

        /*Reference:
        [2]GeeksforGeeks, "Calendar getInstance() Method in Java with Examples", GeeksforGeeks [Online].
           Available: https://www.geeksforgeeks.org/calendar-getinstance-method-in-java-with-examples/
           [Accessed: 02-Dec-2022].

        */

        Calendar calendar = Calendar.getInstance();
        calendar.setTime(new Date());
        calendar.add(Calendar.DATE, (int) (Math.random()*(4-0+1)+0)+1);

        //Creating Json string
        String jsonString= "{\n\t";
        jsonString += "\"orderId\": " + random_value3 + ",";
        jsonString += "\n\t\"dish\": \"" + foodDishes[random_value1] + "\",";
        jsonString += "\n\t\"customer\": \"" + customers[random_value2] + "\",";
        jsonString += "\n\t\"time\": \"" + calendar.getTime() + "\"";
        jsonString += "\n}";

        System.out.println(jsonString);

        /* Reference:
        [3]Amazon Web Services, Inc., [Online].
           Available:
https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/sqs/AmazonSQSClientBuilder
.html.
           [Accessed: 03-Dec-2022].
         */

        AmazonSQS sqs = AmazonSQSClientBuilder.standard()
                .withRegion(Regions.DEFAULT_REGION)
                .withCredentials(new AWSStaticCredentialsProvider(bsc))
                .build();

        /*Reference:
        [4]Amazon Web Services, Inc., "Sending, Receiving, and Deleting Amazon SQS Messages", Amazon Web
```

```
Services, Inc.,
        2022 [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs-messages.html
        [Accessed: 01-Dec-2022].
       */

      SendMessageRequest messageRequest = new SendMessageRequest()
            .withQueueUrl(awsConfiguration.queueURL)
            .withMessageBody(jsonString)
            .withDelaySeconds(5);

      SendMessageResult messageResult = sqs.sendMessage(messageRequest);
      System.out.println("Message sent to SQS Queue");

   }
}
```

AWSConfiguration.java:

```
package org.example;

public class AWSConfiguration {

   String AWS_ACCESS_KEY = "ASIAYXVLDLPNGJX53CHD";
   String AWS_SECRET_KEY ="W9rR097b1HKGJhHQolk05UVQBM8bdJeRbyCSTSRh";
   String AWS_SESSION_TOKEN =
"FwoGZXIvYXdzEKn/////////wEaDOdR49bVj0inKVZrzSLAAU+anvWMCidm0LDeC7dPxEqAAimT2kMx+zeB
YSZB0wYcMMf82CUqOOp4h41UGBhD/URriDiOYiLf/NDz0+30kJICHazXmrOGBTLilJ+eNjnD6D/Ec1faMNM
Y3j6nr9TNLGSIulHG1mY4eDxexPOk1SPbHxguepTGVU5IHL4StXDWqxcc+02XG4CD3No5ZbE1+TZftGhhjW
2yWfdSsByTd/UDSfUt2FDPf6DvP597U795hpwDjFJq2QvFG4eLKpd5Cii3laqcBjItzfe0fcCL9ibP3o71iCGLSRy0f
4DEI8EMvU6ZfzLh32e+JVCWCOhAI3KxVhAo";
   String queueURL = "https://sqs.us-east-1.amazonaws.com/600580512730/Part-A_SQSqueue";

}
```

Lambda function: index.js (code reference [5])

```
'use strict';

const AWS = require('aws-sdk');

const SQS = new AWS.SQS({ apiVersion: '2012-11-05' });

const Lambda = new AWS.Lambda({ apiVersion: '2015-03-31' });

const sns = new AWS.SNS({apiVersion: '2010-03-31'});

// these come from environment vars

const queueUrl = "https://sqs.us-east-1.amazonaws.com/600580512730/Part-A_SQSqueue";

const funcName = "SQS_SNS";

const topicArn = "arn:aws:sns:us-east-1:600580512730:Part-A_SNS";
```

```
function handleMessage (msg) {
  console.log(`Handling message: ${msg.Body} - message ID [${msg.MessageId}]`);
  const jsonString = JSON.parse(msg.Body);
  let finalNotificationMessage = `Customer ${jsonString.customer} has ordered ${jsonString.dish} at
${jsonString.time} having order id = ${jsonString.orderId}`;
  sns.publish({
    Message: finalNotificationMessage,
    TopicArn: topicArn
    }, function(err, data) {
        if (err) {
            console.log(`Message ID [${msg.MessageId}]`, err, err.stack)
            return;
        }
        console.log(`Message ID [${msg.MessageId}] published`);
    });

  let delParams = {
    QueueUrl: queueUrl,
    ReceiptHandle: msg.ReceiptHandle
  };

  return SQS
    .deleteMessage(delParams)
    .promise()
    .then(() => console.log(`Message ID [${msg.MessageId}] deleted`))
    .catch(err => console.log(`Message ID [${msg.MessageId}]`, err, err.stack));
}

function recurse () {
```

```
  let params = {
    FunctionName: funcName,
    InvokeArgs: "{}"
  };

  return Lambda
    .invokeAsync(params)
    .promise()
    .then((data) => console.log("Lambda Function recursed"));
}

module.exports.handler = function(event, context) {
  let params = {
    QueueUrl            : queueUrl,
    MaxNumberOfMessages : 10,
    VisibilityTimeout   : 6,
    WaitTimeSeconds     : 20
  };

  SQS
    .receiveMessage(params)
    .promise()
    .then(res => {
      if (res.Messages) {
        return Promise.all(res.Messages.map(handleMessage));
      }
    })
    // handle any errors and restore the chain so we always get
    // to the next step - which is to recurse
    .catch(err => console.log(err, err.stack))
```

```
    .then(() => recurse())
    .then(() => context.succeed())
    // only fail the function if we couldn't recurse, which we
    // can then monitor via CloudWatch and trigger
    .catch(err => context.fail(err, err.stack));
};
```

References:

[1] Amazon Web Services, Inc., “Creating an Amazon SNS topic”, Amazon Web Services, Inc., 2022 [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/sns-create-topic.html [Accessed: 01-Dec-2022].

[2] Amazon Web Services, Inc., “Create a queue (console)”, Amazon Web Services, Inc., 2022 [Online]. Available: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/step-create-queue.html [Accessed: 01-Dec-2022].

[3] Amazon Web Services, Inc., “Sending, Receiving, and Deleting Amazon SQS Messages”, Amazon Web Services, Inc., 2022 [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs-messages.html [Accessed: 01-Dec-2022].

[4] GeeksforGeeks, “Calendar getInstance() Method in Java with Examples”, GeeksforGeeks [Online]. Available: https://www.geeksforgeeks.org/calendar-getinstance-method-in-java-with-examples/ [Accessed: 02-Dec-2022].

[5] R. P. Roizman, “SQS-TO-SNS-LAMBDA,” GitHub, 2022[Online]. Available: https://github.com/panazzo/sqs-to-sns-lambda/blob/master/index.js [Accessed: 01-Dec-2022].

ASSIGNMENT 4 – PART B

I read and understood GCP BigQueryML, which helped me create KMeans cluster of the given dataset

Following are the steps I followed to solve the assignment:

Step 1:

- Open GCP BigQuery homepage as shown in the fig 1.

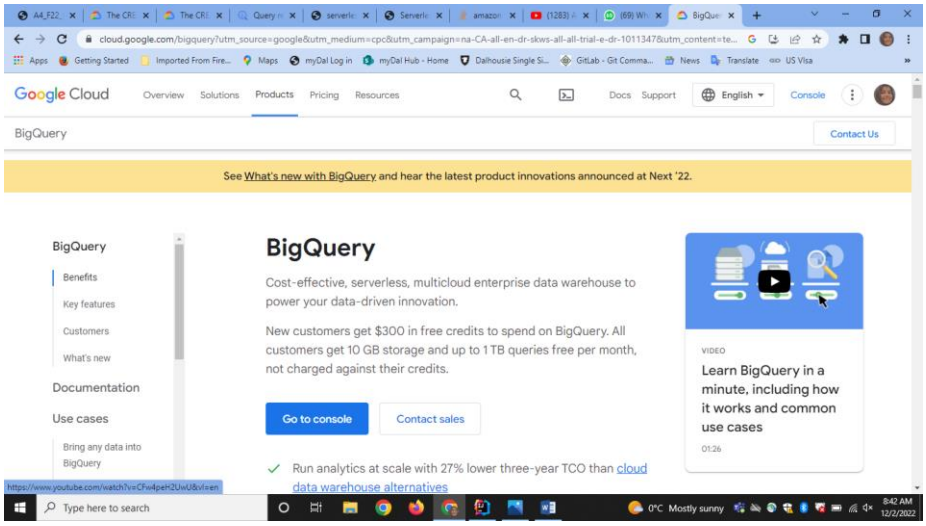


Fig 1: BigQuery home page

Step 2:

- Go to GCP console as shown in the fig 2.

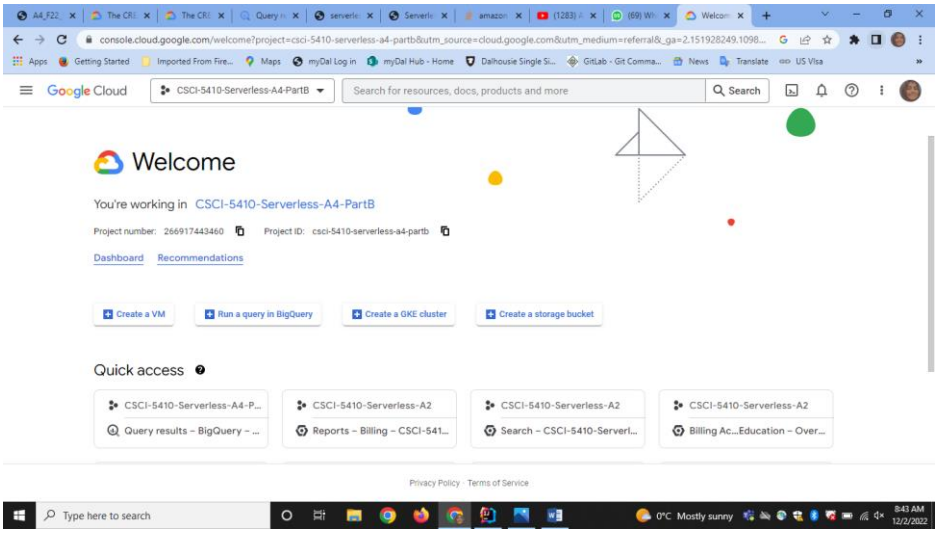


Fig 2: GCP console page

Step 3:

- Create a new project named “CSCI-5410-Serverless-A4-PartB” as shown in the fig 3, 4 & 5.

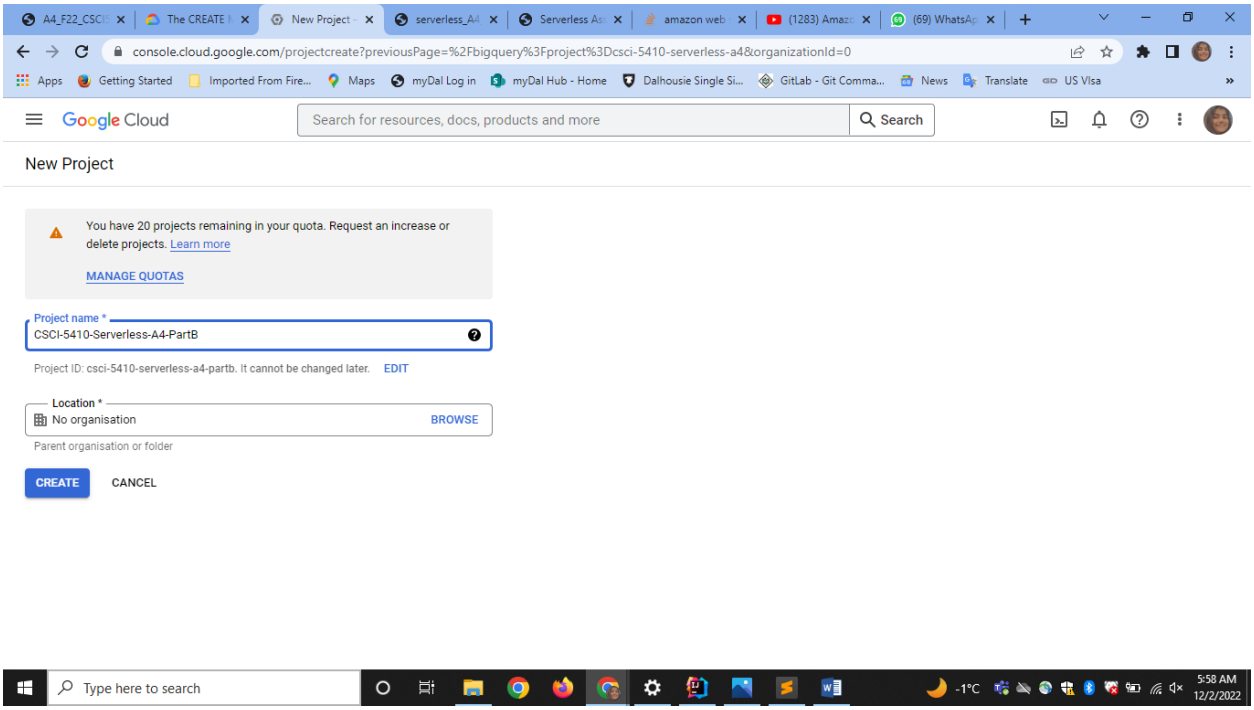


Fig 3: New project creation on GCP console

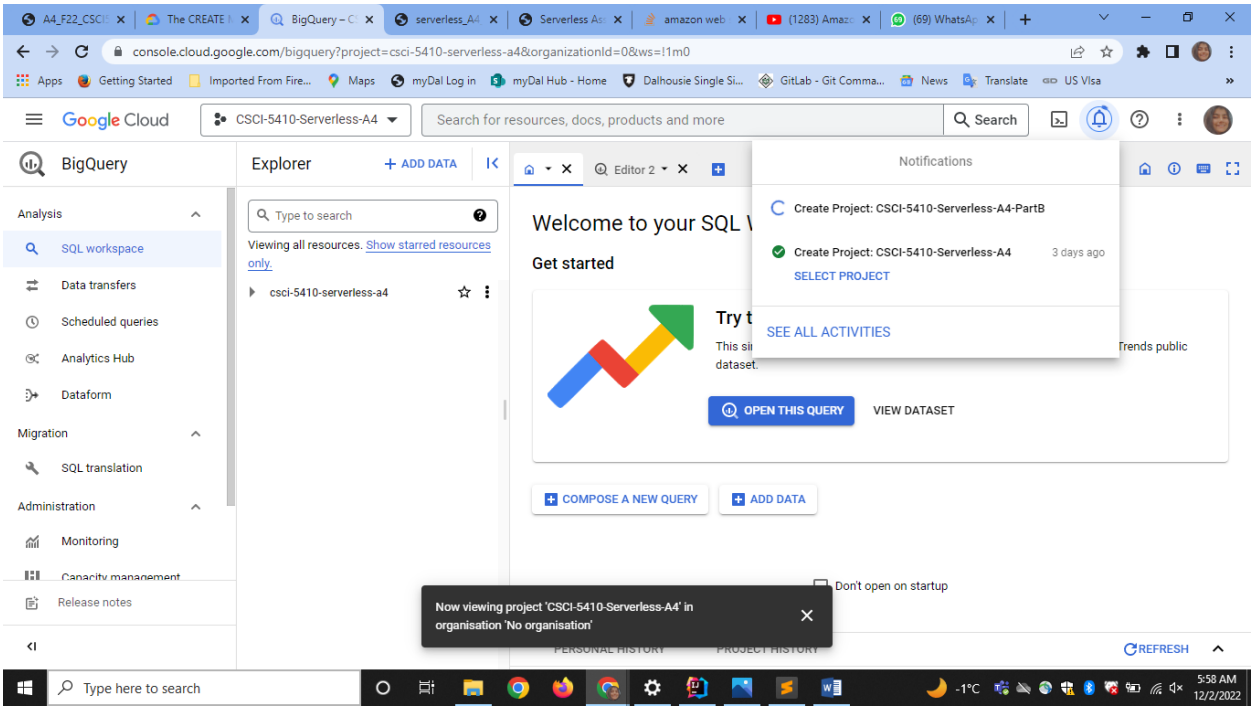


Fig 4: Created new project on GCP console

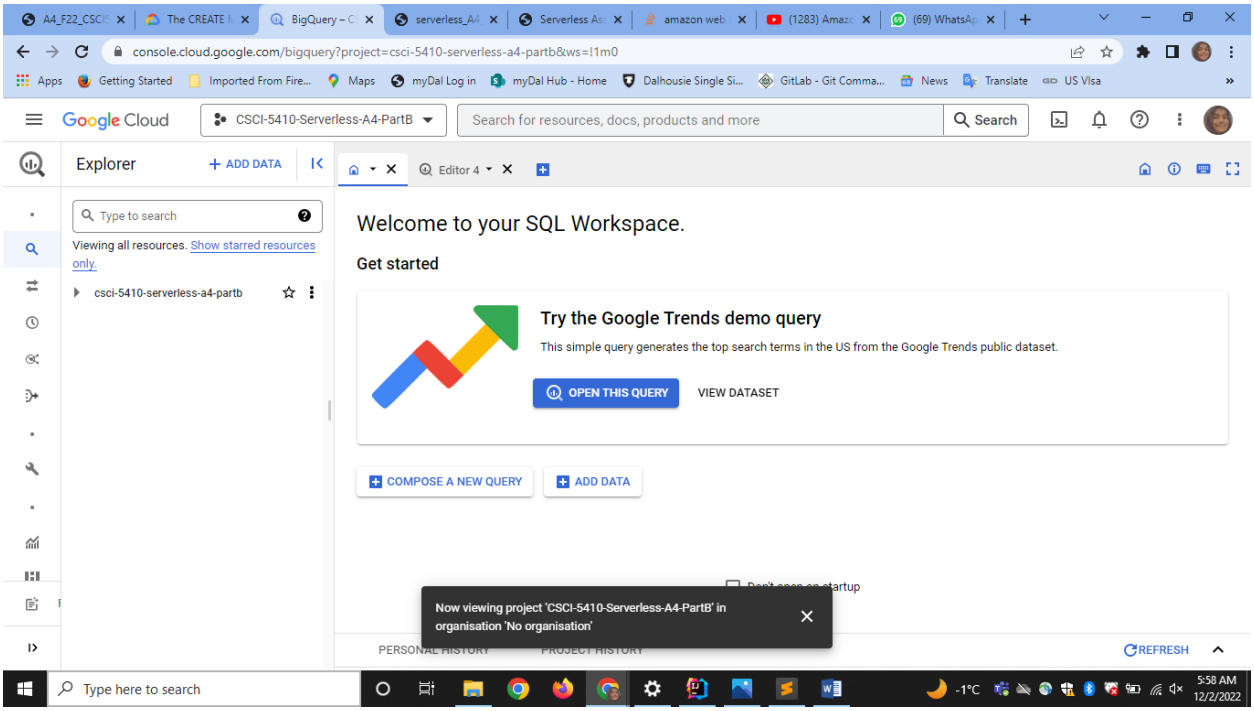


Fig 5: Created new project on GCP console

Step 4:

- Create training dataset as shown in the fig 6, 7. Fig 8 and 9 shows dataset has been created.

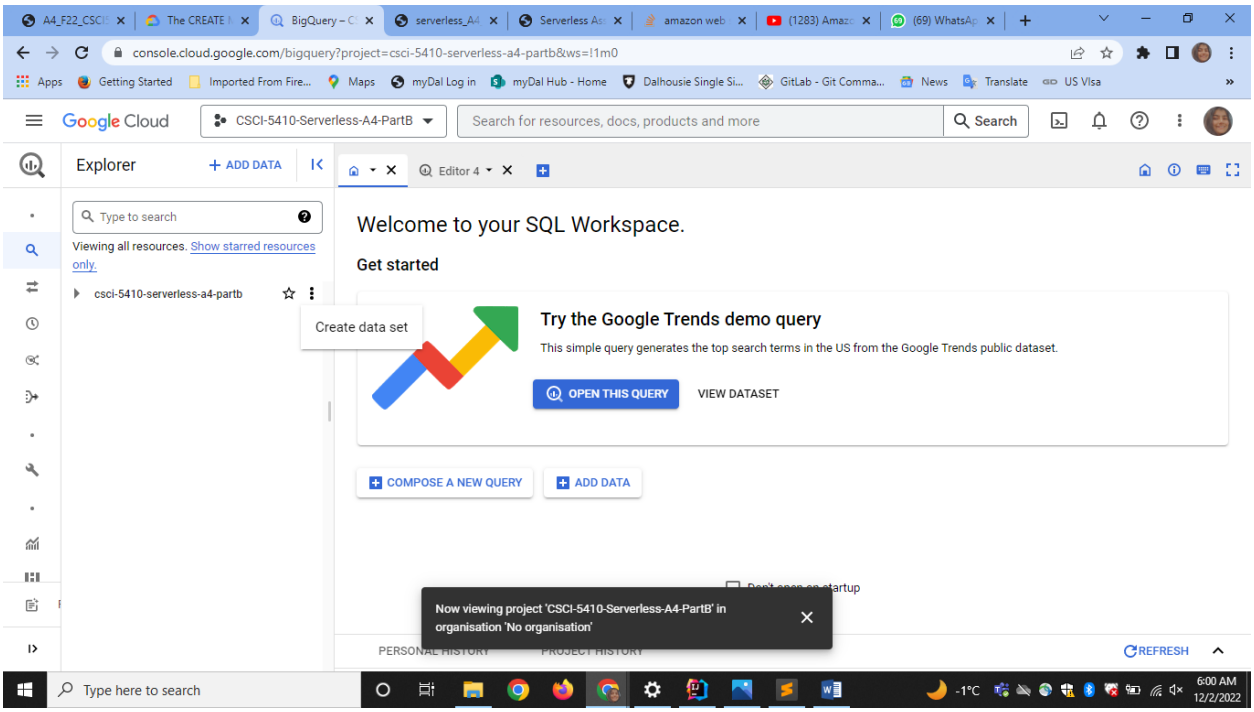


Fig 6: select "create data set" option

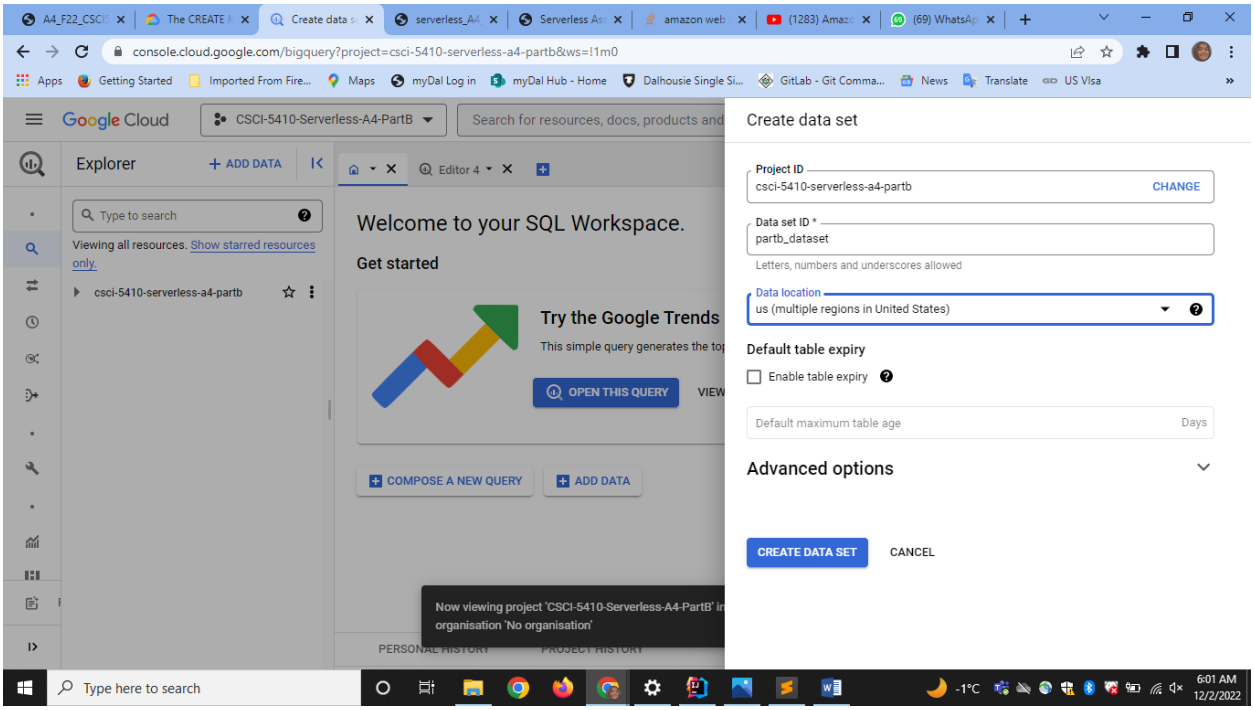


Fig 7: "create data set" form

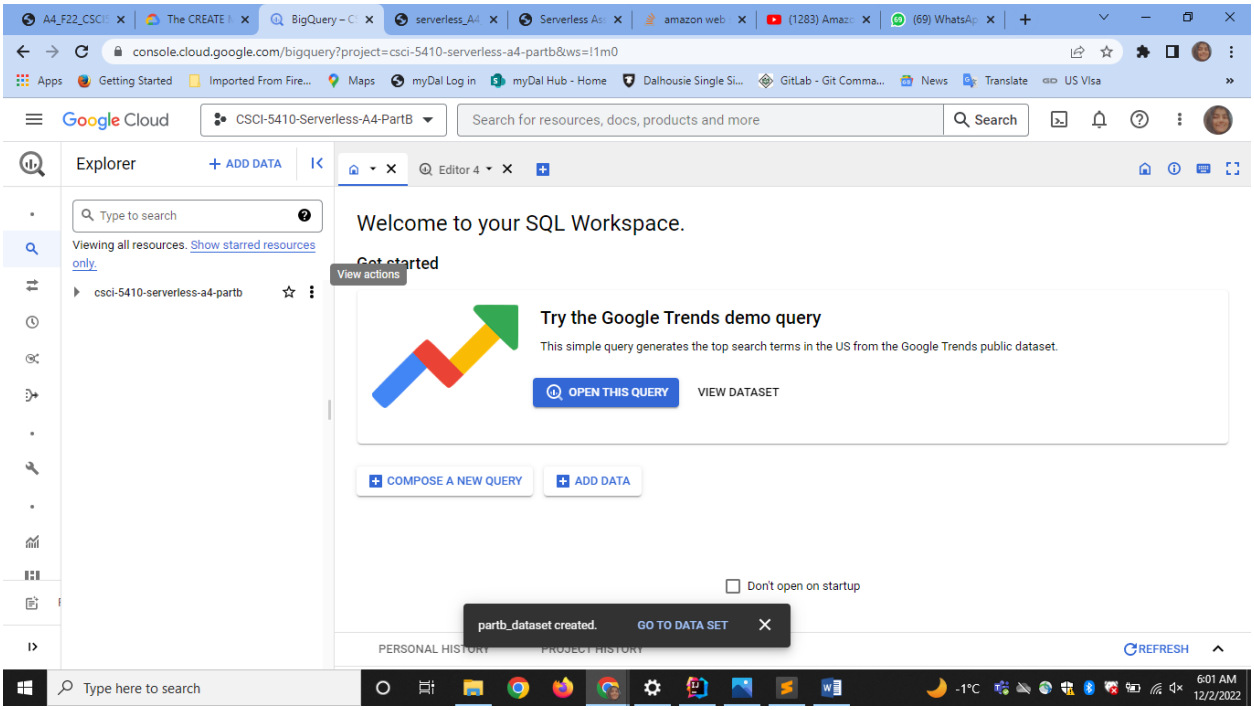


Fig 8: Dataset has been created

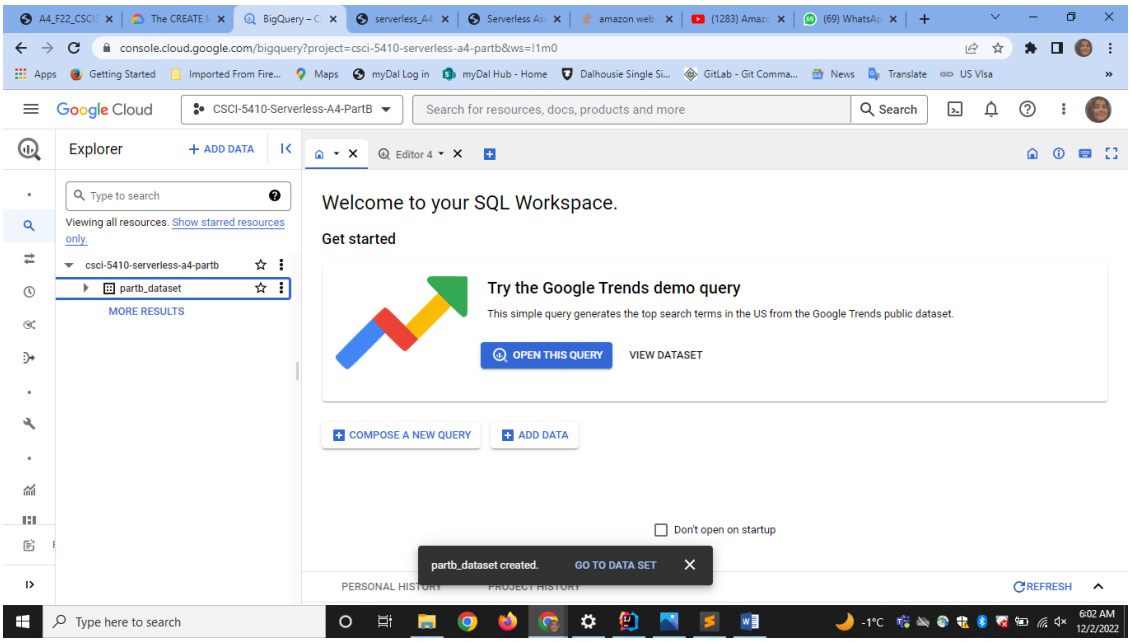


Fig 9: Dataset "partb_dataset" has been created

Step 5:

- Now we need to create table and upload data into the table.
- Fig 10 and 11 are the provided data set named"SDey_FTP_input" which is to be uploaded.

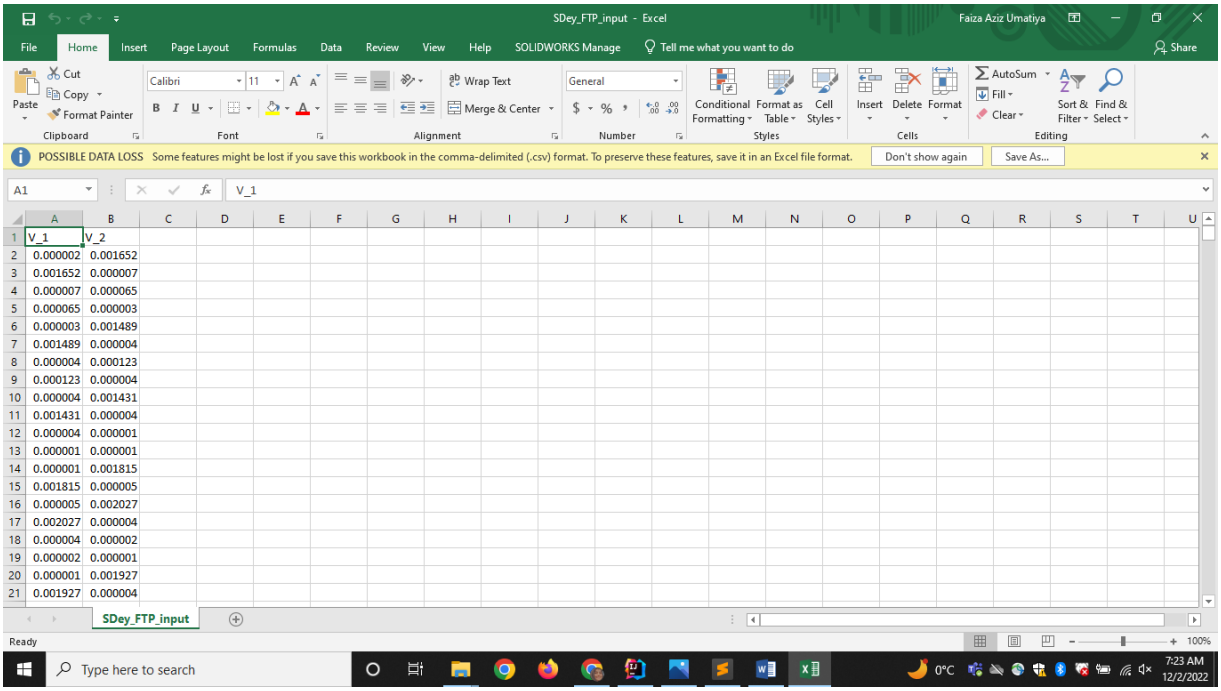


Fig 10: Dataset

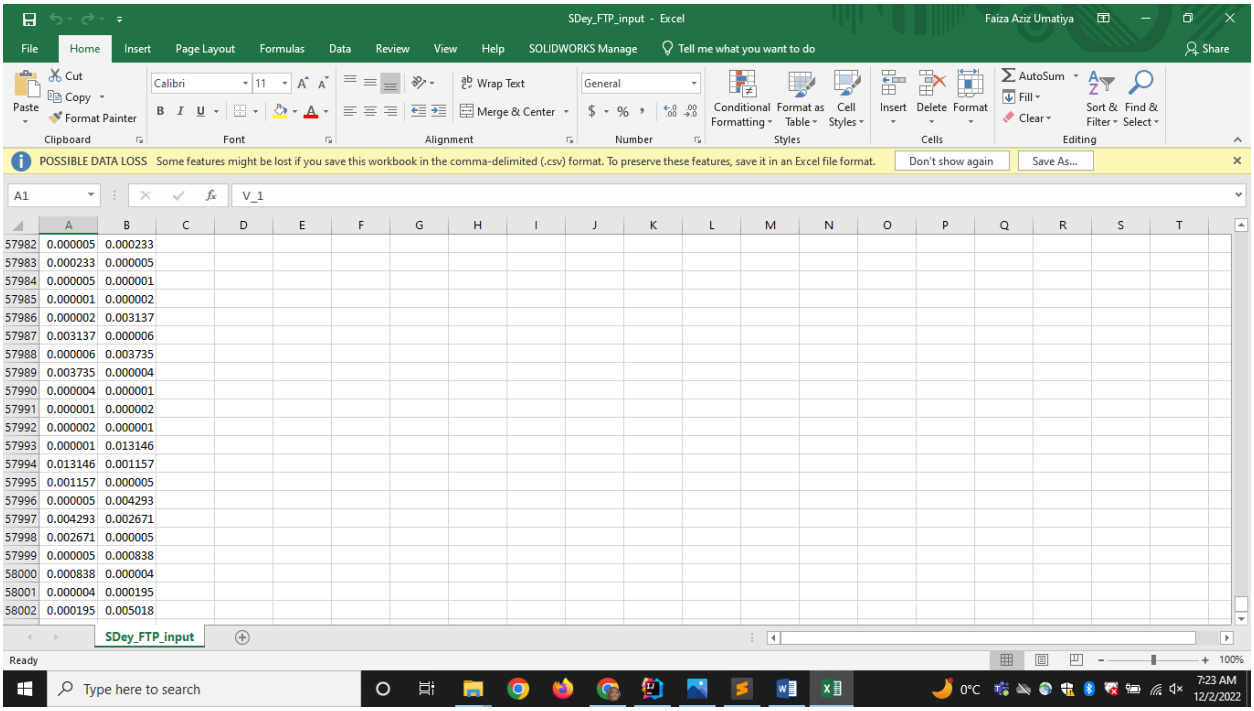


Fig 11: Dataset

Step 6:

- To add data, fill the details and upload file as shown in the figure 12, 13 and 14.
- Fig 15 shows "partb_dataset_table" has been created.
- Fig 16 shows the field values.

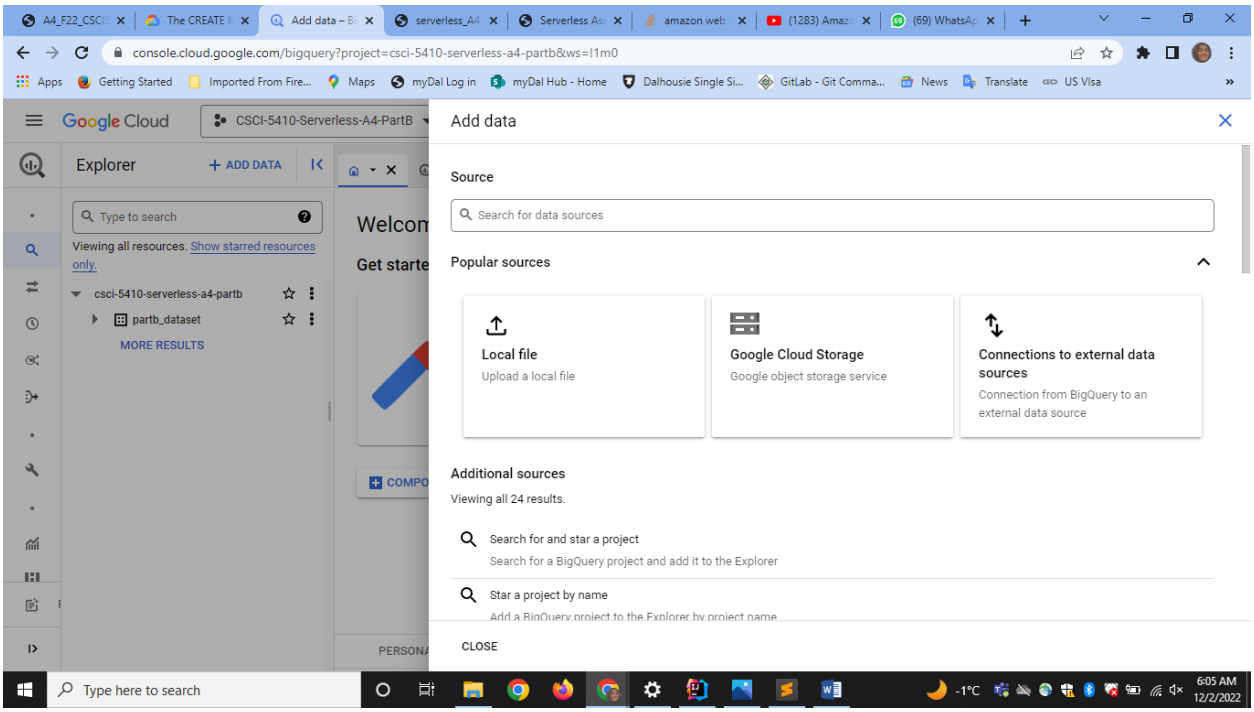


Fig 12: "Add data" form

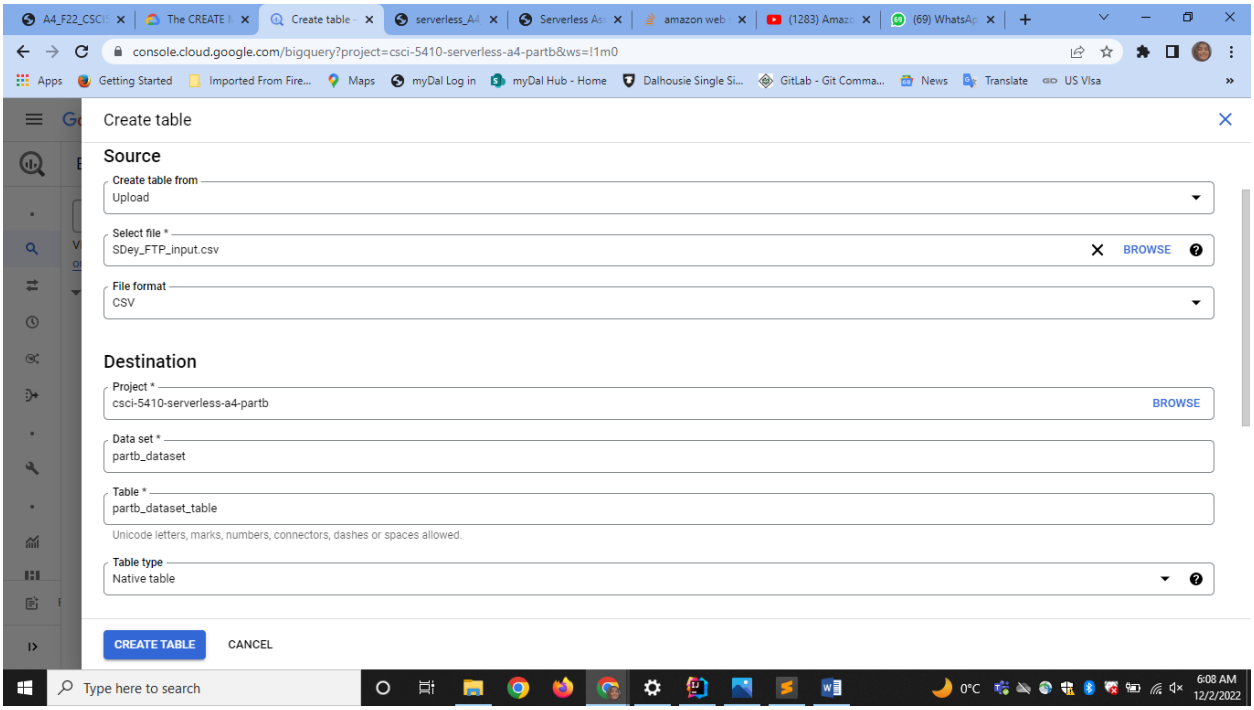


Fig 13: "Add data" form

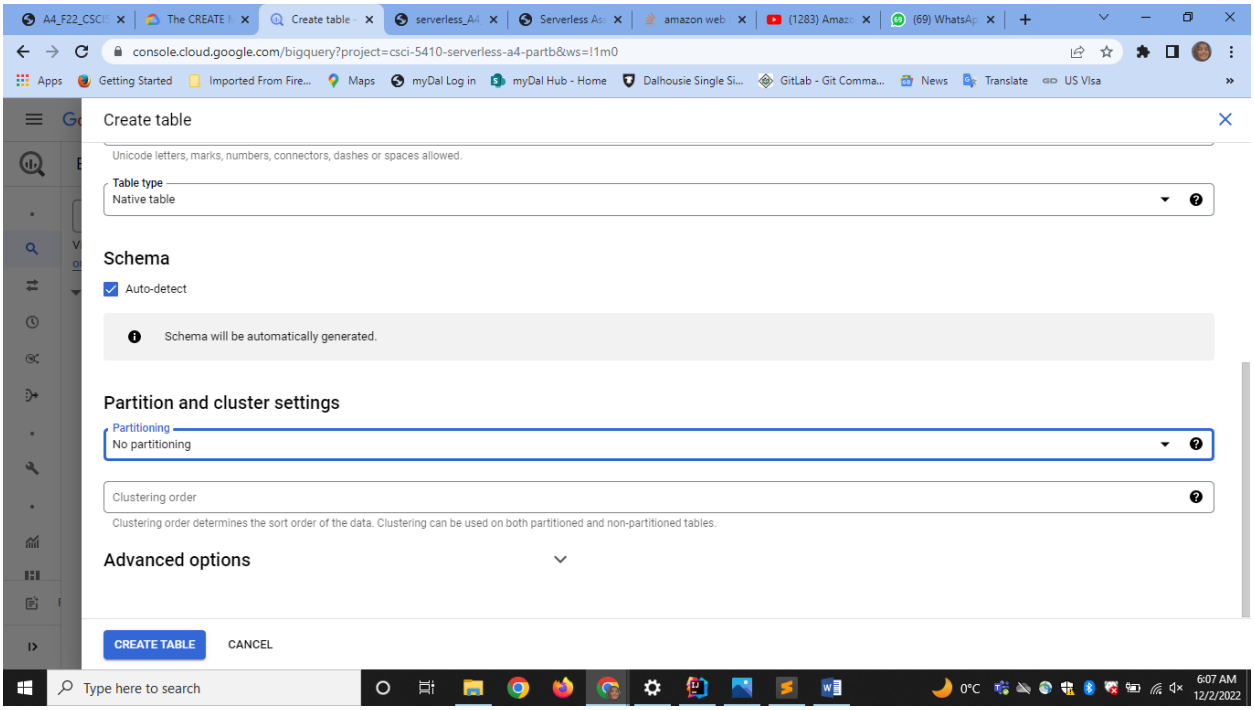


Fig 14: "Add data" form

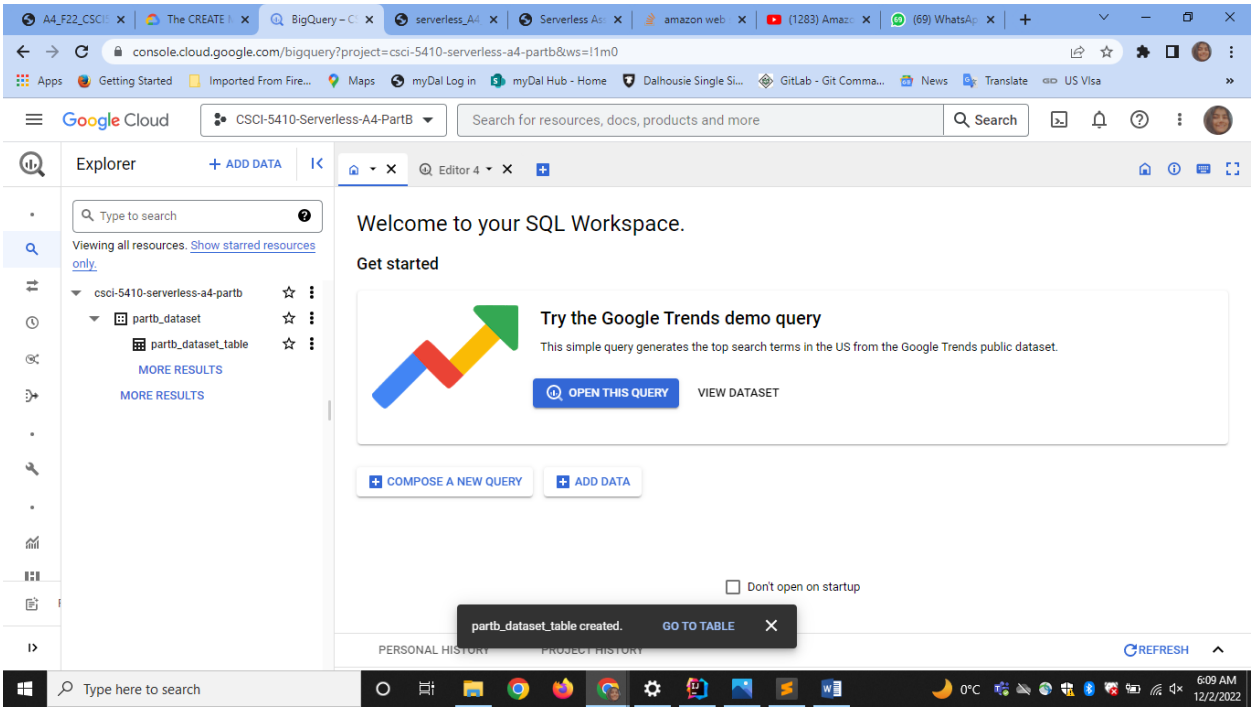


Fig 15: "parthb_dataset_table" has been created

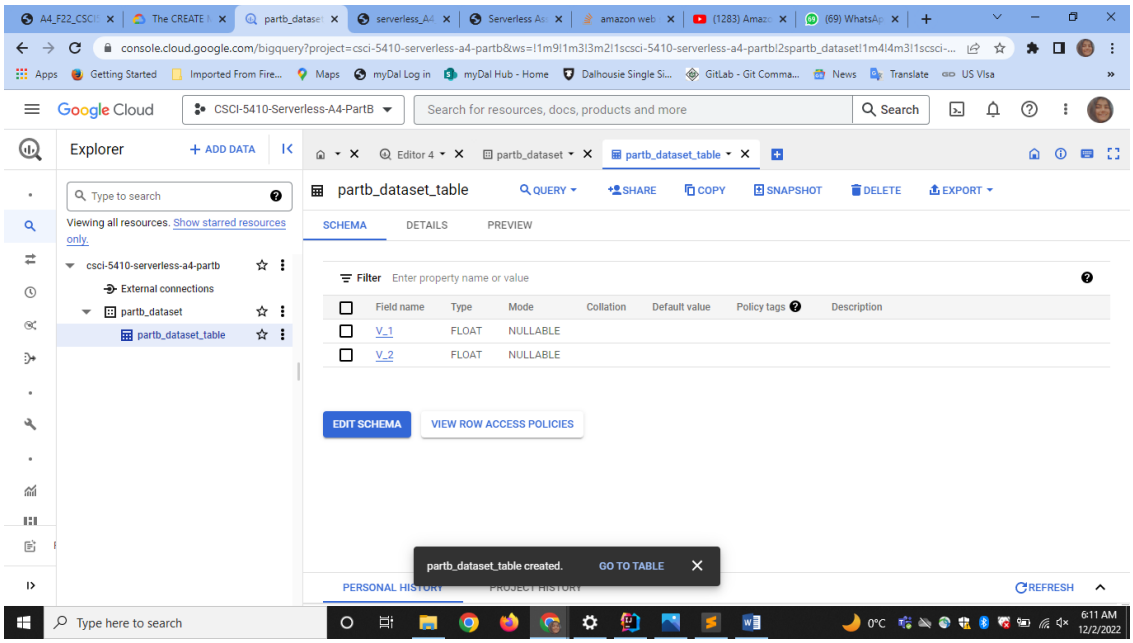


Fig 16: Field values of the "partb_dataset_table"

Step 7:

- Change the settings for the query by clicking on the "more" option under "partb_dataset_table" as shown in the fig 17. Then go to "Query settings" as shown in the fig. 17.
- Fill the "Query settings" details as shown in the fig 18, 19, 20.

- Keep the location same as the “partb_dataset” which is “US”.

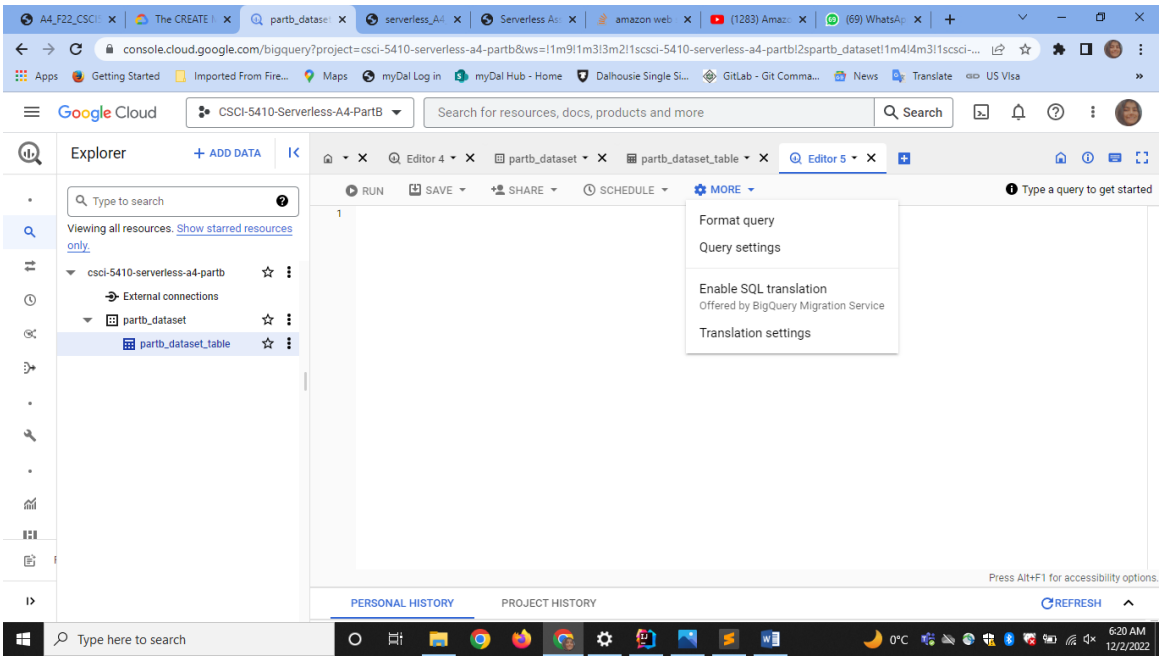


Fig 17: “Query setting” option

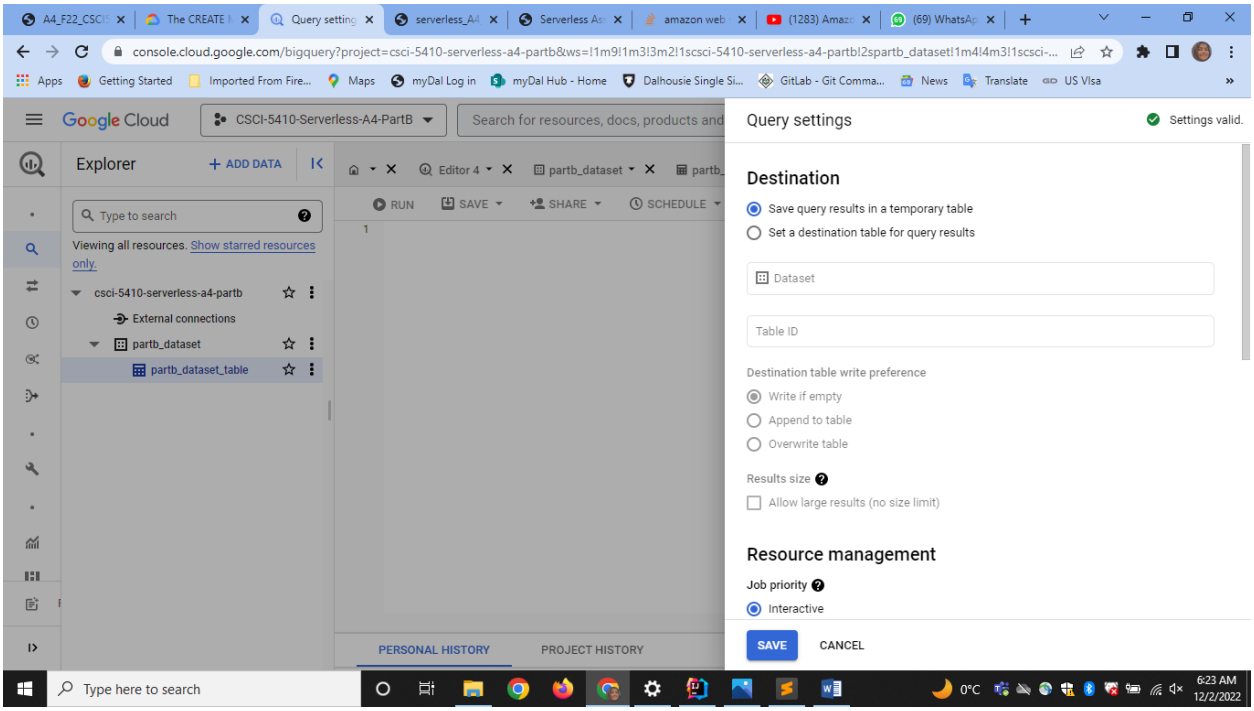


Fig 18: “Query settings” form

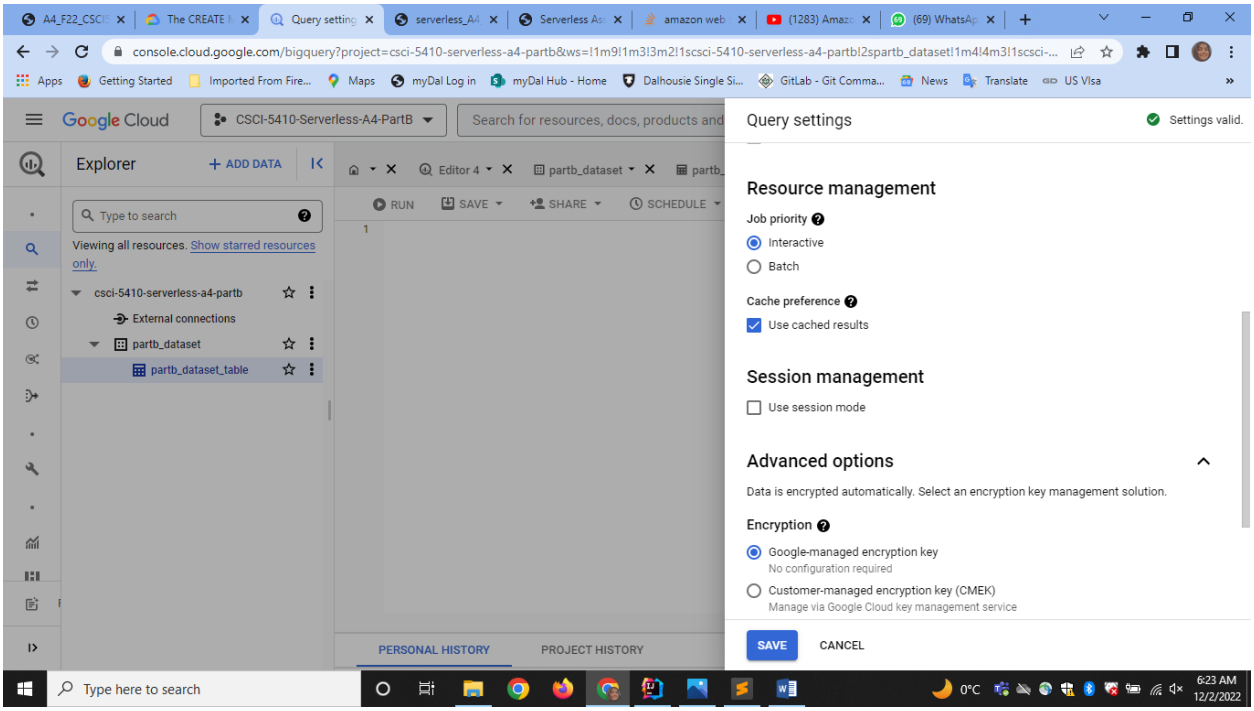


Fig 19: "Query settings" form

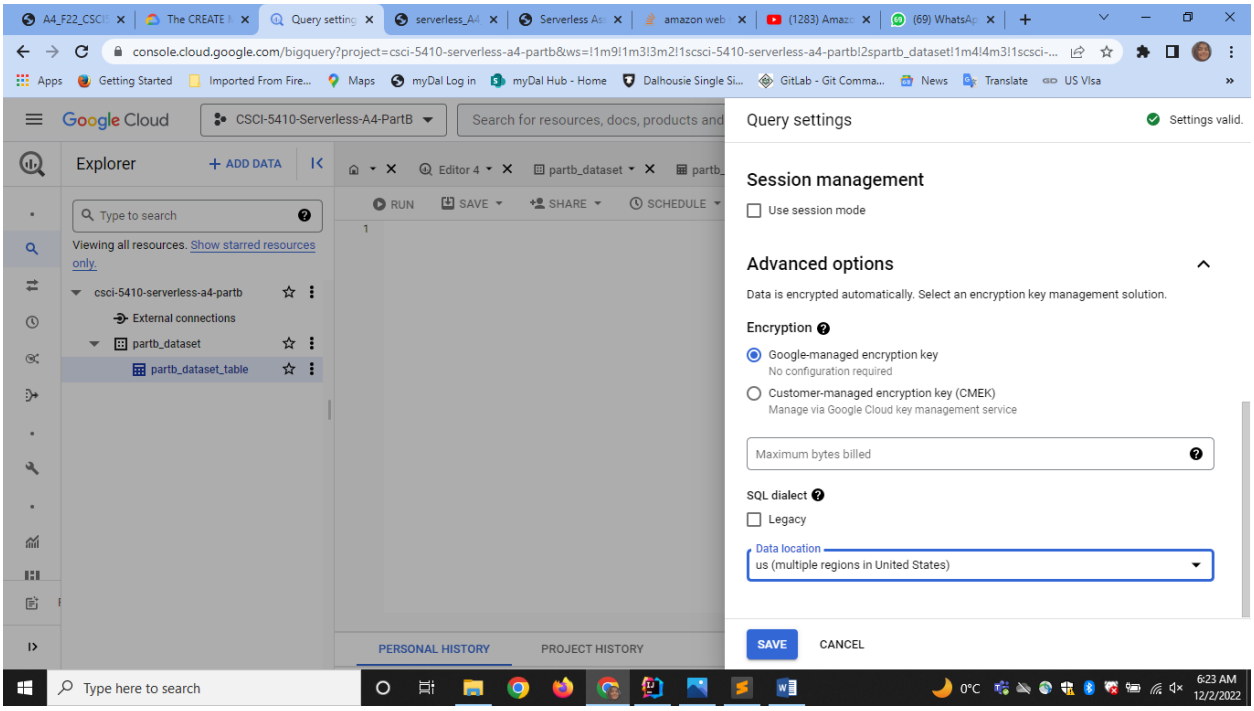


Fig 20: "Query settings" form

Step 8:

- I wrote a query to train the model as shown in the fig 21. The query creates a k-means model with 5 clusters using the default “DISTANCE_TYPE” value of “EUCLIDEAN_DISTANCE” [1].
- Fig 22 shows query running with the elapsed time.
- Fig 23 shows that query is executed.
- Fig 24 and 25 provides the model details which is been created.
- Fig 26, 27, 28, 29, 30 and 31 gives the graphical view, table view of the results.
- Fig 32 represents evaluation details of the “partb_model”.
- Fig 33 represents job details of the “partb_model”.

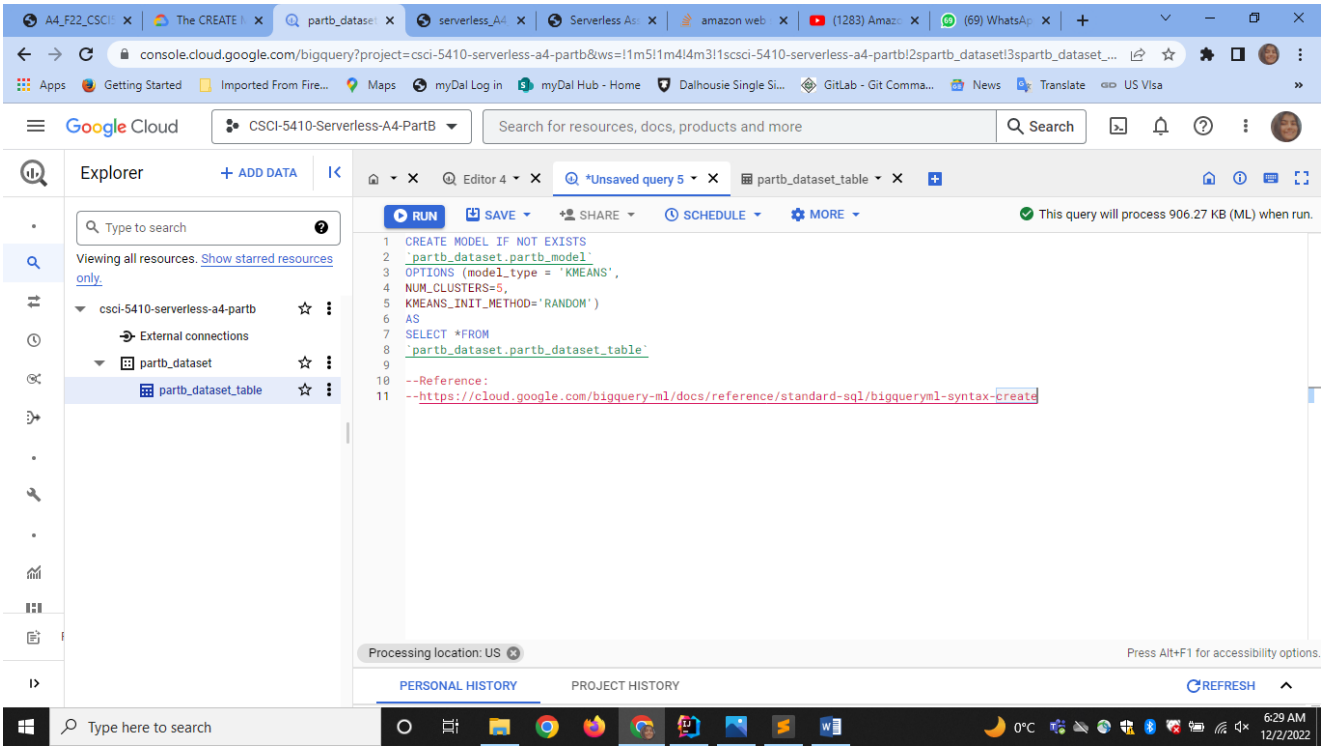


Fig 21: Query to train the model

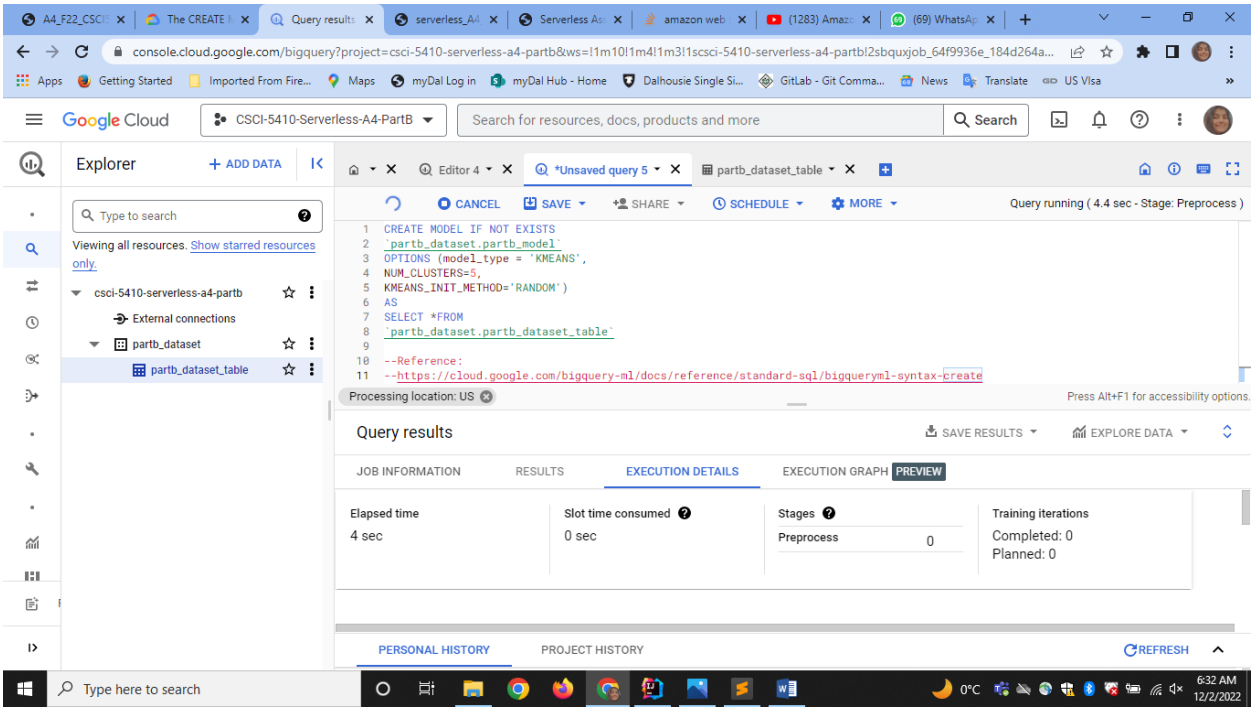


Fig 22: Query running

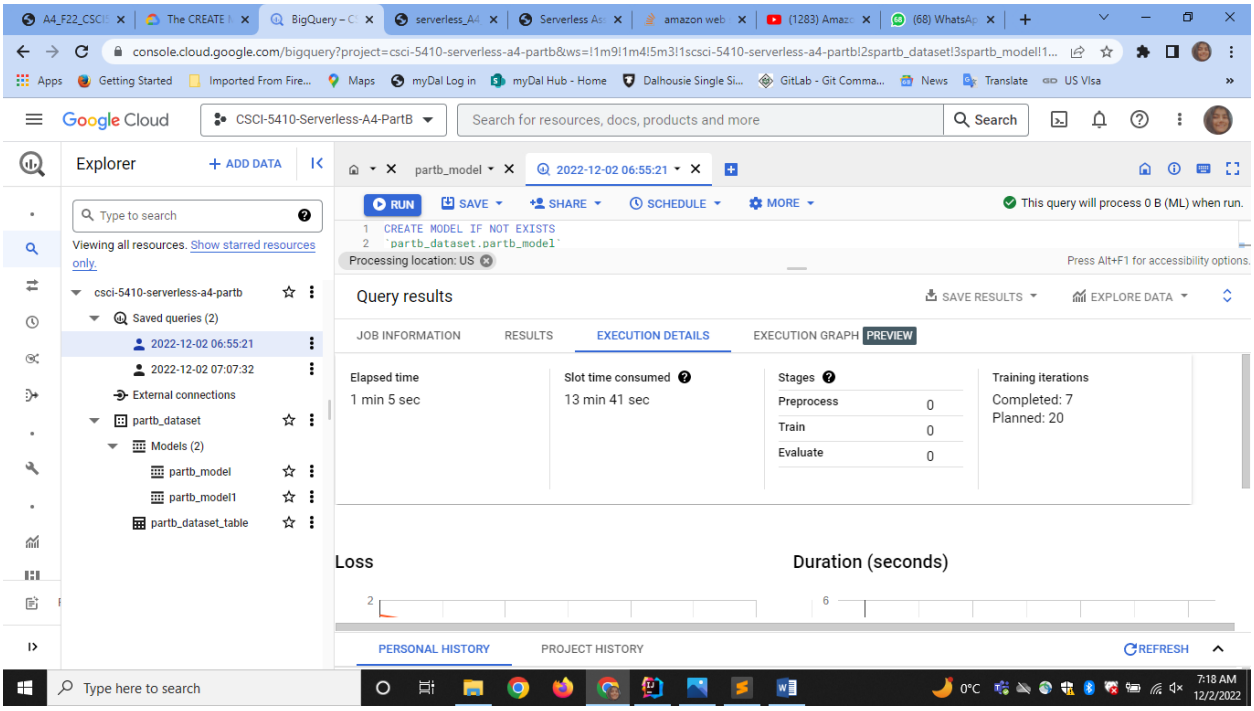


Fig 23: Query results

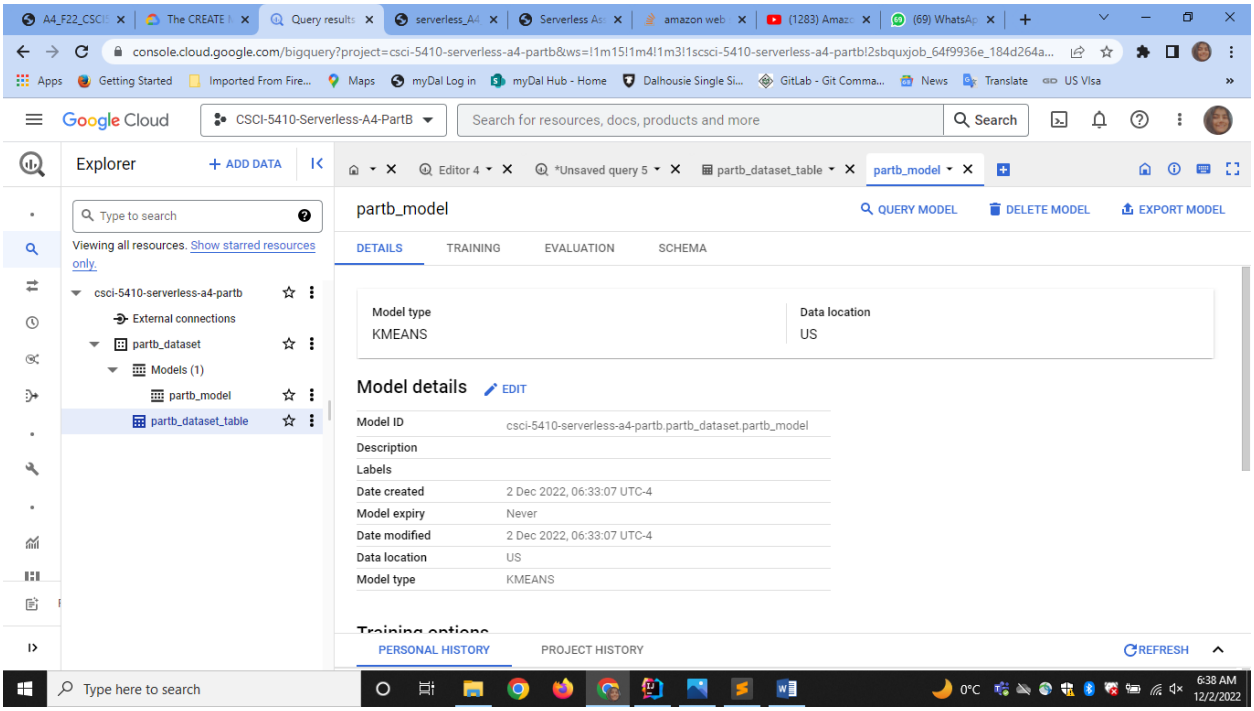


Fig 24: Model details for "partb_model"

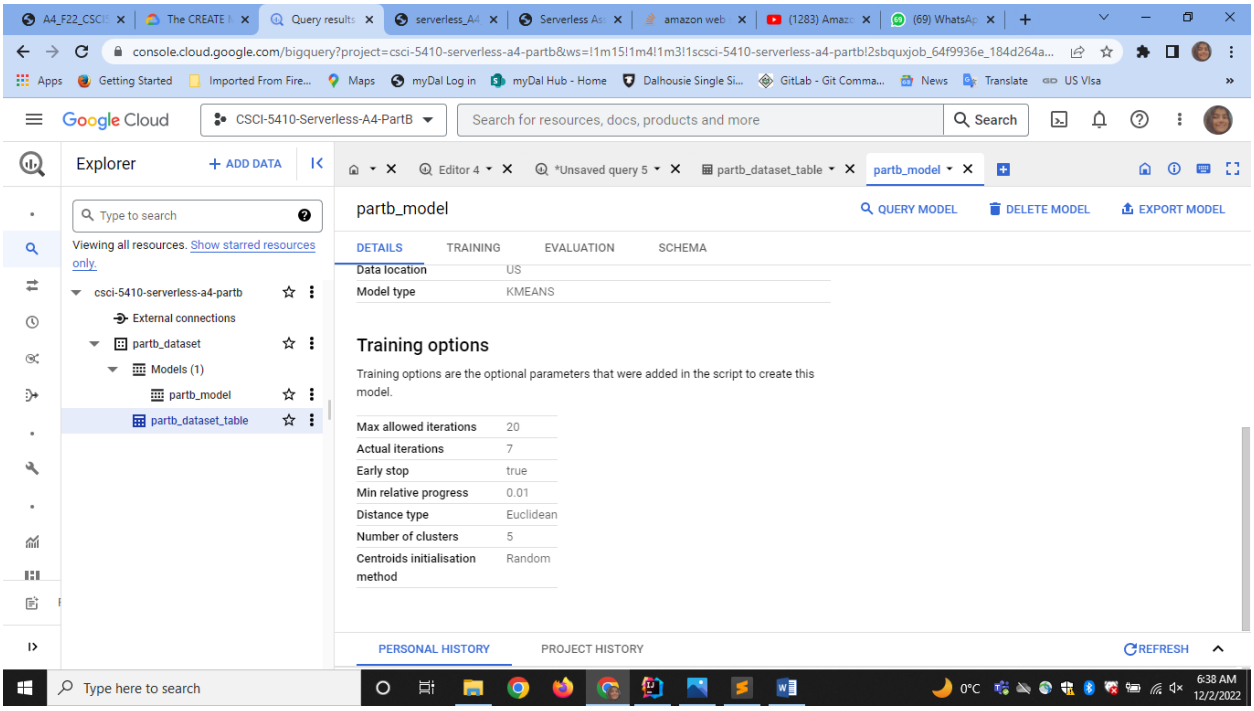


Fig 25: Model details for "partb_model"

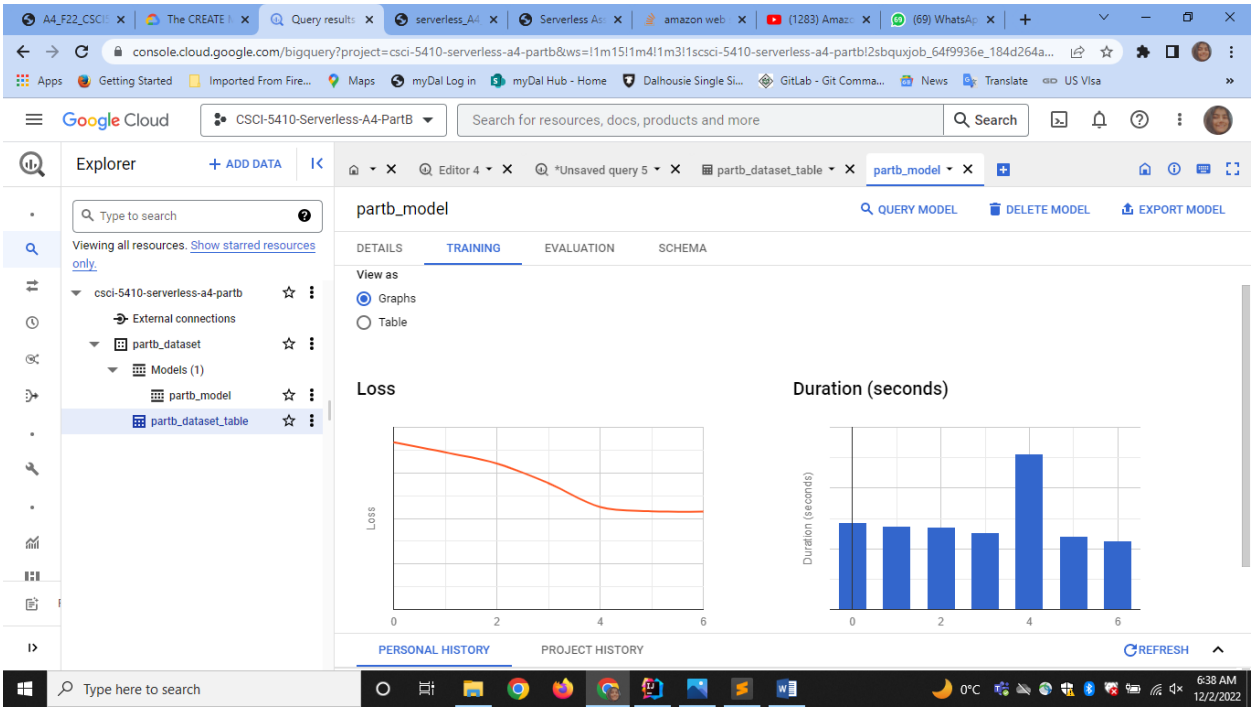


Fig 26: Training details for "partb_model" – Graph view

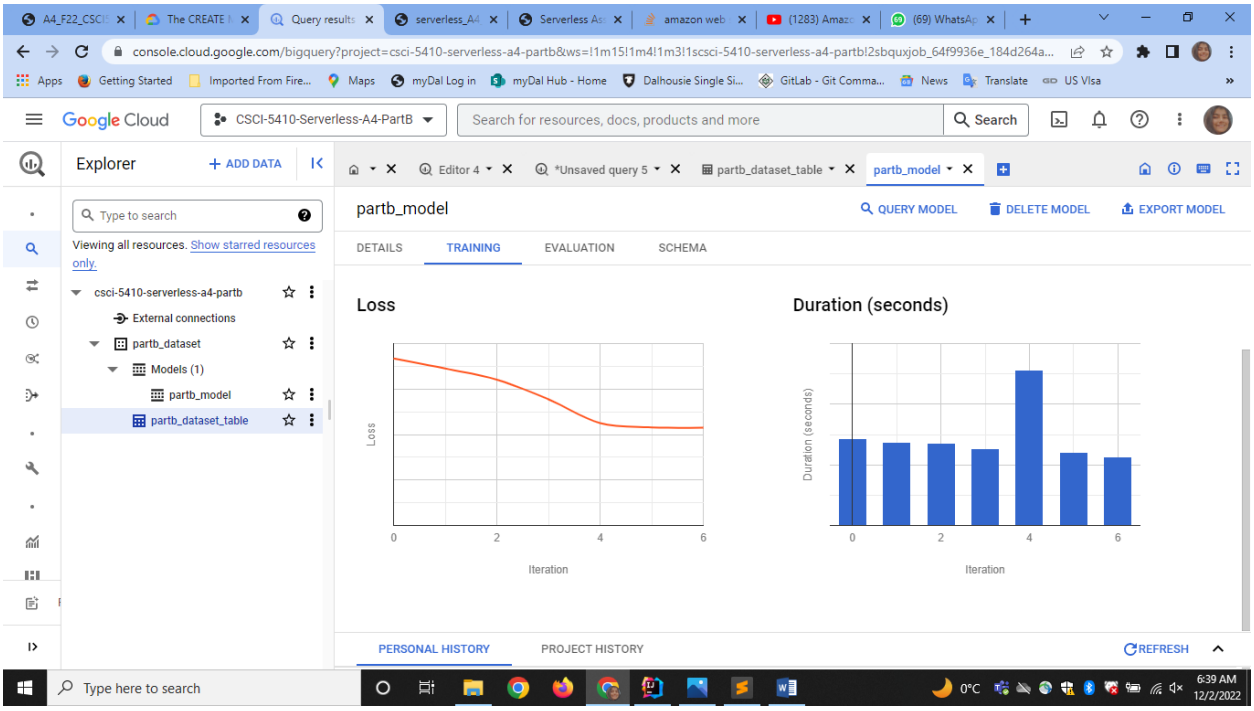


Fig 27: Training details for "partb_model" – Graph view

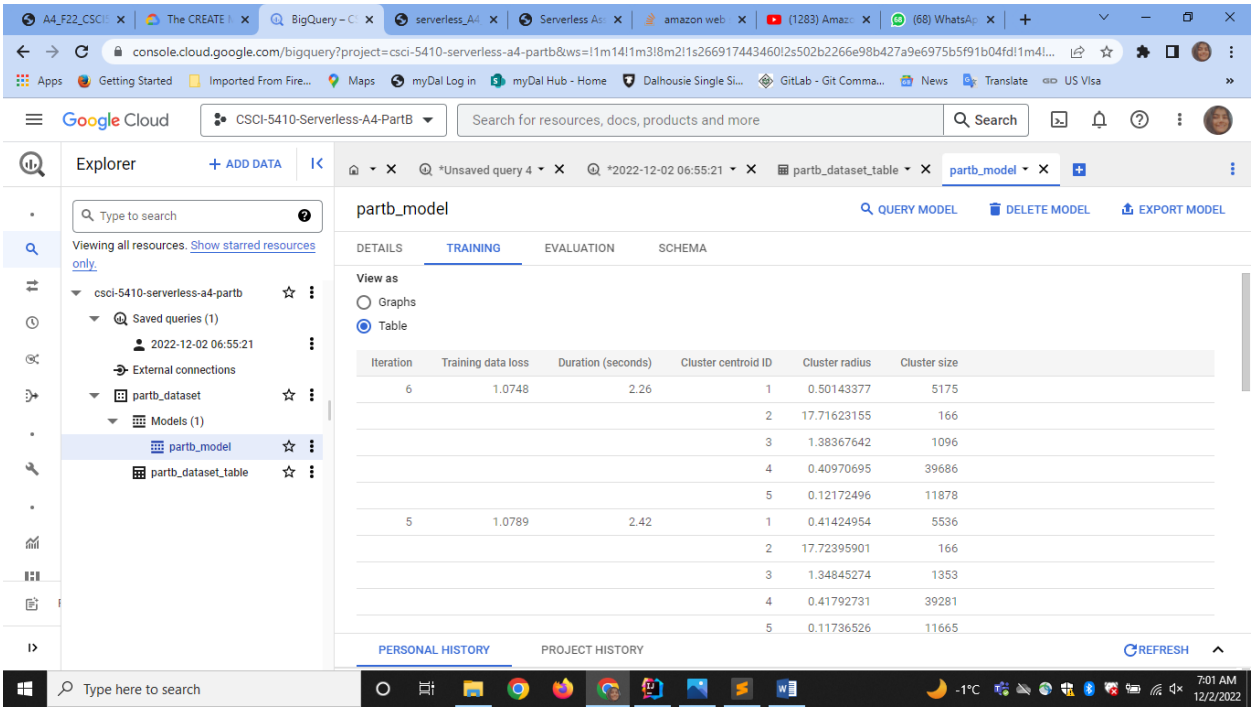


Fig 28: Training details for "partb_model" – table view

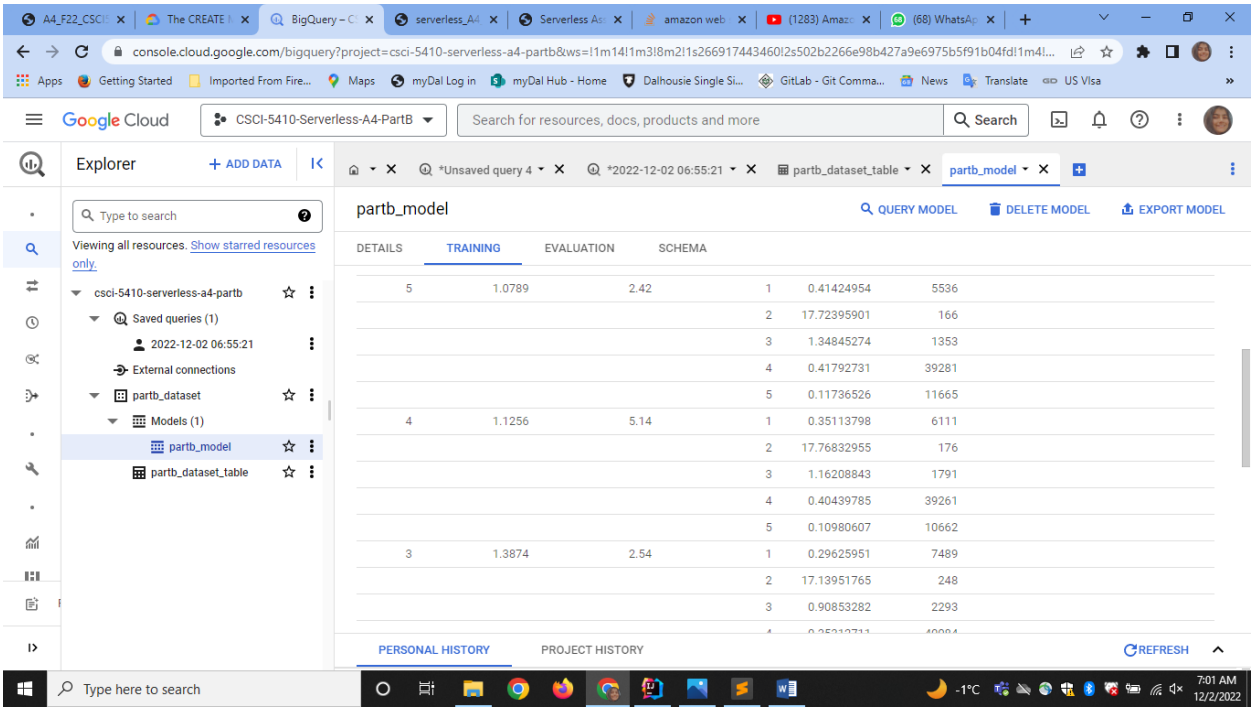


Fig 29: Training details for "partb_model" – table view

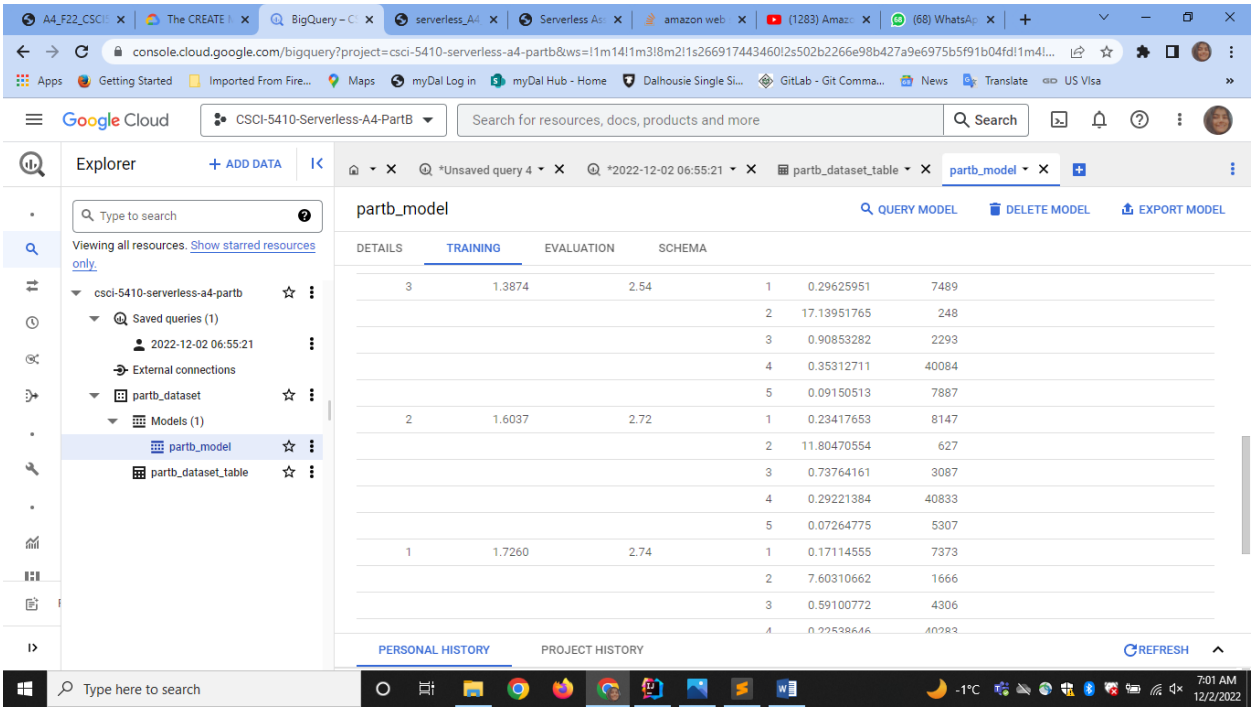


Fig 30: Training details for "partb_model" – table view

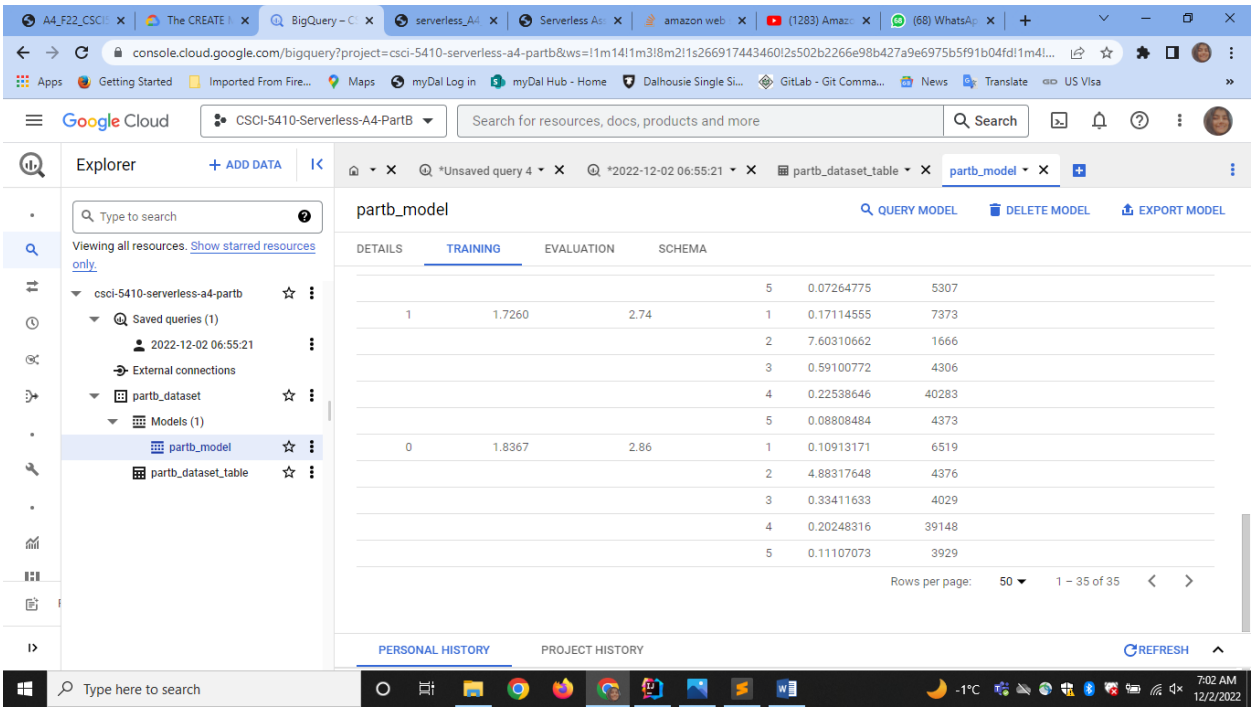


Fig 31: Training details for "partb_model" – table view

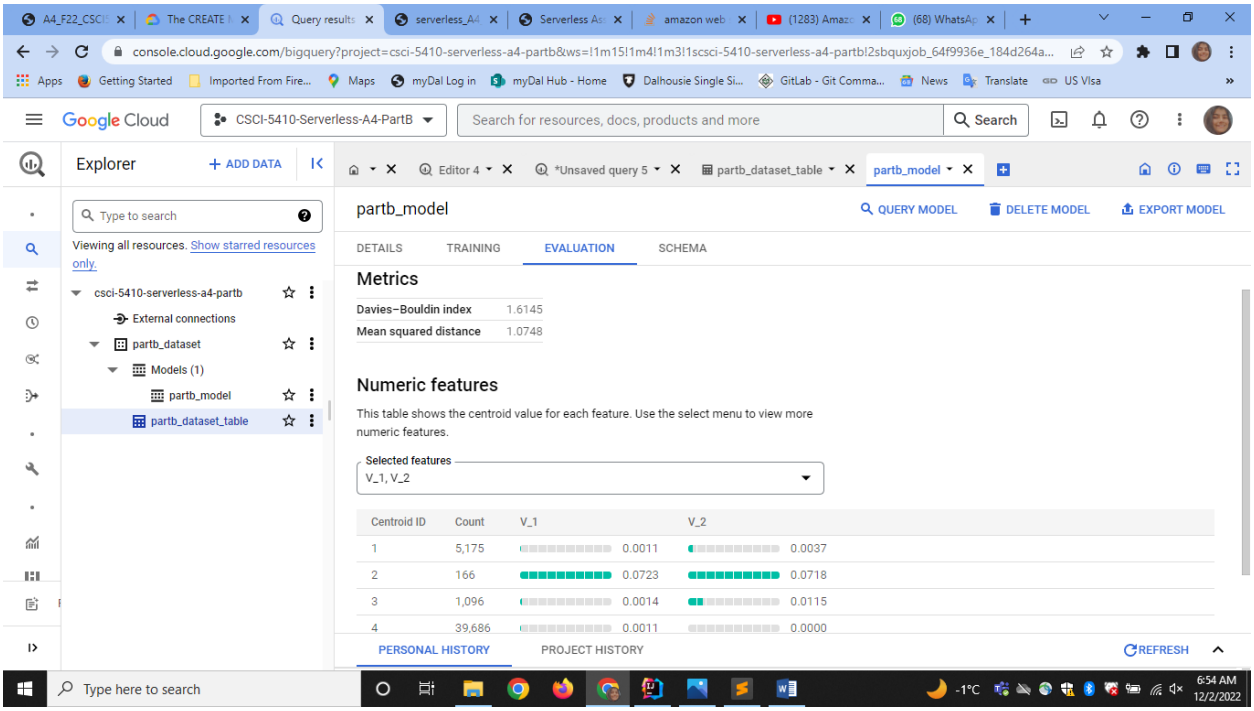


Fig 32: Evaluation details for "partb_model"

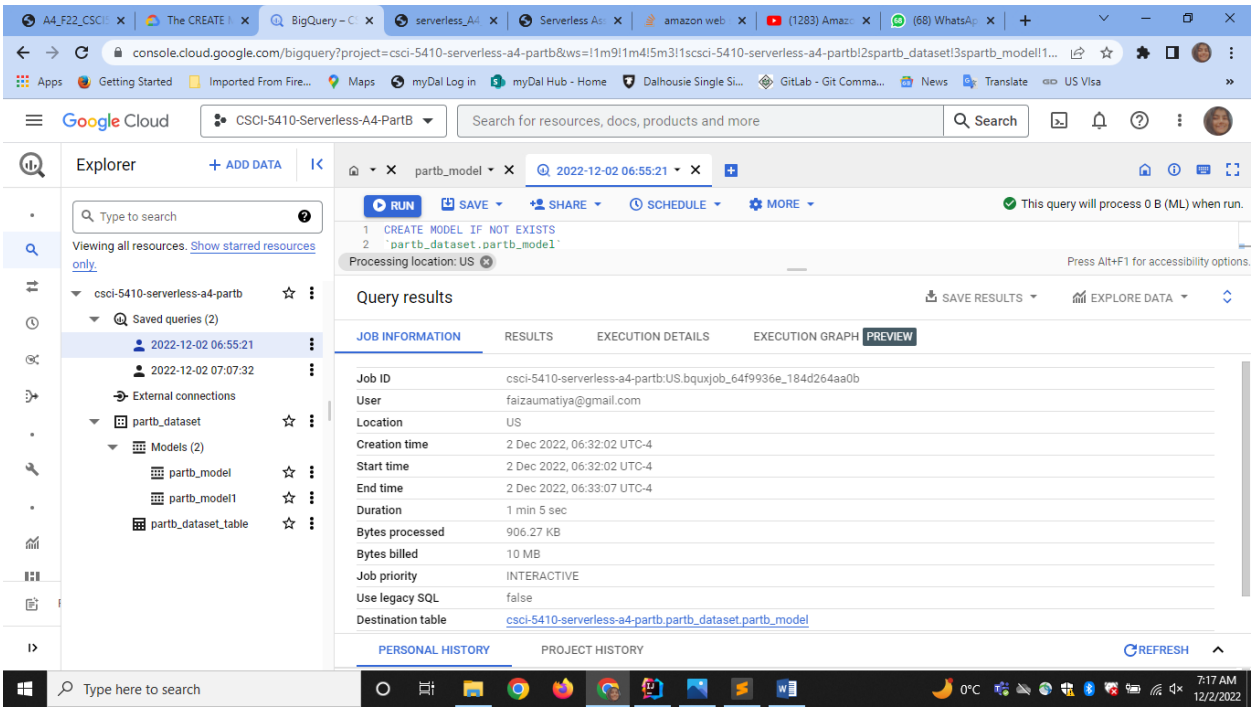


Fig 33: Job details for "partb_model"

EXTRA:

- We can also check the training model training statics by writing the query as shown in the fig 33.1.
- Fig 33.2, 33.3, 33.4 and 33.5 shows the query results for training model.

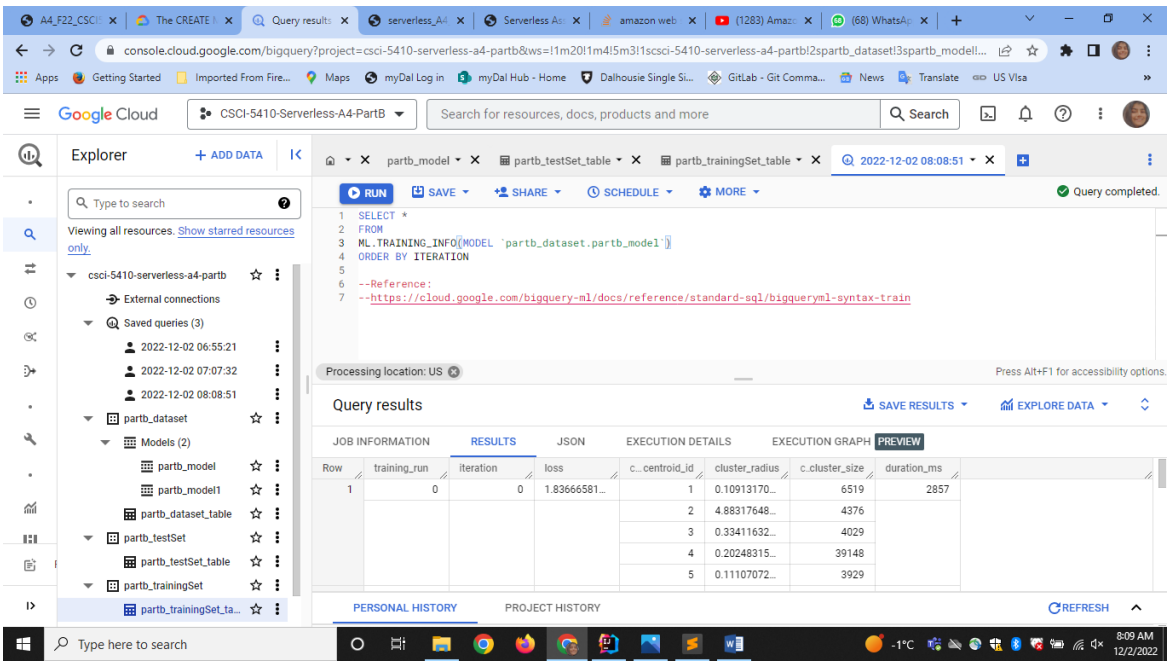


Fig 33.1: Query to check model training

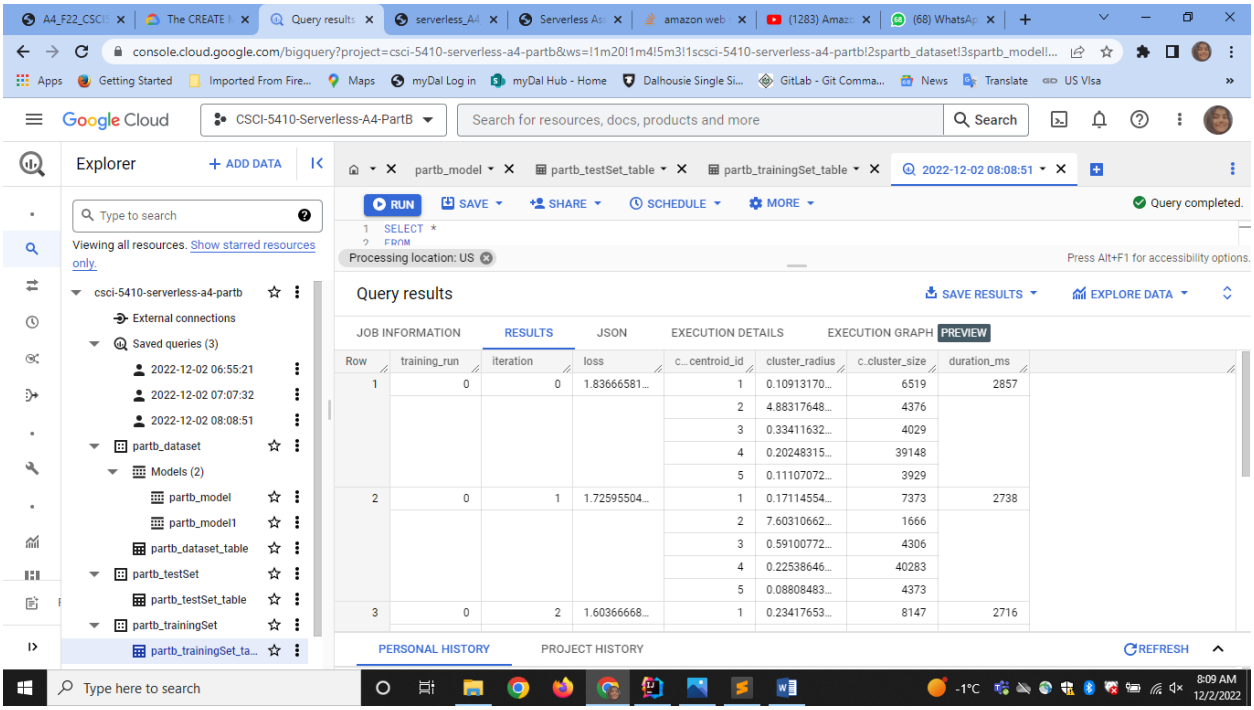


Fig 33.2: Query results for training statistics

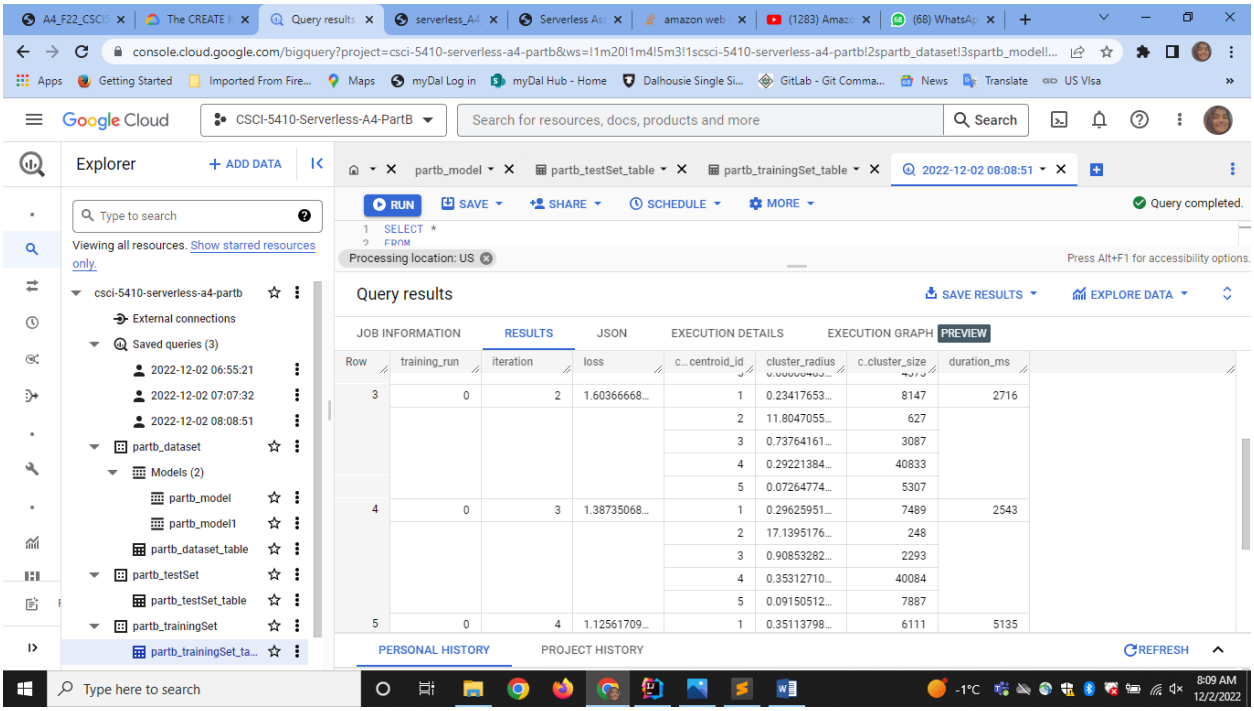


Fig 33.3: Query results for training statistics

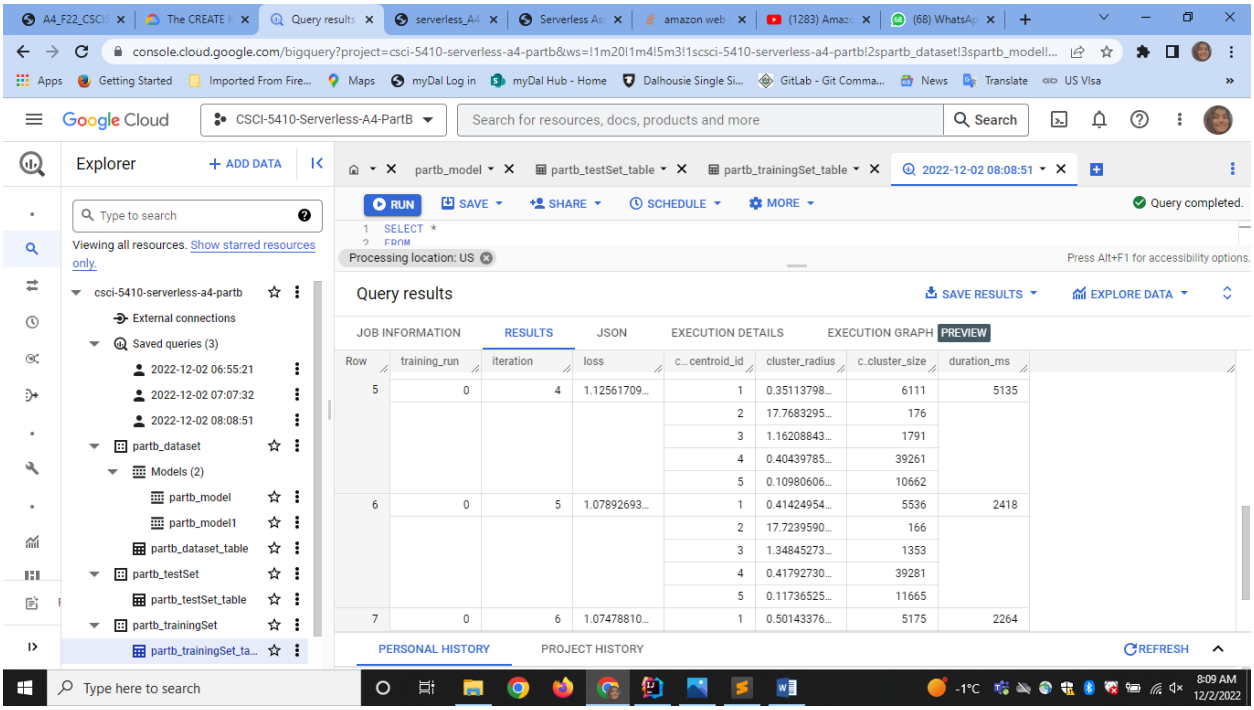


Fig 33.4: Query results for training statistics

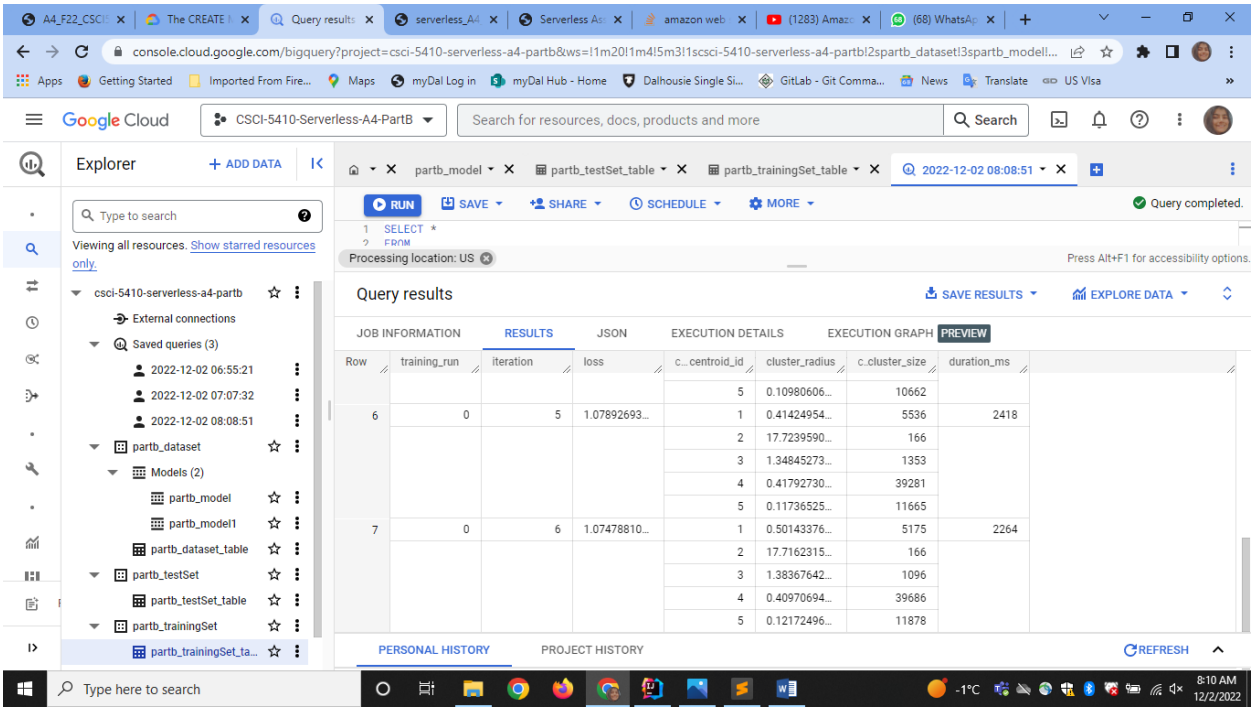


Fig 33.5: Query results for training statistics

Alternative query:

Follow the same step as Step 7:

Step 8:

- Change the settings for the query by clicking on the “more” option under “partb_dataset_table” as shown in the fig 34. Then go to “Query settings” as shown in the fig. 34.
- Fill the “Query settings” details as shown in the fig 35, 36, 37 and 38.
- Keep the location same as the “partb_dataset” which is “US”.

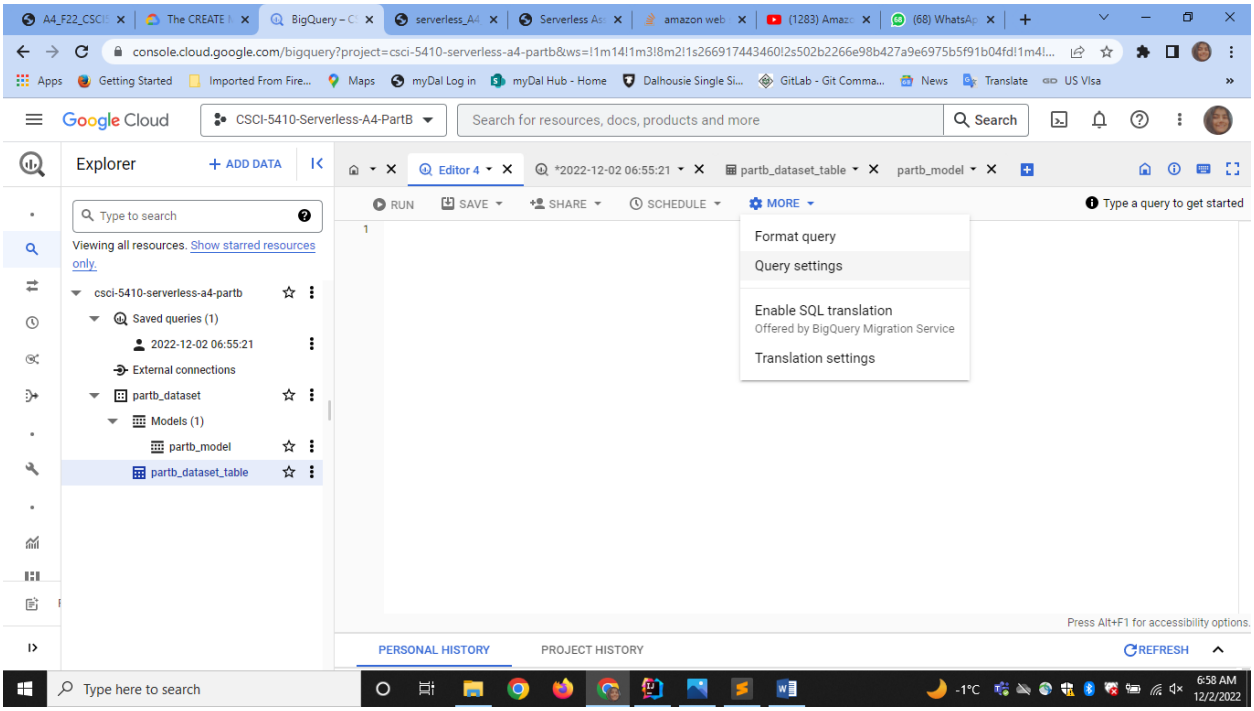


Fig 34: "Query setting" option

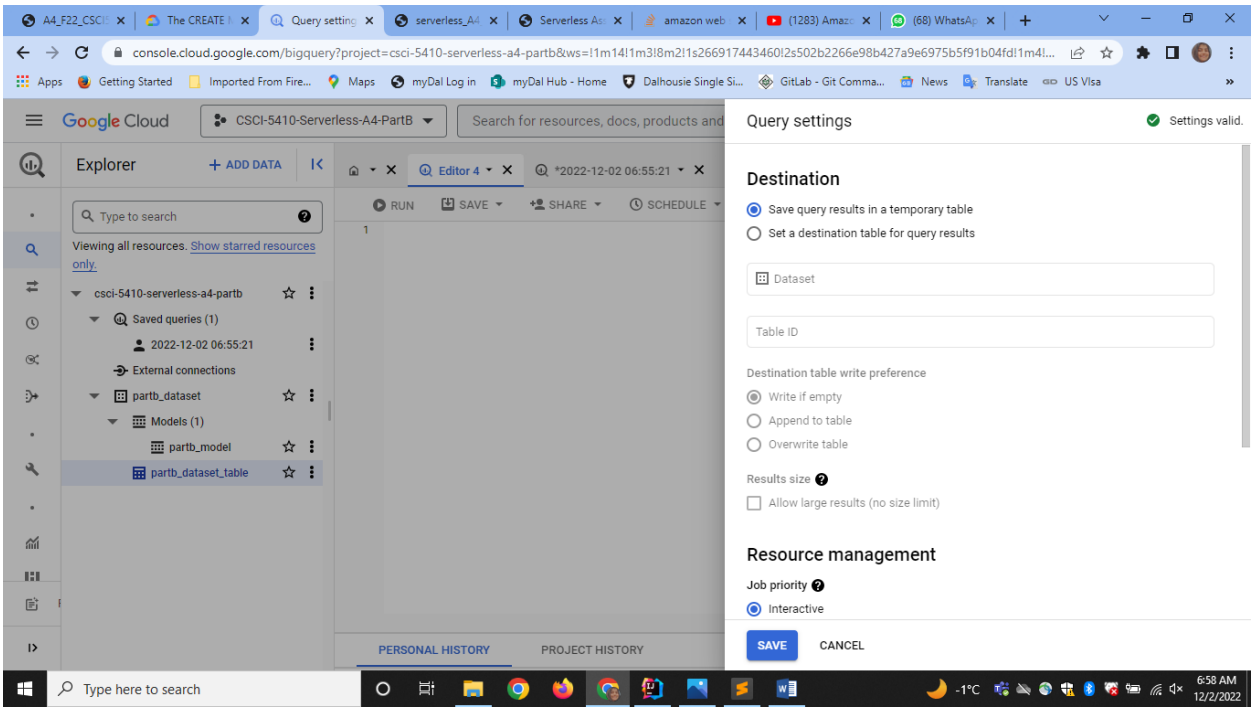


Fig 35: "Query settings" form

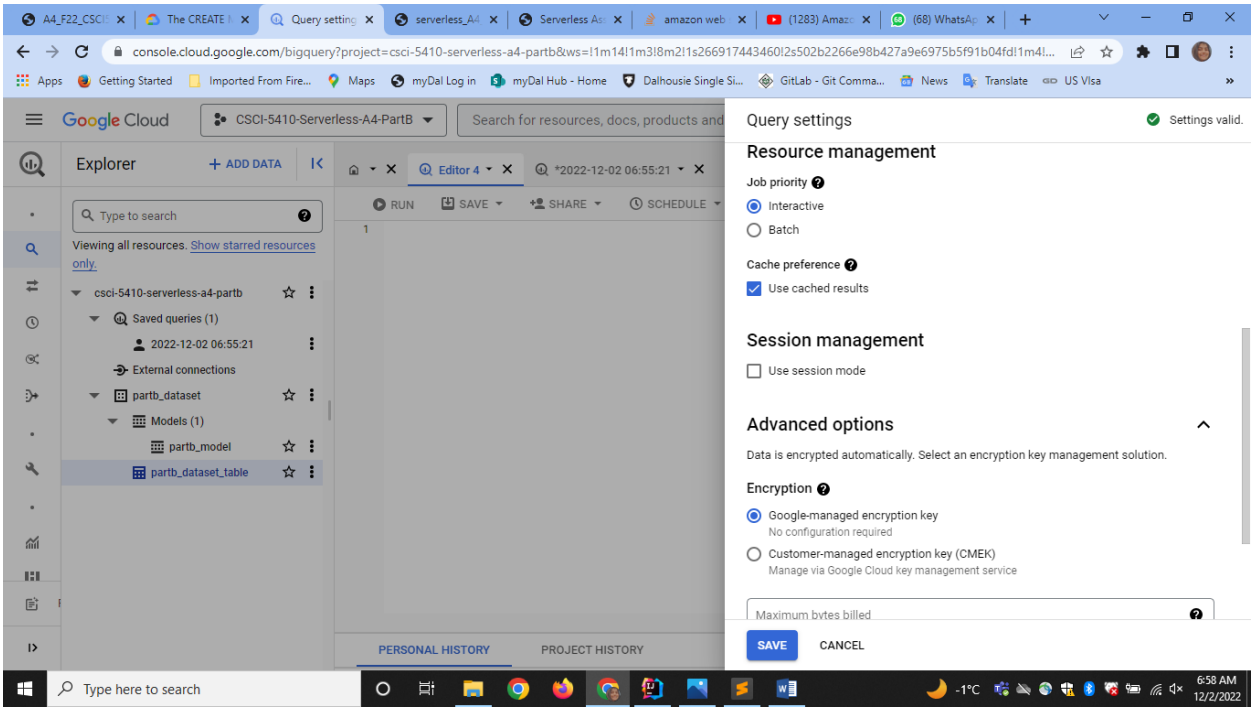


Fig 36: "Query settings" form

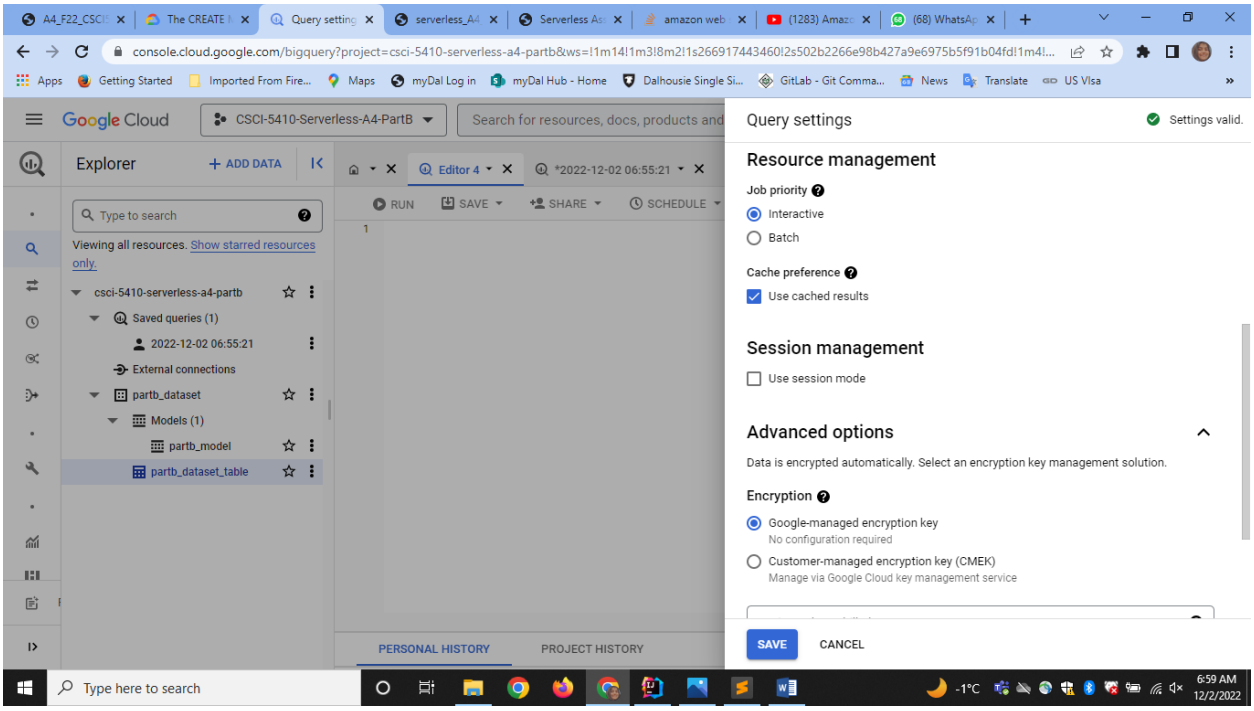


Fig 37: "Query settings" form

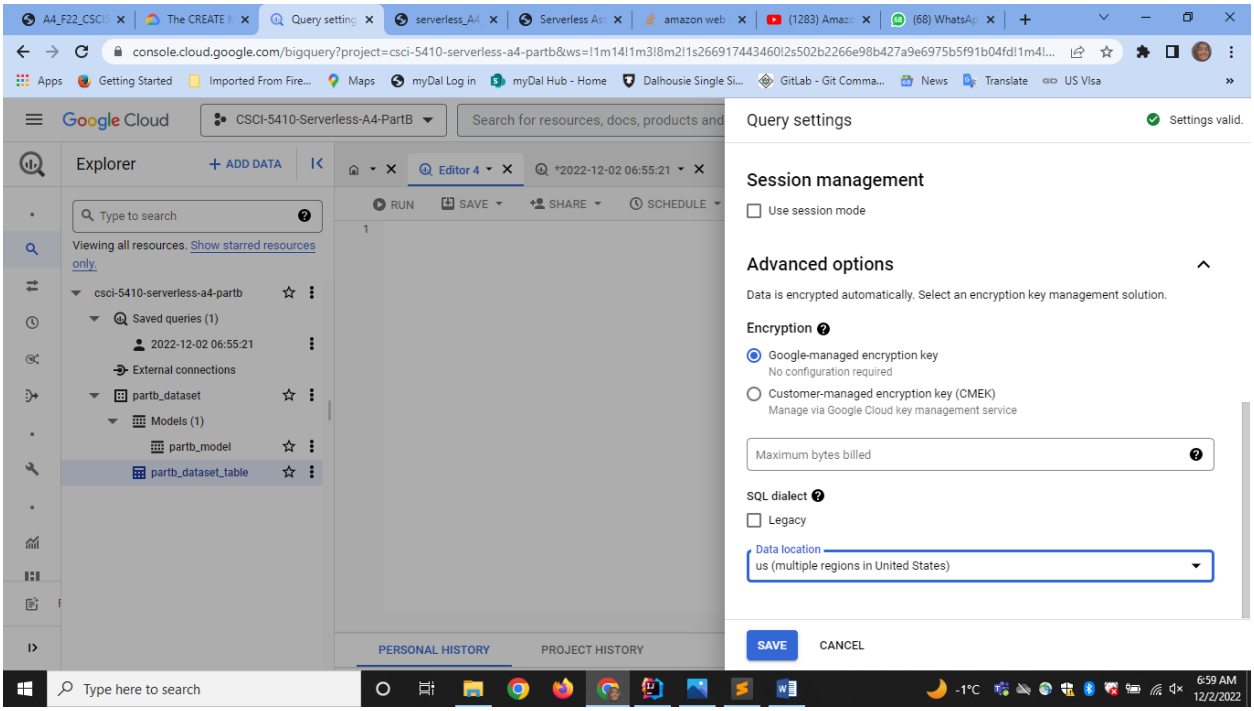


Fig 38: "Query settings" form

Step 9:

- I wrote a query to train the model as shown in the fig 39. The query creates a k-means model with 11 clusters using the "DISTANCE_TYPE" value of "EUCLIDEAN_DISTANCE" [1].
- Fig 40 shows query running with the elapsed time.
- Fig 41 shows that query is executed.
- Fig 42 and 43 provides the model details which is been created.
- Fig 44, 45, 46, 47, 48, 49 and 50 gives the graphical view, table view of the results.
- Fig 51 and 52 represents evaluation details of the "partb_model1".
- Fig 53 represents job details of the "partb_model1".

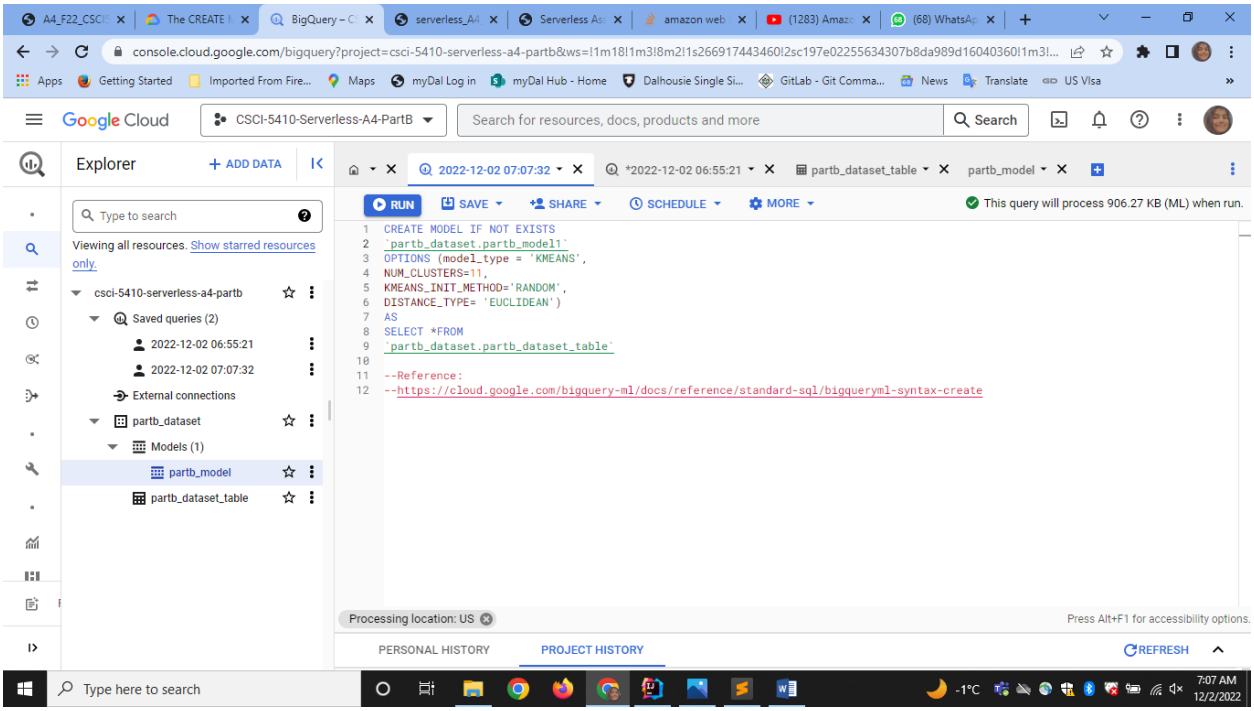


Fig 39: Query to train the model

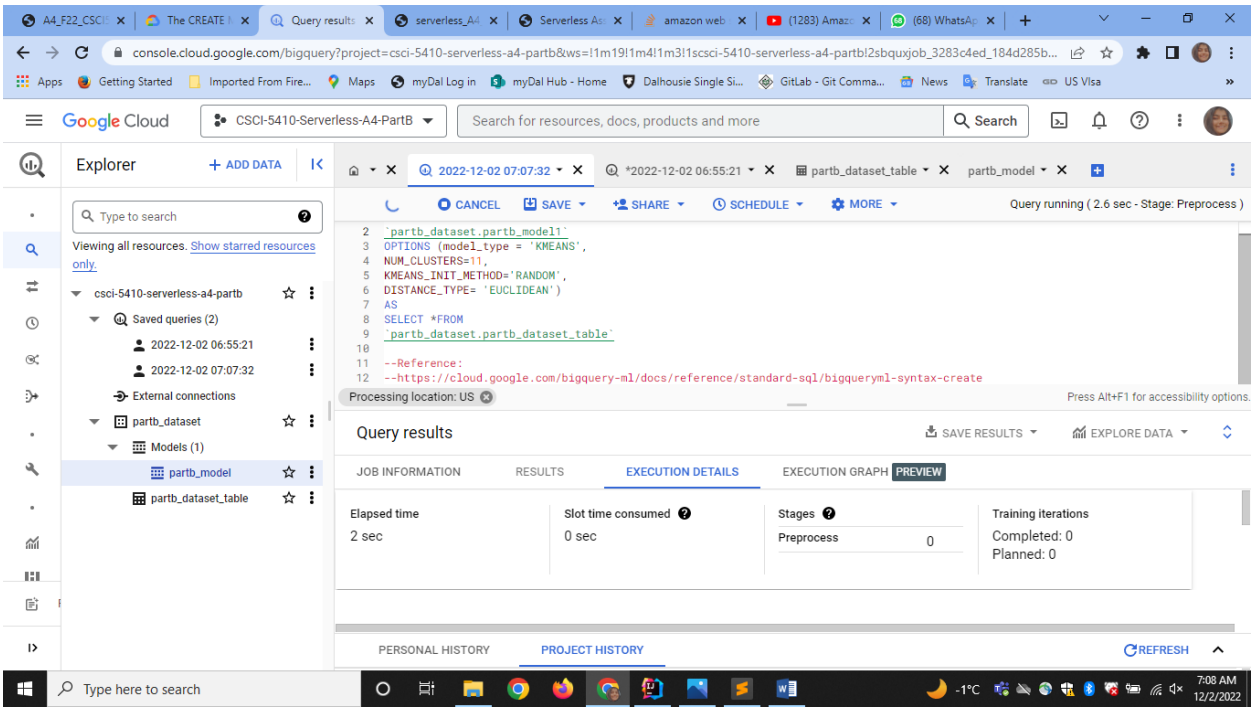


Fig 40: Query running

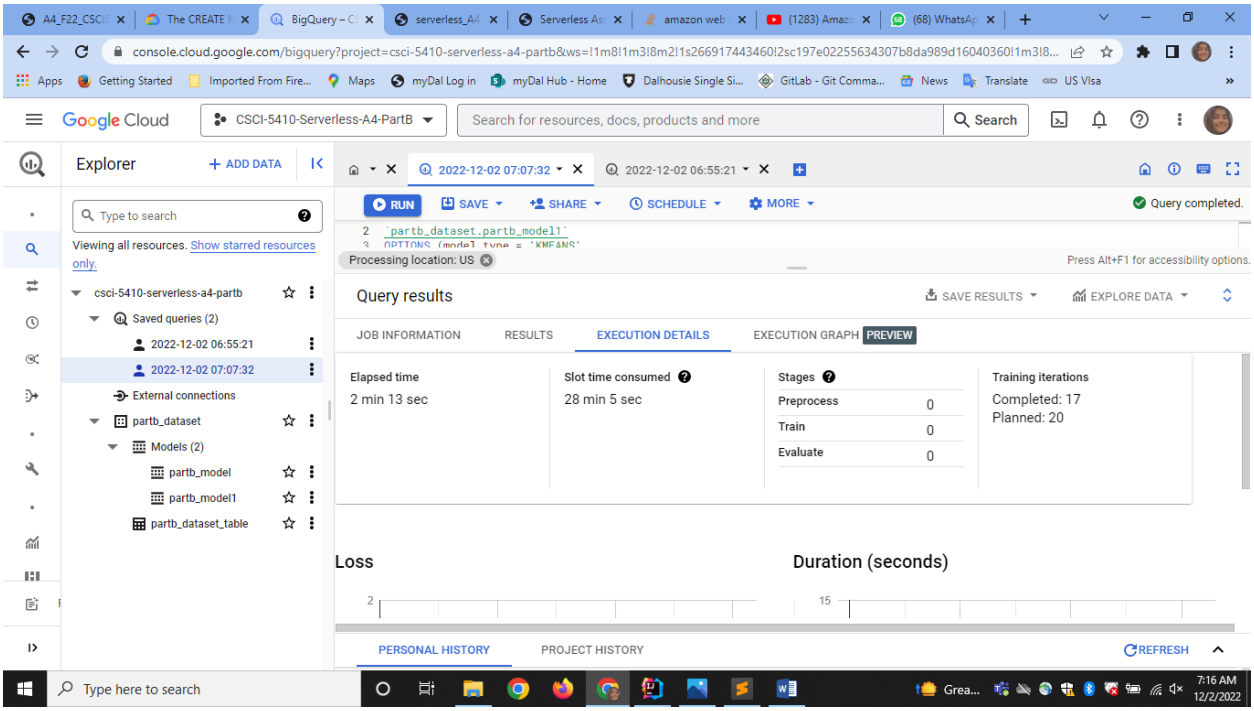


Fig 41: Query completed

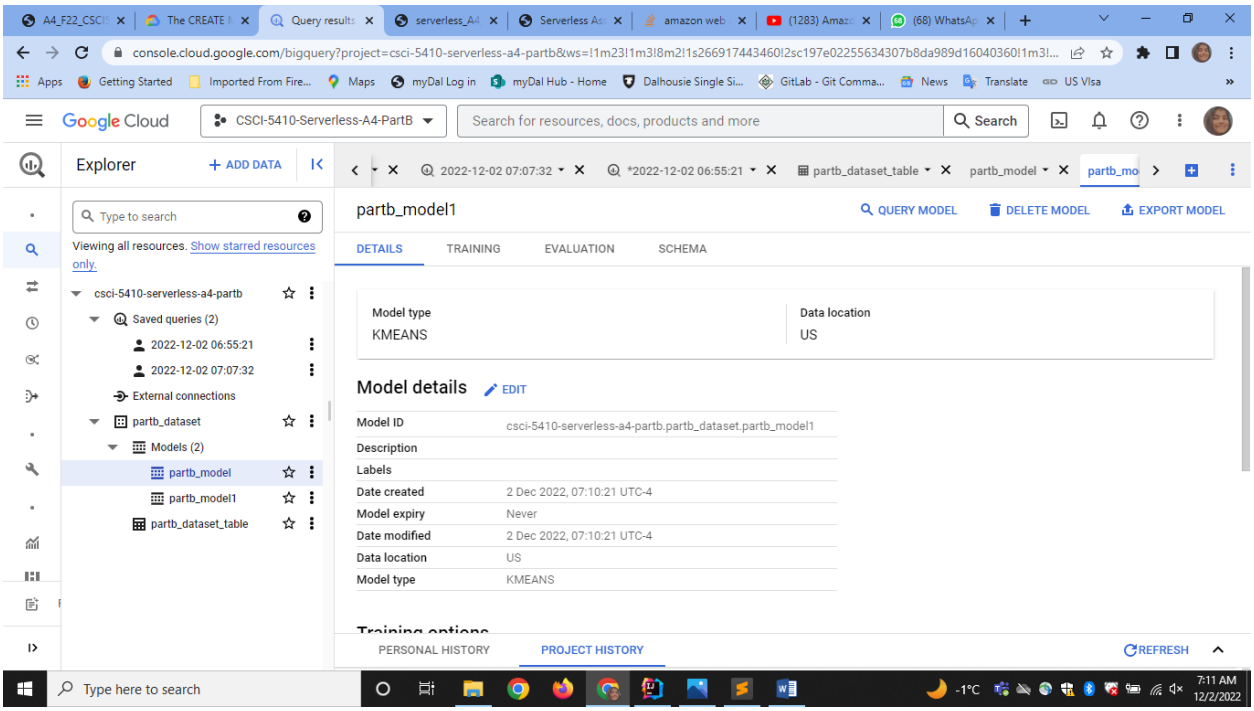


Fig 42: Model details for "partb_model1"

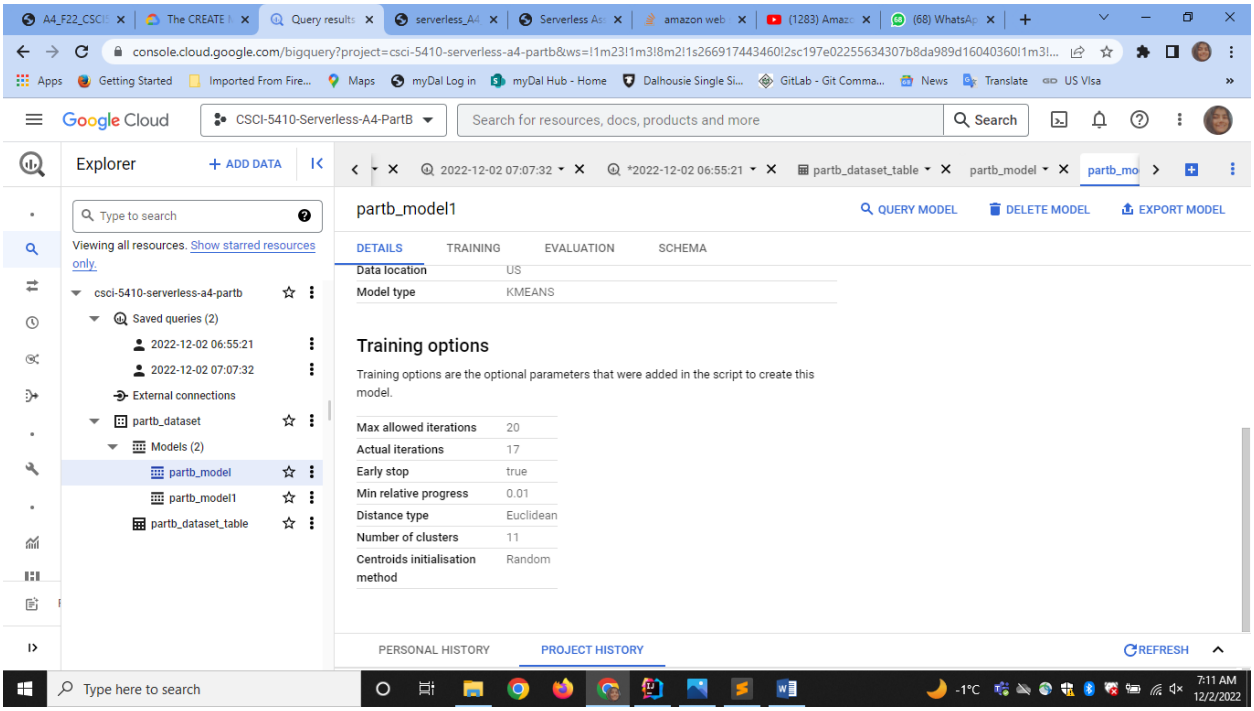


Fig 43: Model details for "partb_model1"

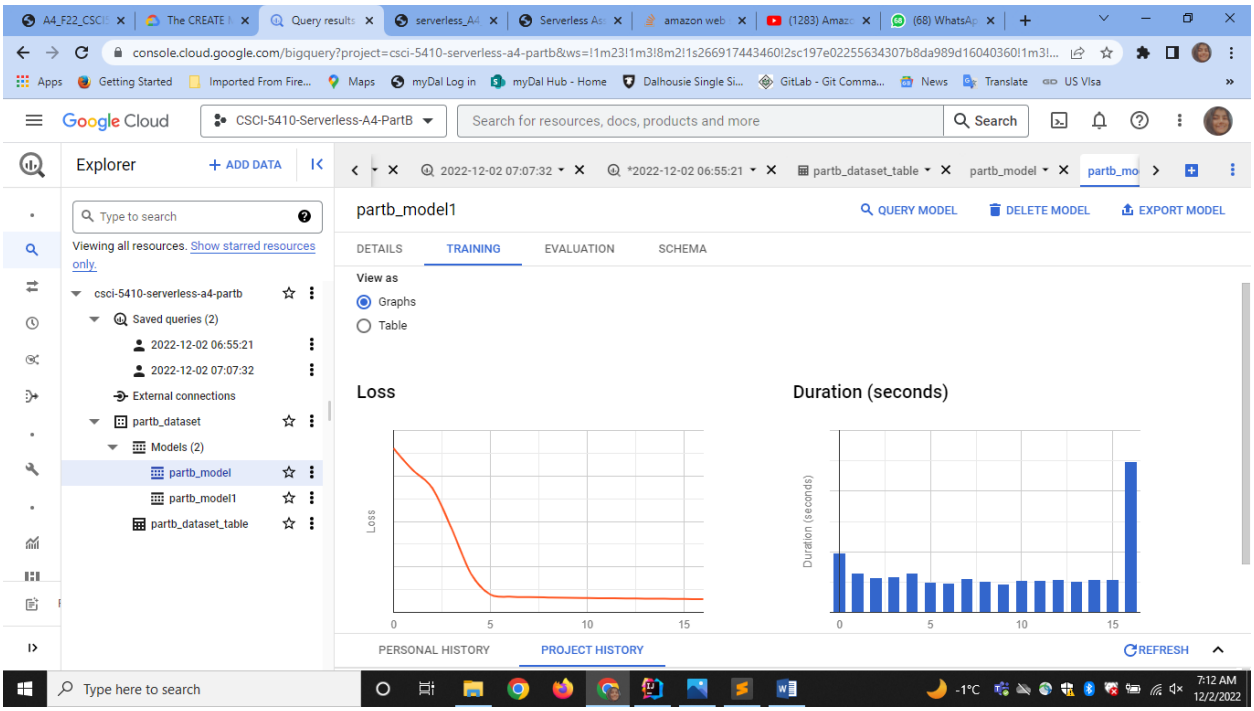


Fig 44: Training details for "partb_model1" – Graph view

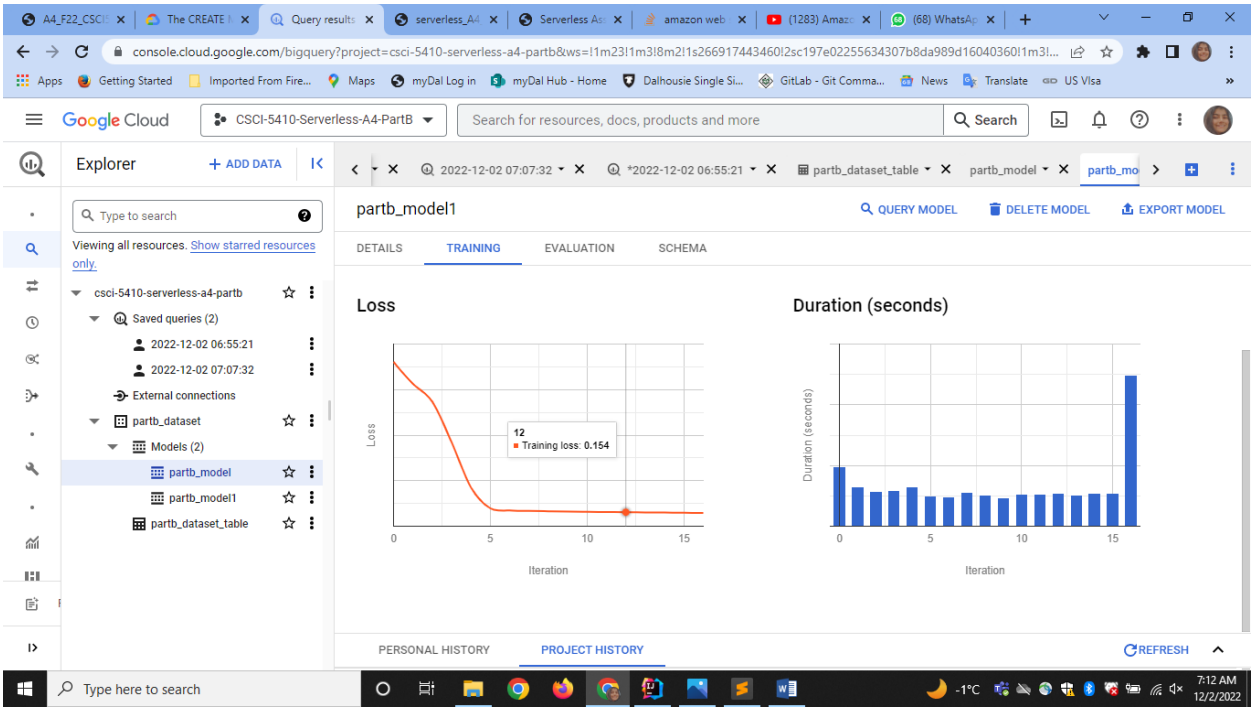


Fig 45: Training details for "partb_model1" – Graph view

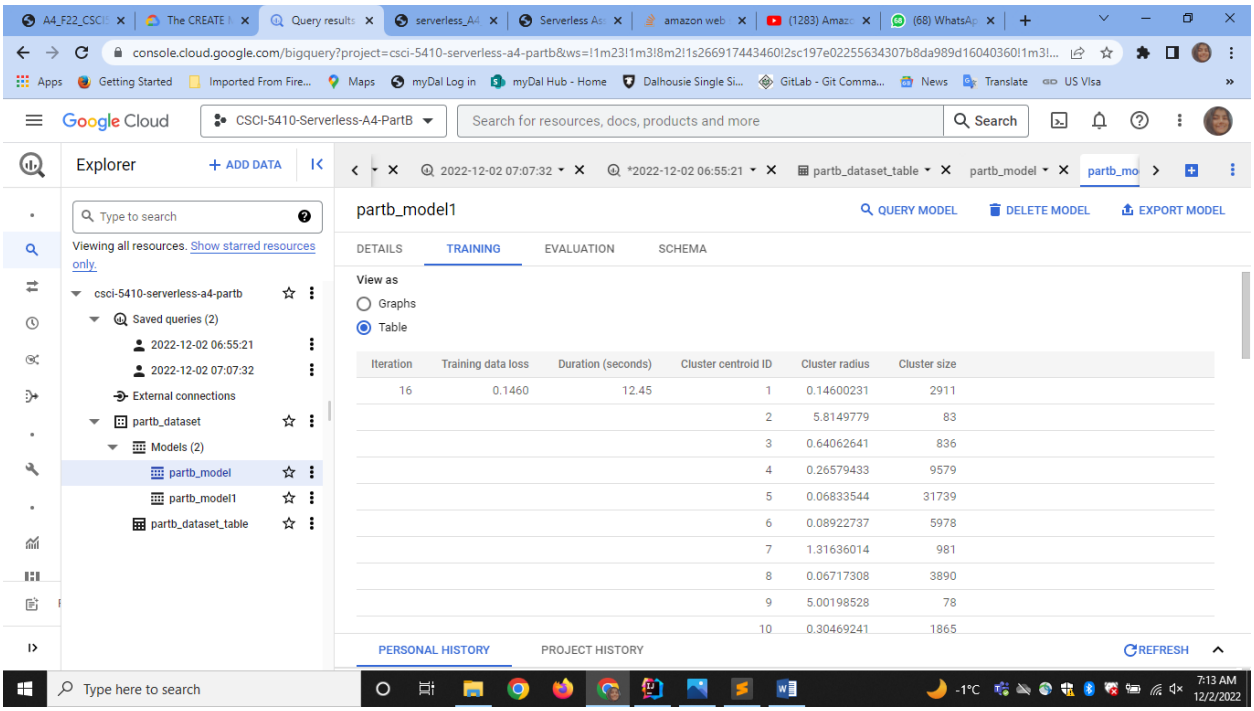


Fig 46: Training details for "partb_model1" – Table view

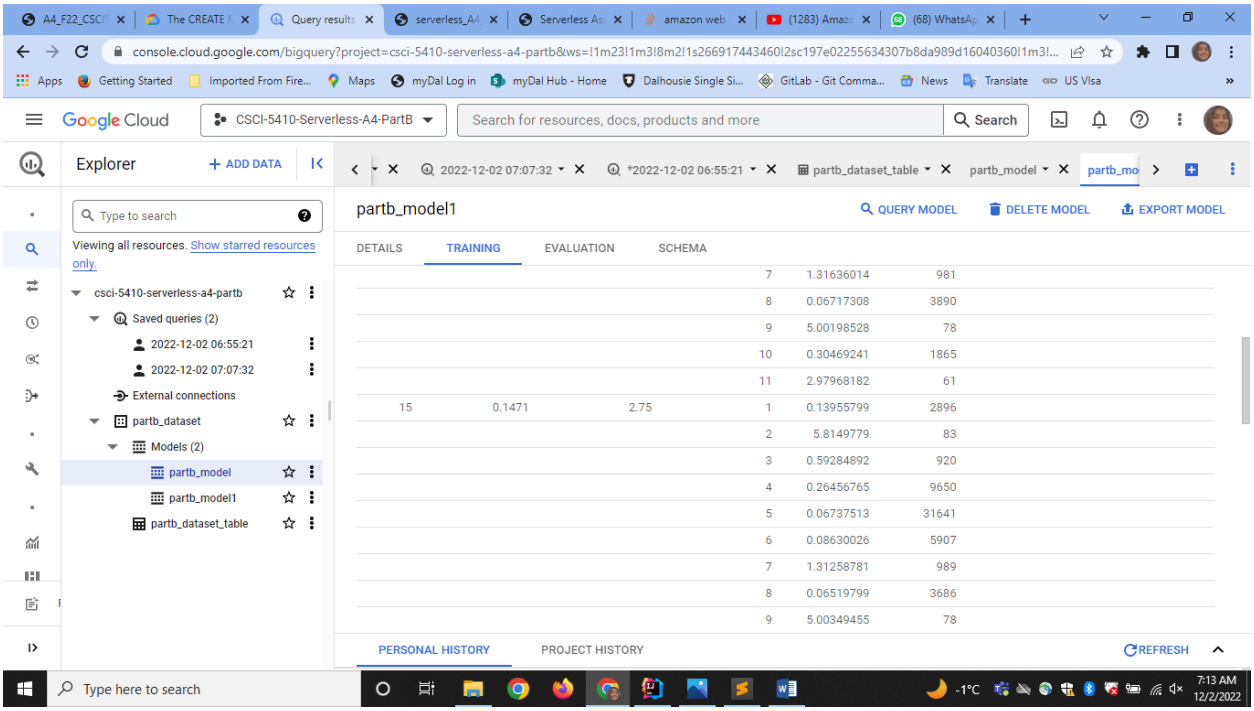


Fig 47: Training details for "partb_model1" – Table view

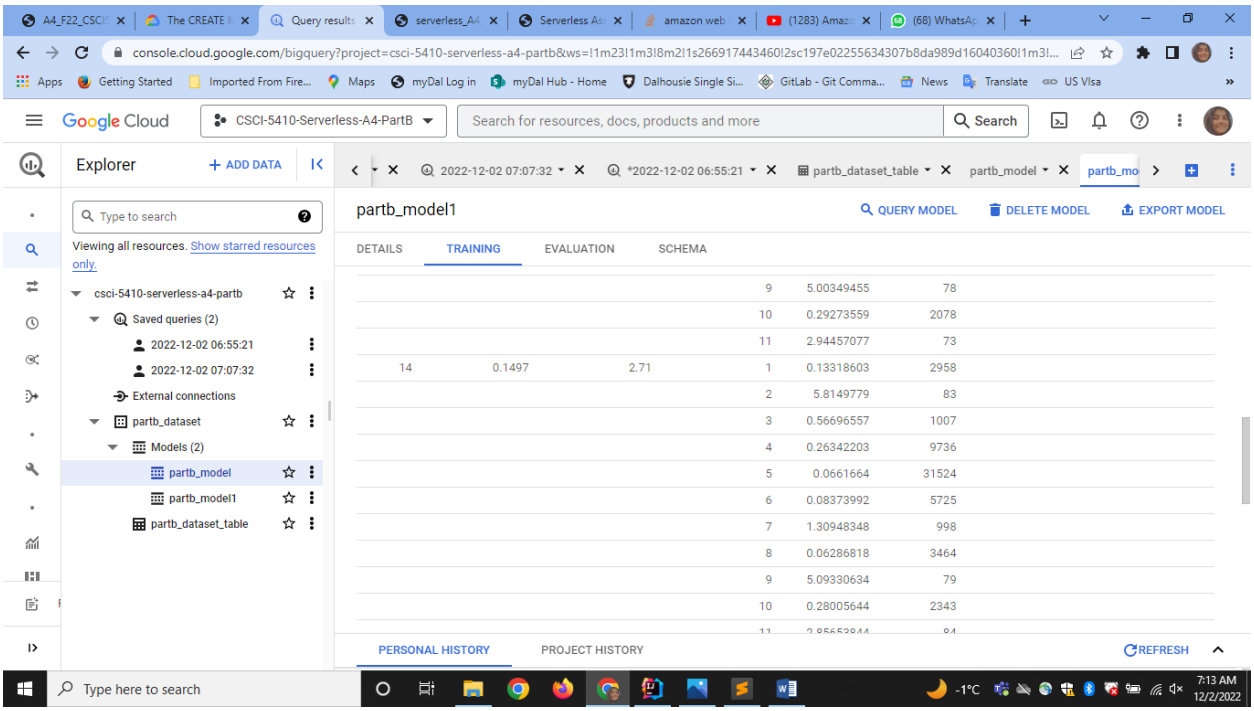
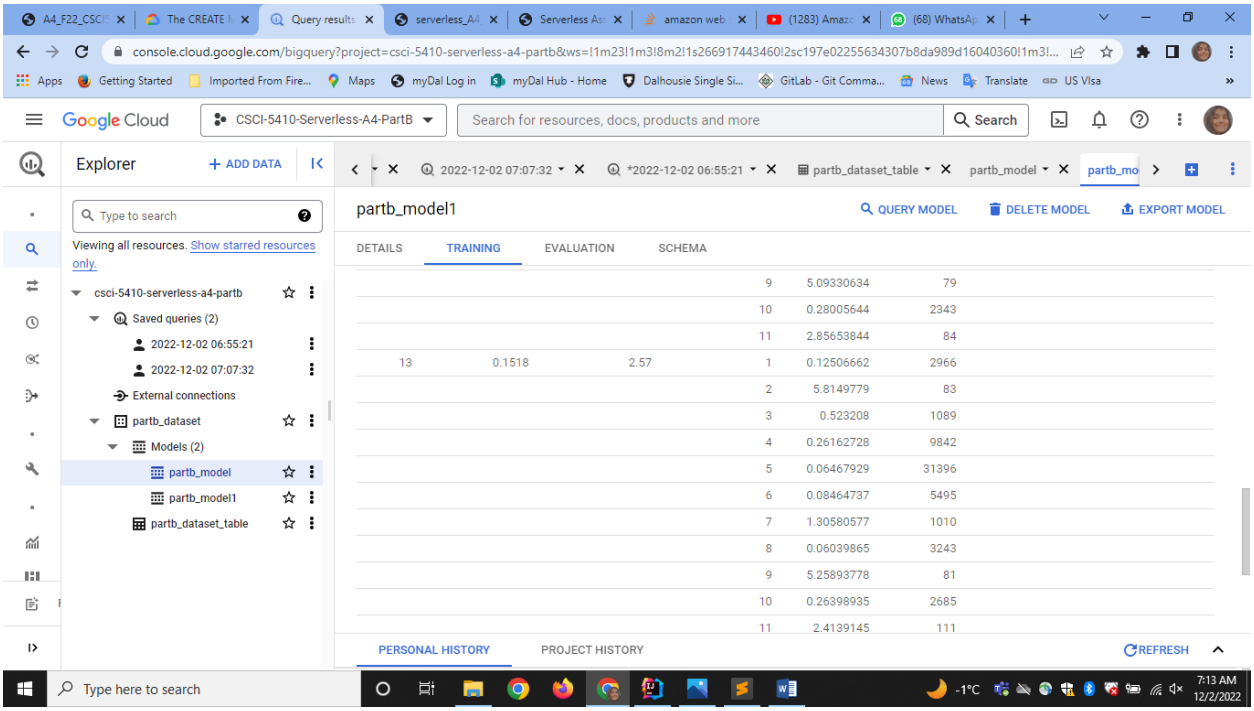
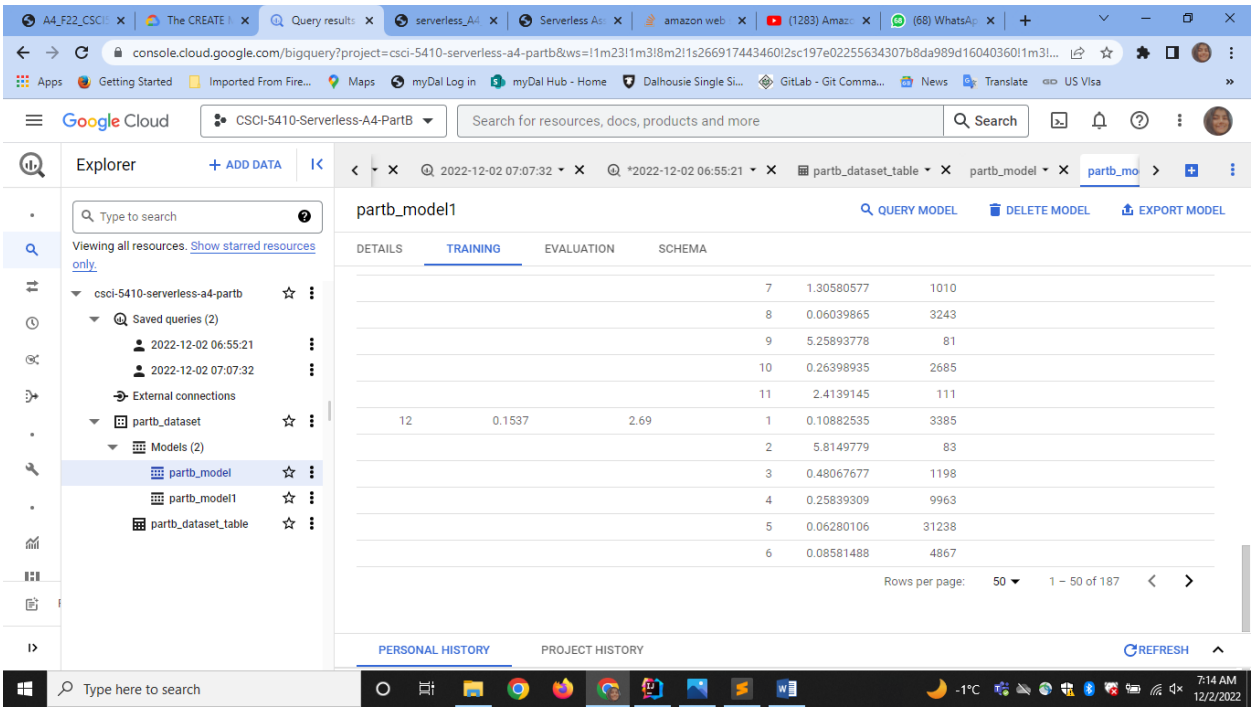


Fig 48: Training details for "partb_model1" – Table view



			9	5.09330634	79
			10	0.28005644	2343
			11	2.85653844	84
13	0.1518	2.57	1	0.12506662	2966
			2	5.8149779	83
			3	0.523208	1089
			4	0.26162728	9842
			5	0.06467929	31396
			6	0.08464737	5495
			7	1.30580577	1010
			8	0.06039865	3243
			9	5.25893778	81
			10	0.26398935	2685
			11	2.4139145	111

Fig 49: Training details for "partb_model1" – Table view



			7	1.30580577	1010
			8	0.06039865	3243
			9	5.25893778	81
			10	0.26398935	2685
			11	2.4139145	111
12	0.1537	2.69	1	0.10882535	3385
			2	5.8149779	83
			3	0.48067677	1198
			4	0.25839309	9963
			5	0.06280106	31238
			6	0.08581488	4867

Fig 50: Training details for "partb_model1" – Table view

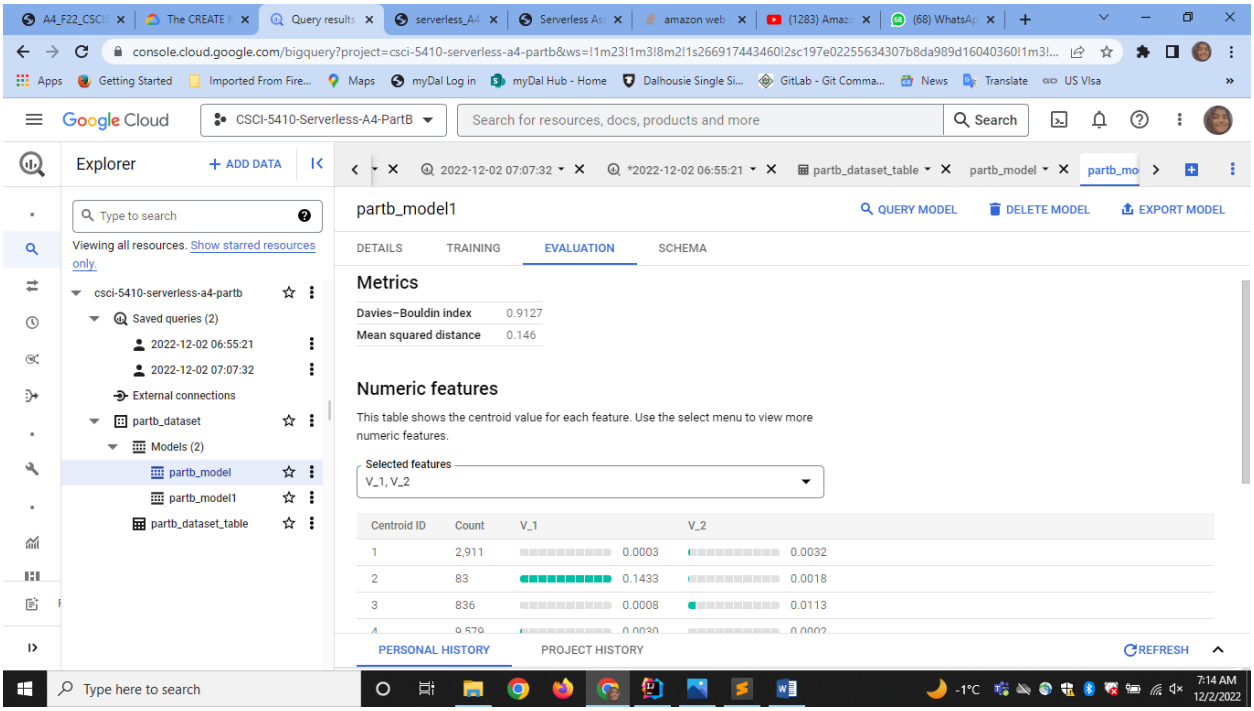


Fig 51: Evaluation details for "partb_model1"

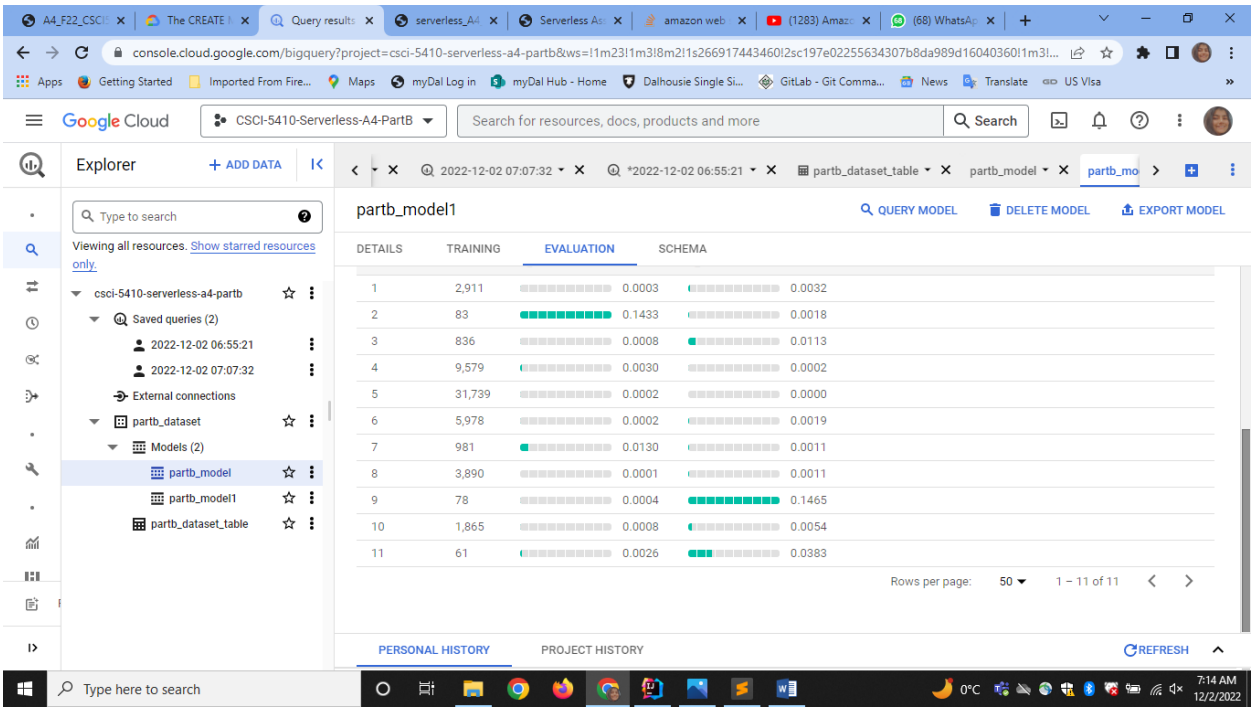


Fig 52: Evaluation details for "partb_model1"

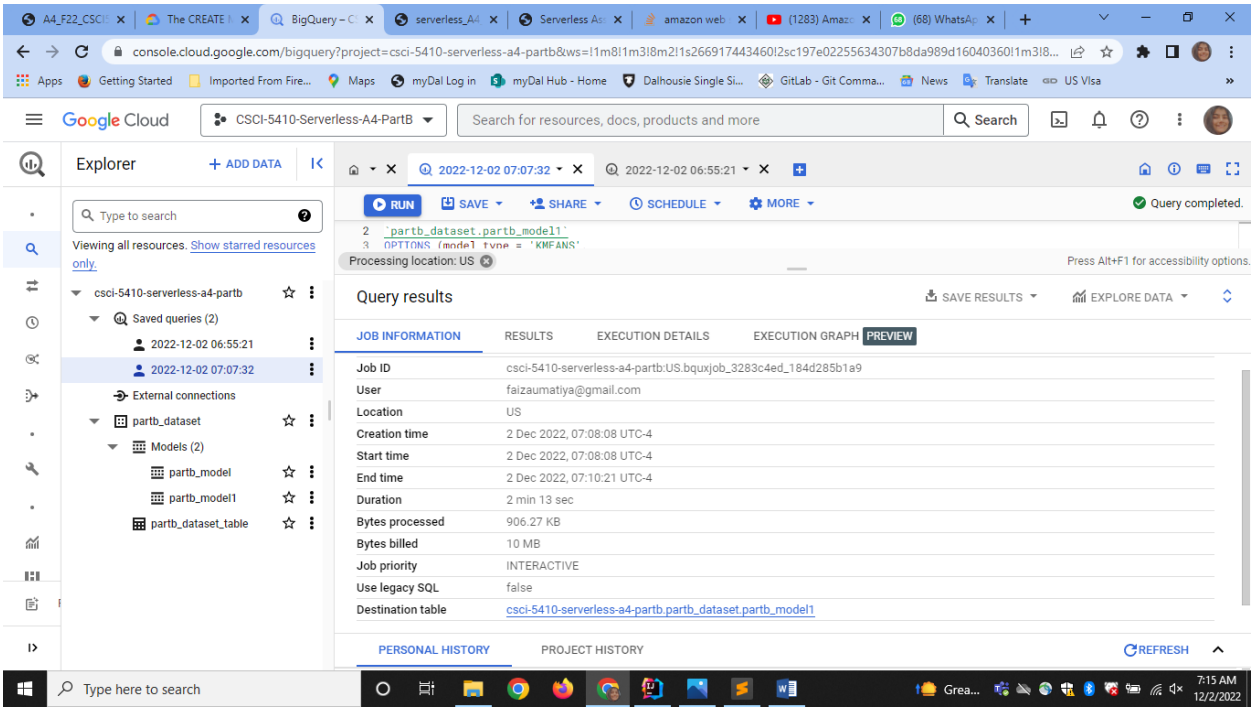


Fig 53: Job details for "partb_model1"

Step 10:

- First divide the provided dataset i.e. "SDey_FTP_input" into 25% test set and 75% training set.
- "SDey_FTP_input" contains total records of 58002.
- I made a separate file for 25% test set named as "SDey_FTP_input_25PercentTestSet" which contains 14500 records (25% of 58002) (Refer fig 54).
- And remaining 75% of the record i.e. 43502 record is in the other file named as "SDey_FTP_input_75PercentTrainingSet" (Refer fig 54).

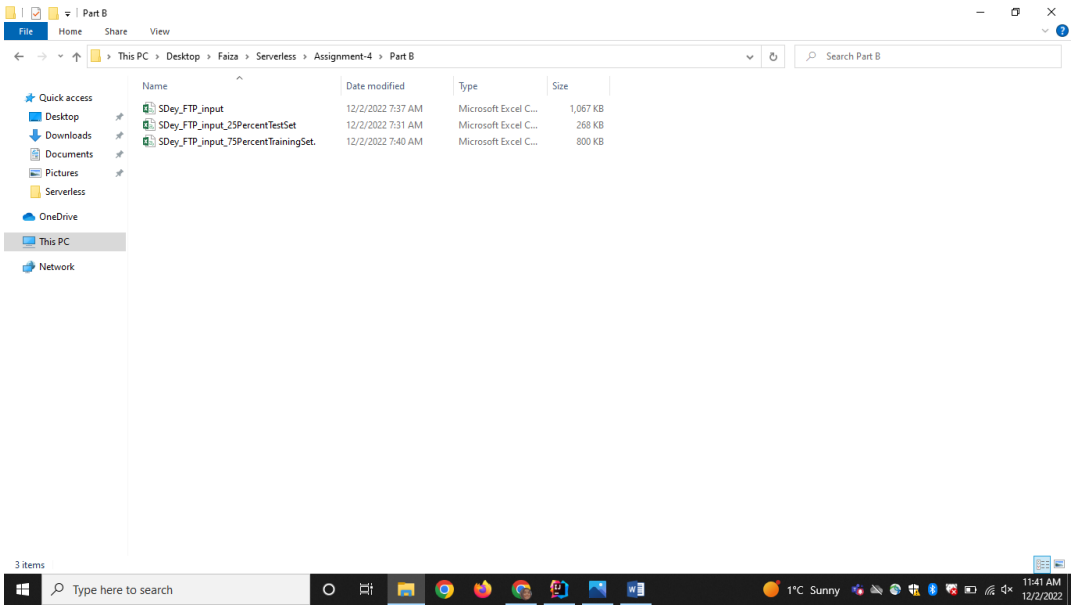


Fig 54: 3 different dataset

Step 11:

- **Create a dataset with 25% dataset** as test set by filling the details as shown in the fig 55 and 56.
- I used the "SDey_FTP_input_25PercentTestSet" dataset as shown in the fig 59.
- Now we need to add data. But in order to add data we need to create a table to upload file as shown in the fig 58, 59, 60 and 61.
- Refer fig 62, to see the partb_testSet_table has been created.

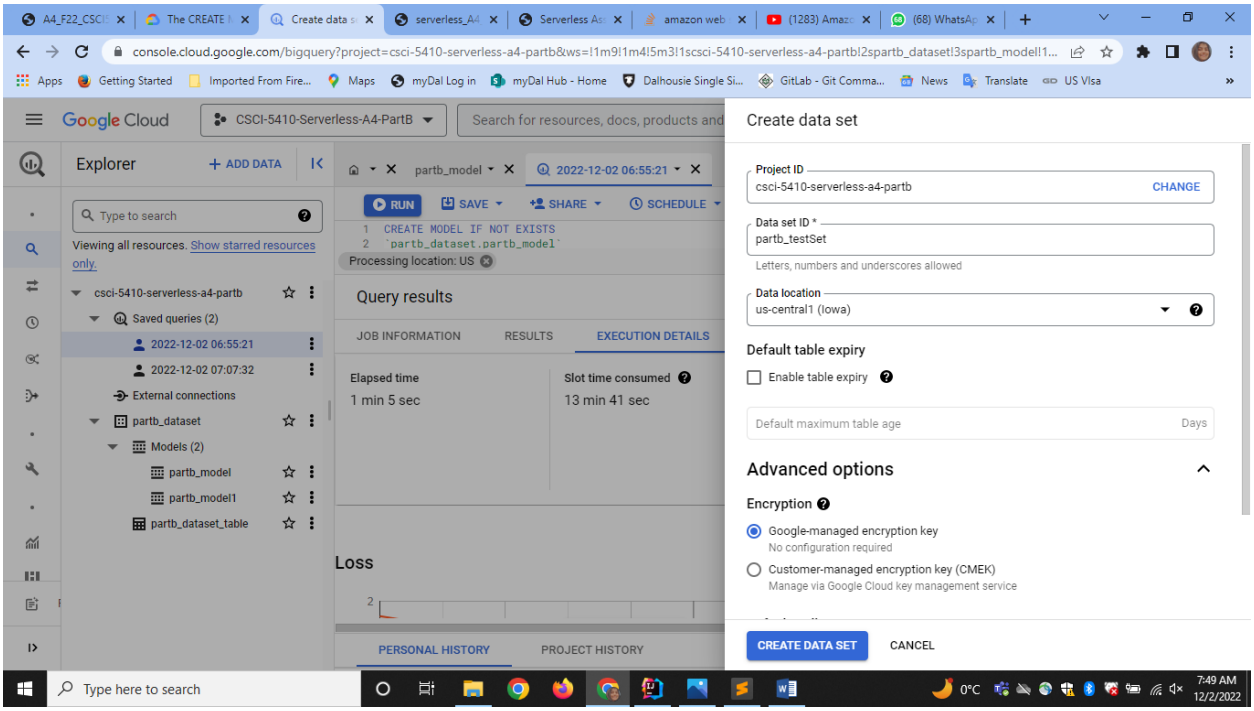


Fig 55: "Create data set" form

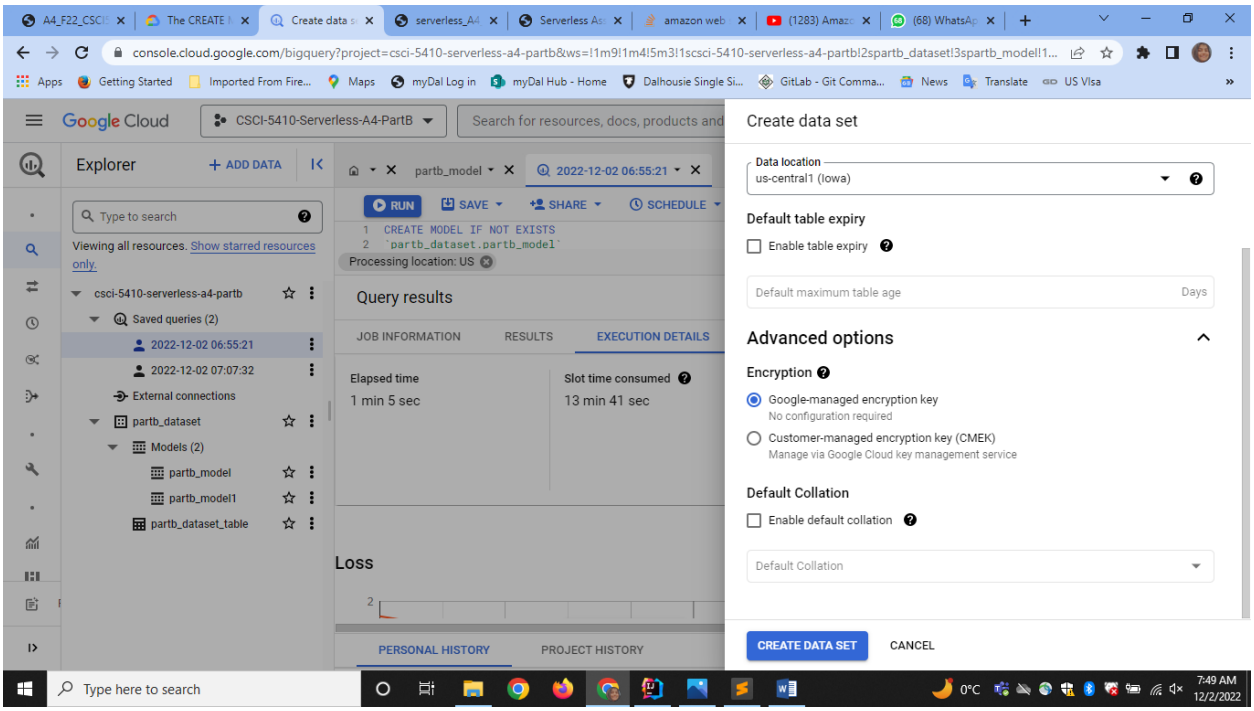


Fig 56: "Create data set" form

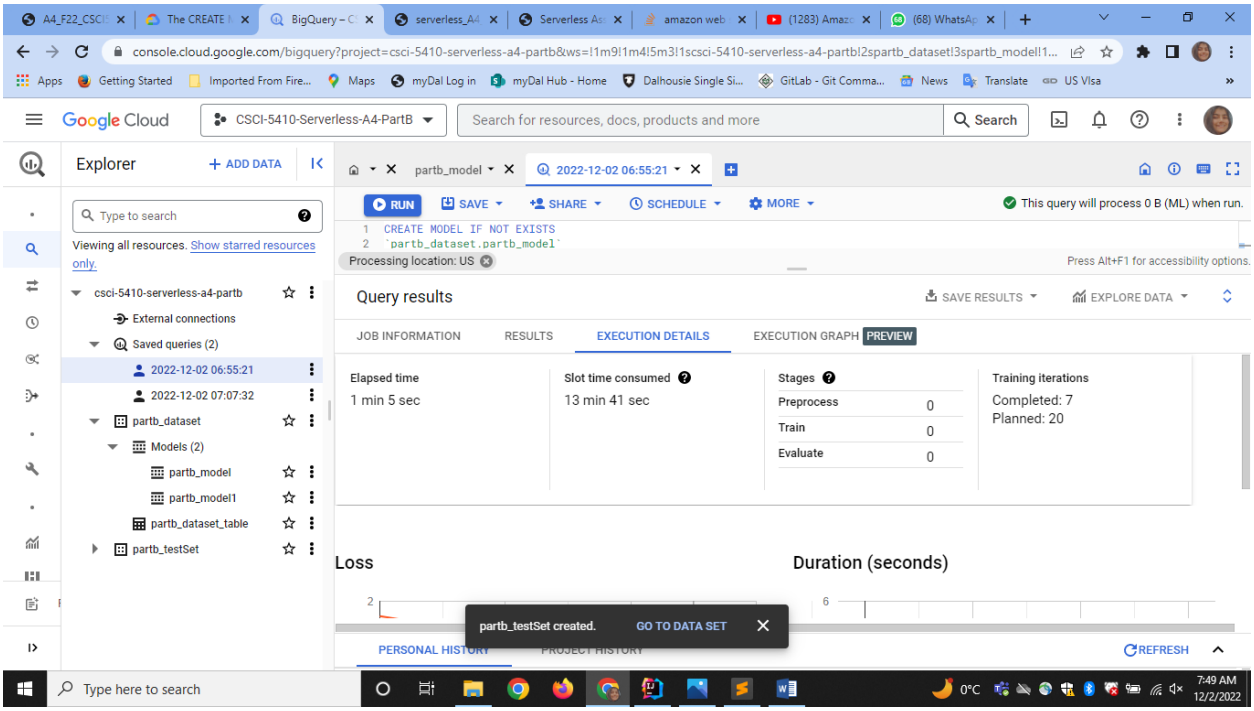


Fig 57: "partb_testSet" dataset has been created

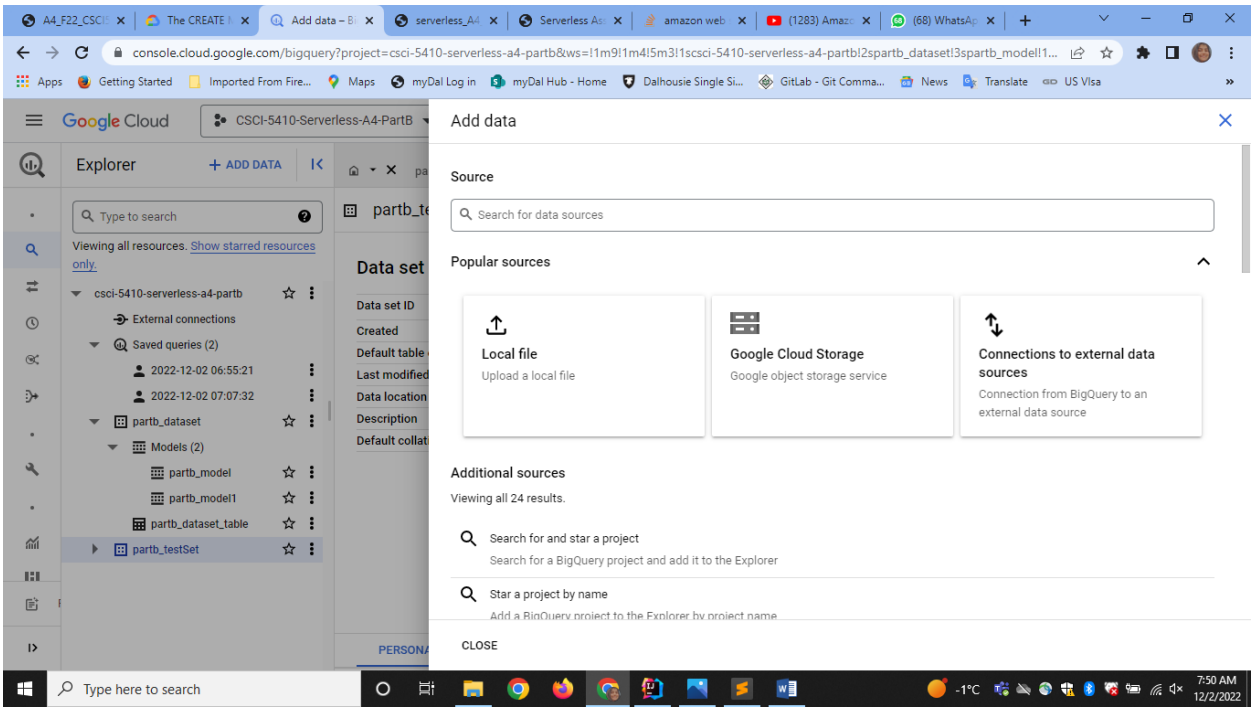


Fig 58: "Add data" form

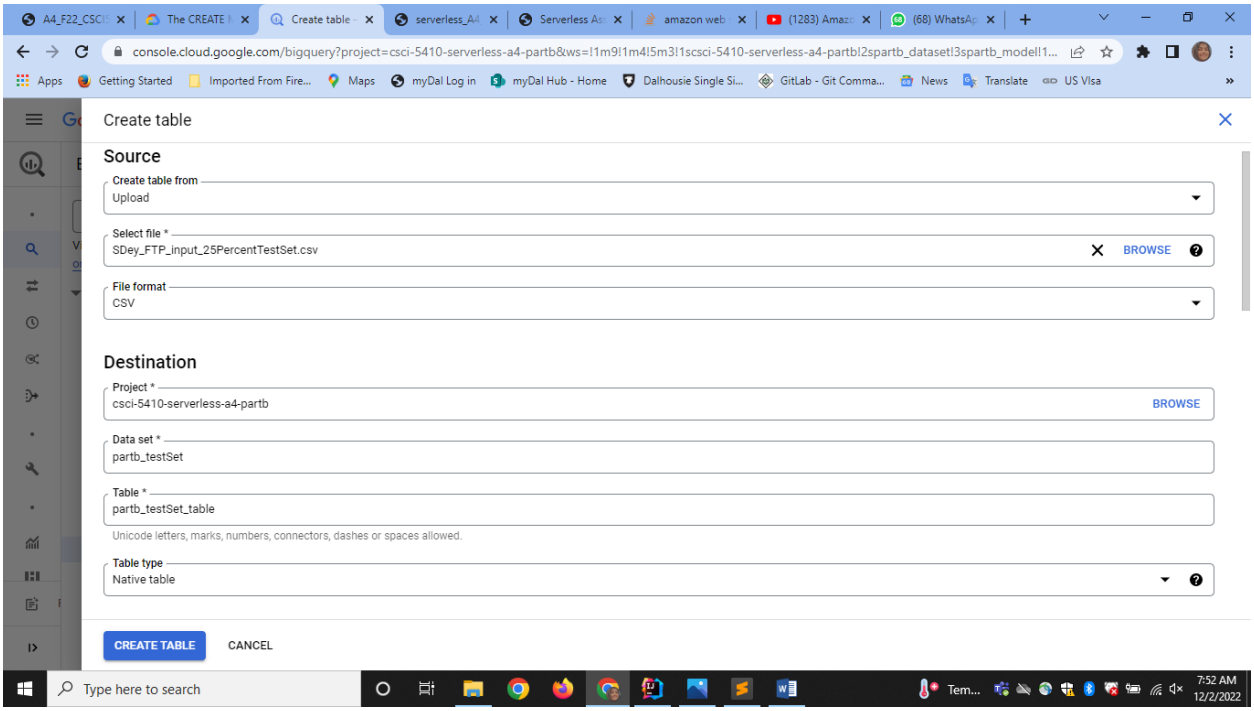


Fig 59: "Add data" form

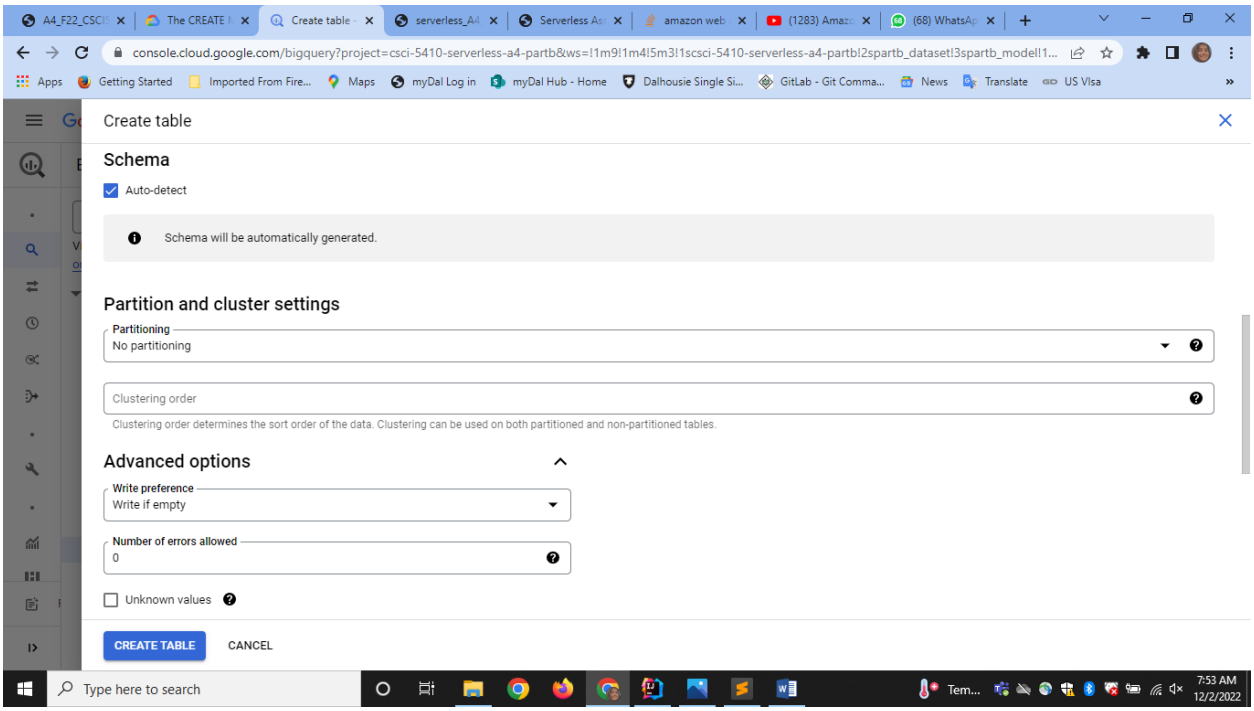


Fig 60: "Add data" form

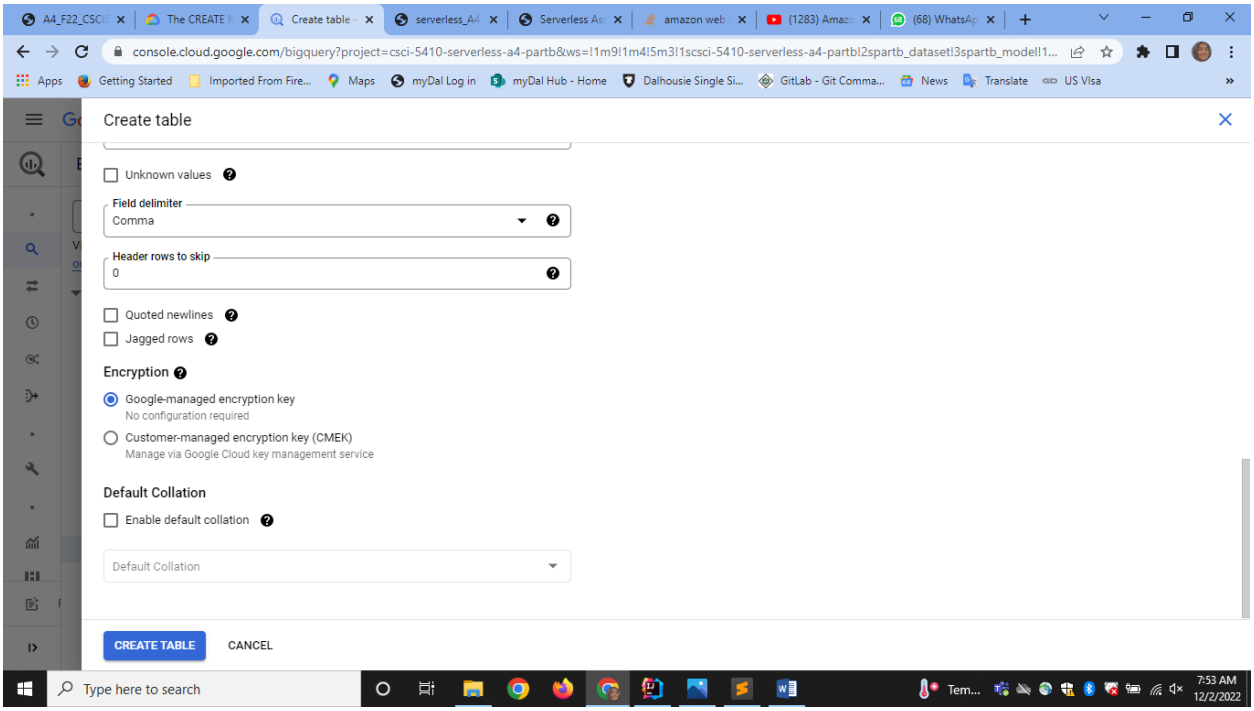


Fig 61: "Add data" form

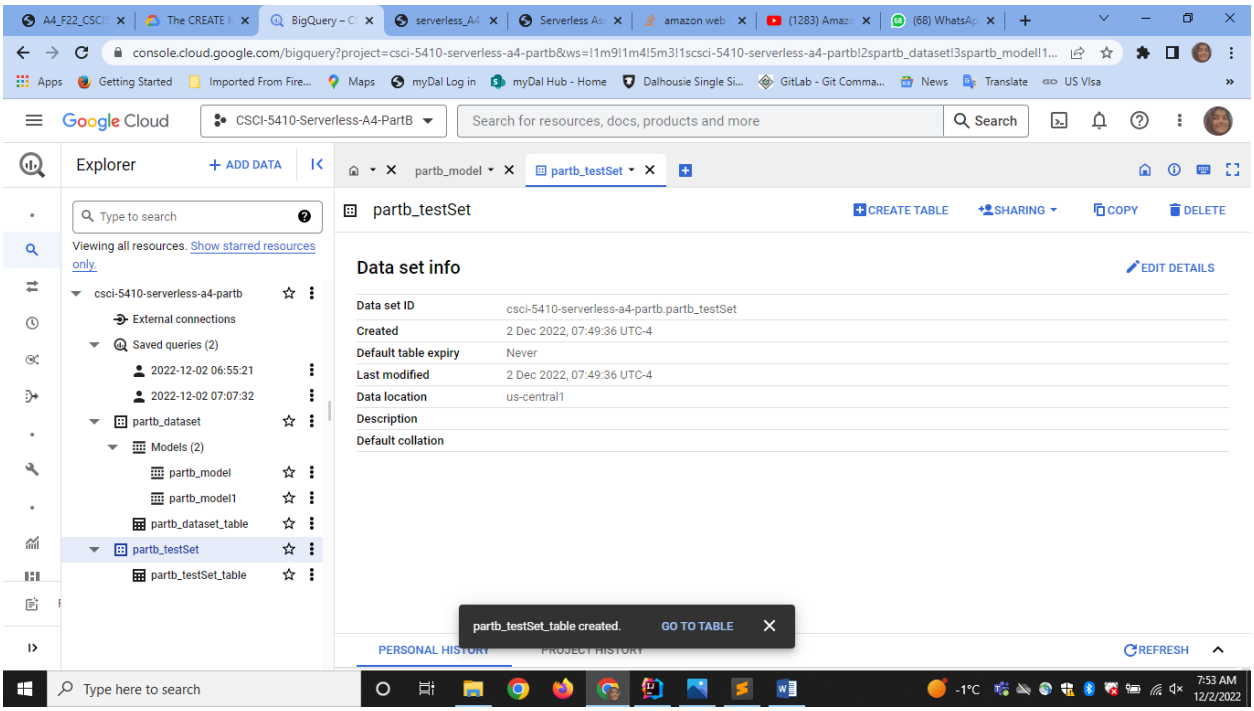


Fig 62: "partb_testSet_table" has been created

Step 12:

- Change the settings for the query by clicking on the “more” option under “partb_testSet_table” as shown in the fig 63. Then go to “Query settings” as shown in the fig. 63.
- Fill the “Query settings” details as shown in the fig 64, 65 and 66.
- Keep the location same as the “partb_testSet” which is “us-central-1 (Iowa)”.

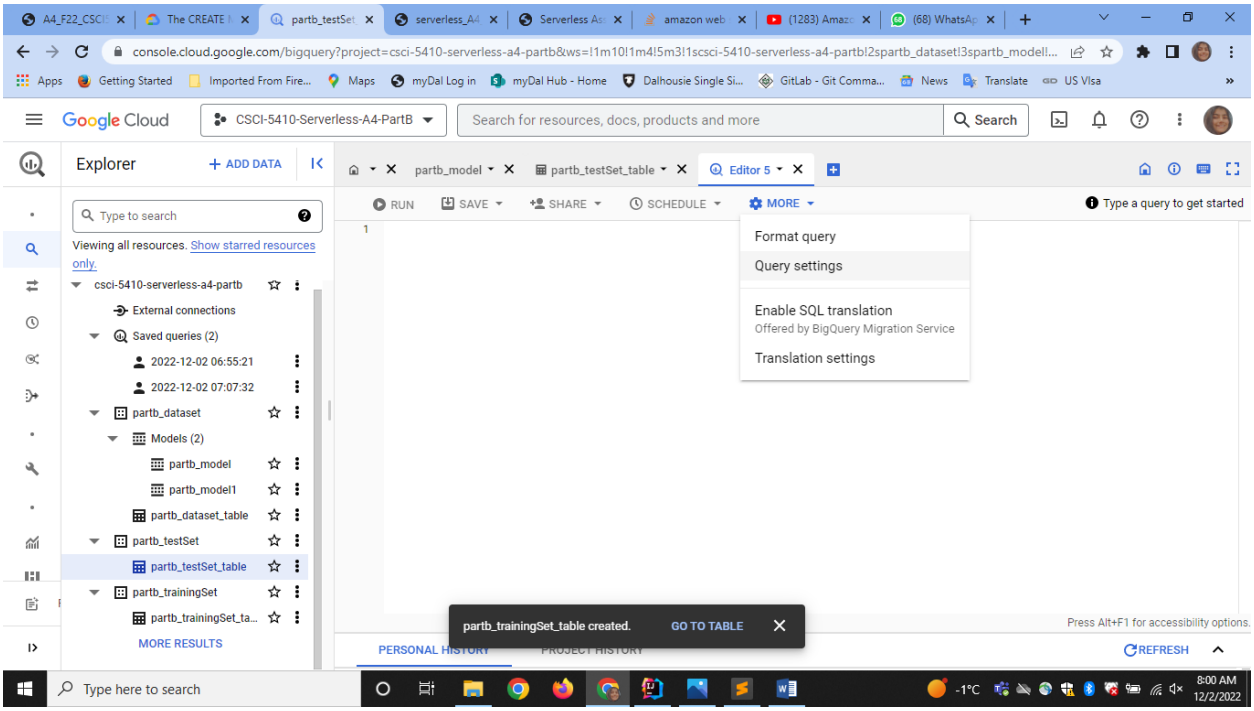


Fig 63: “Query settings” option

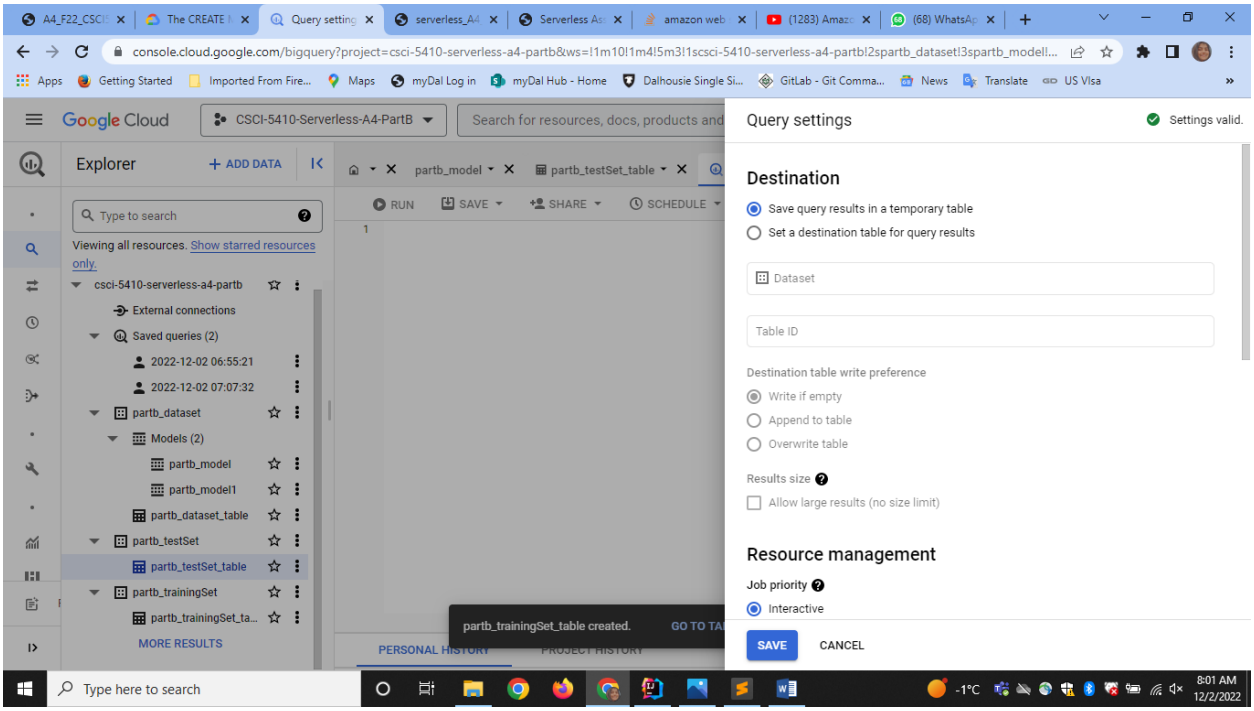


Fig 64: "Query settings" option

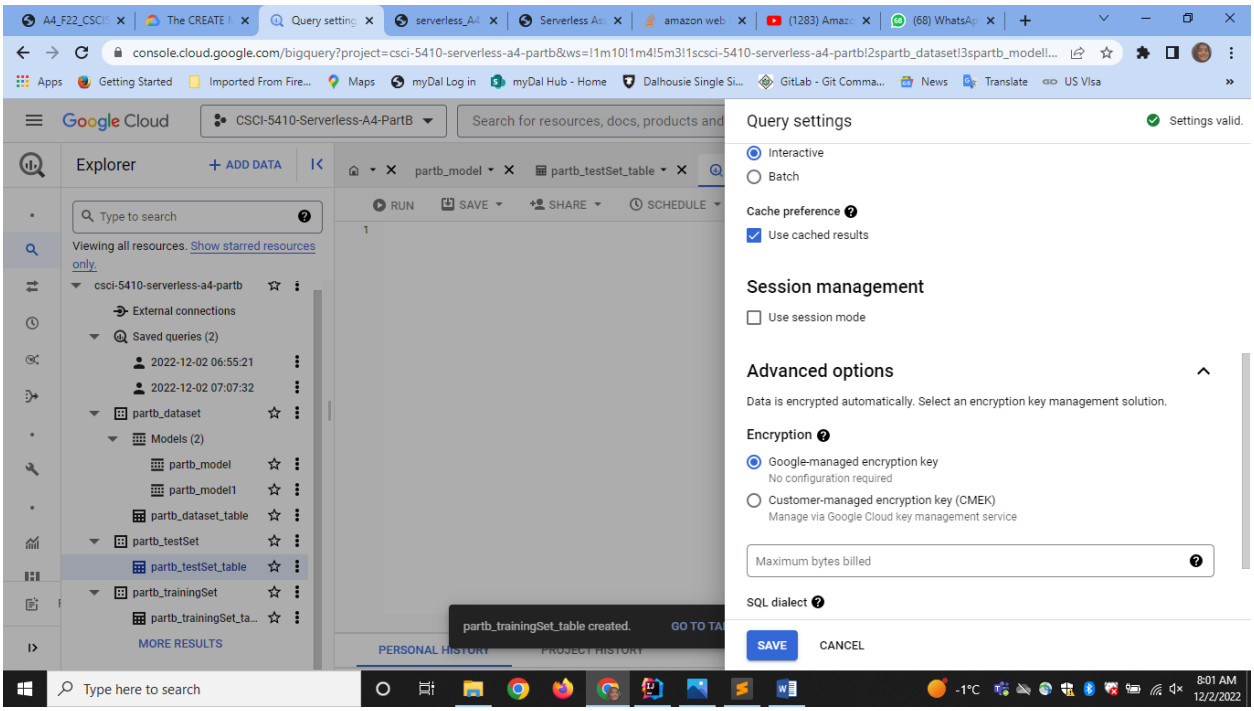


Fig 65: "Query settings" option

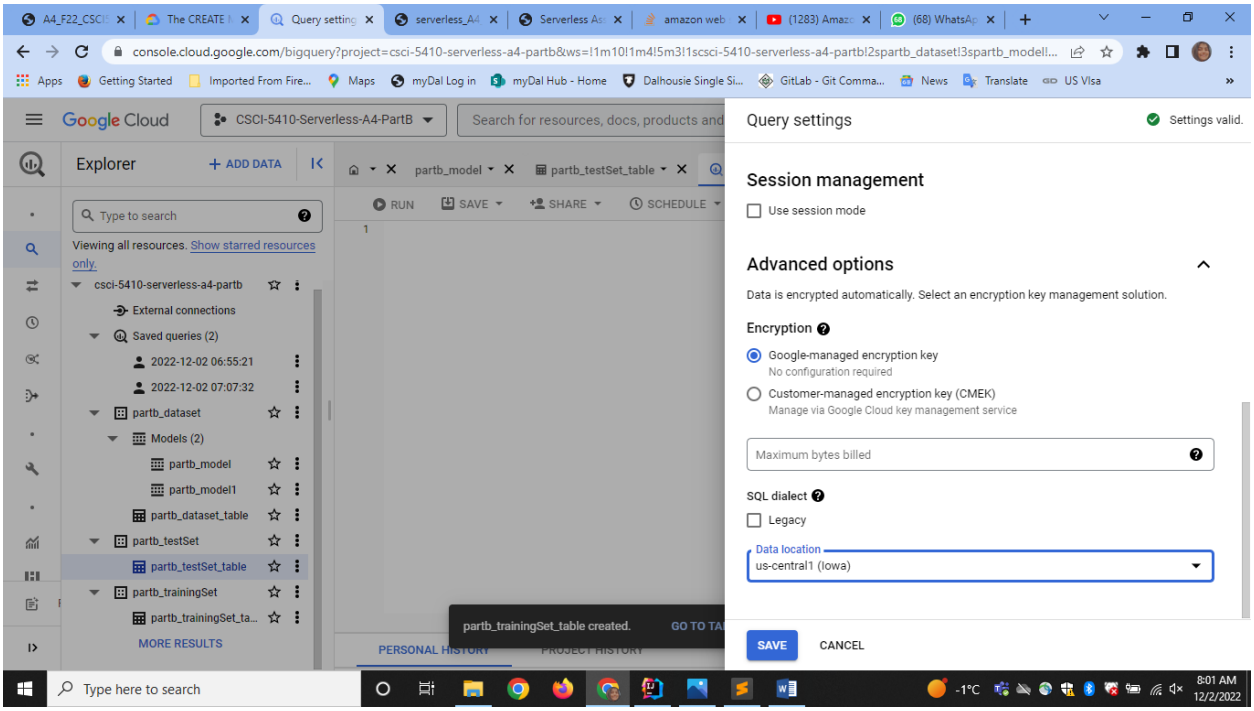


Fig 66: "Query settings" option

Step 13:

- I wrote a query to train the model as shown in the fig 67. The query creates a k-means model with 25 clusters using the "DISTANCE_TYPE" value of "EUCLIDEAN_DISTANCE" and standardize features [1].
- Fig 68 shows query running with the elapsed time.
- Fig 69 and 70 provides the model details which is been created after query completes its execution.
- Fig 71, 72, 73, 74, 75 and 76 gives the graphical view, table view of the results.
- Fig 77, 78 and 79 represents evaluation details of the "partb_model3".
- Fig 53 represents job details of the "partb_model3".

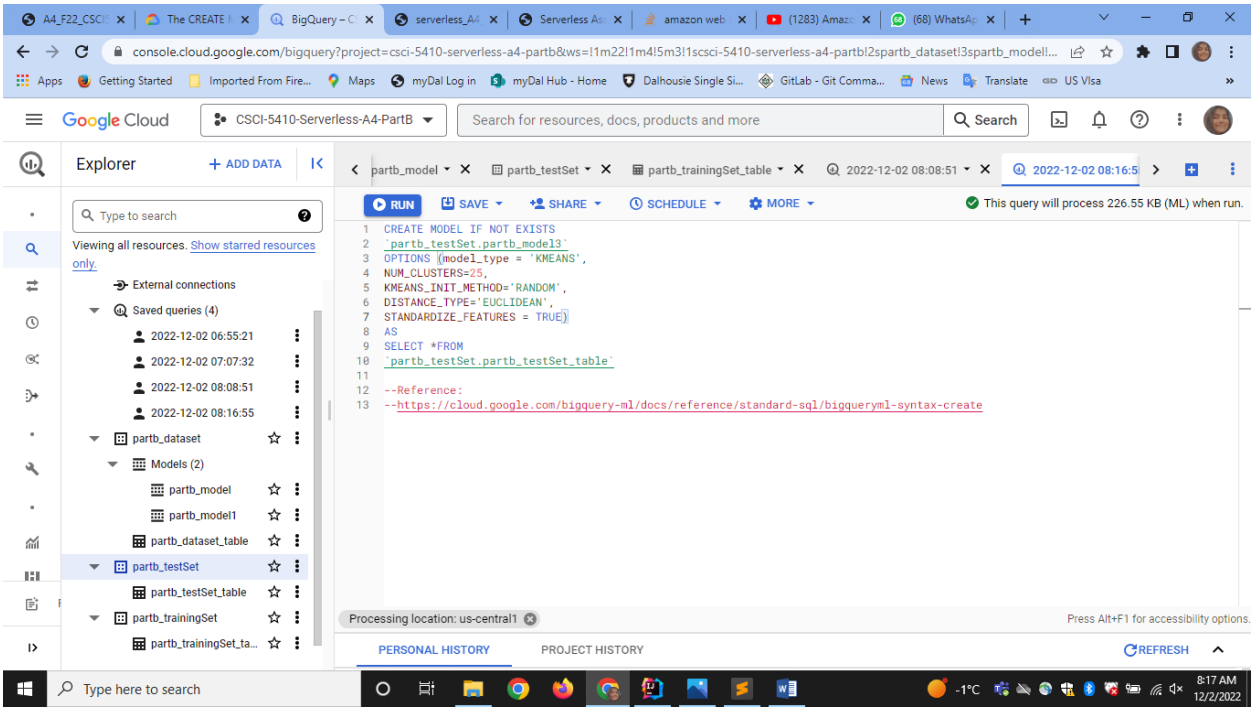


Fig 67: Query to create model

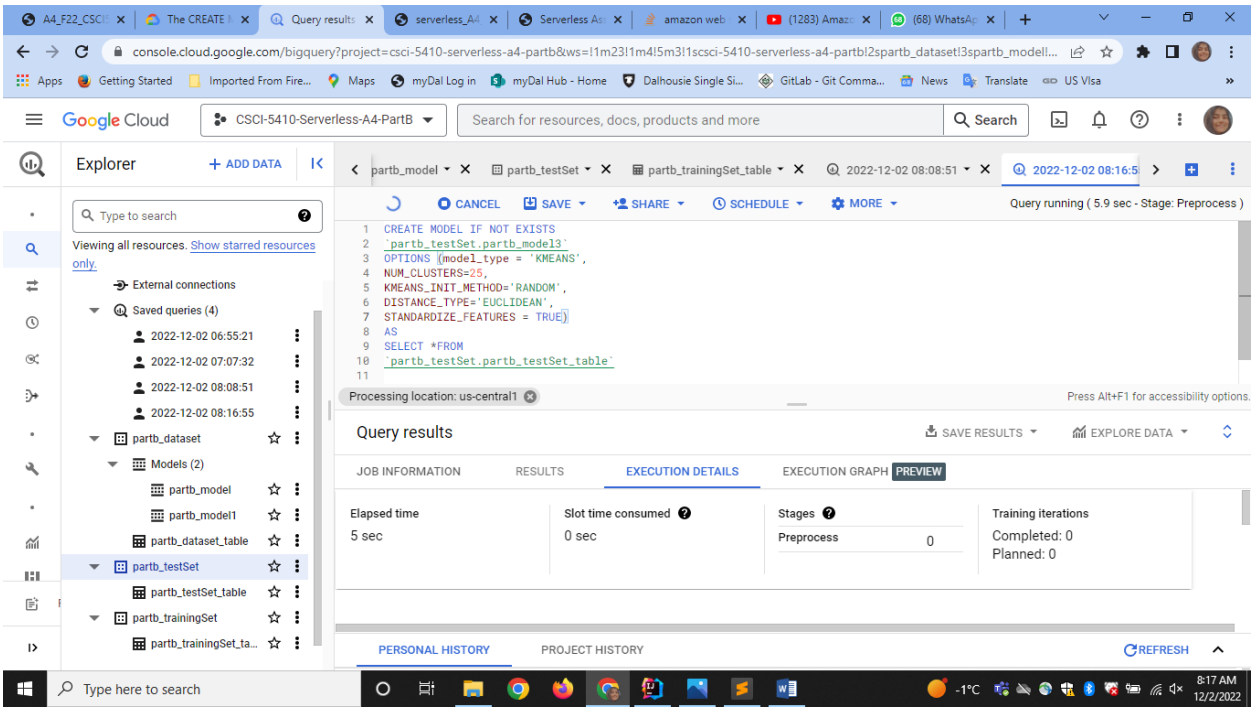


Fig 68: Query running

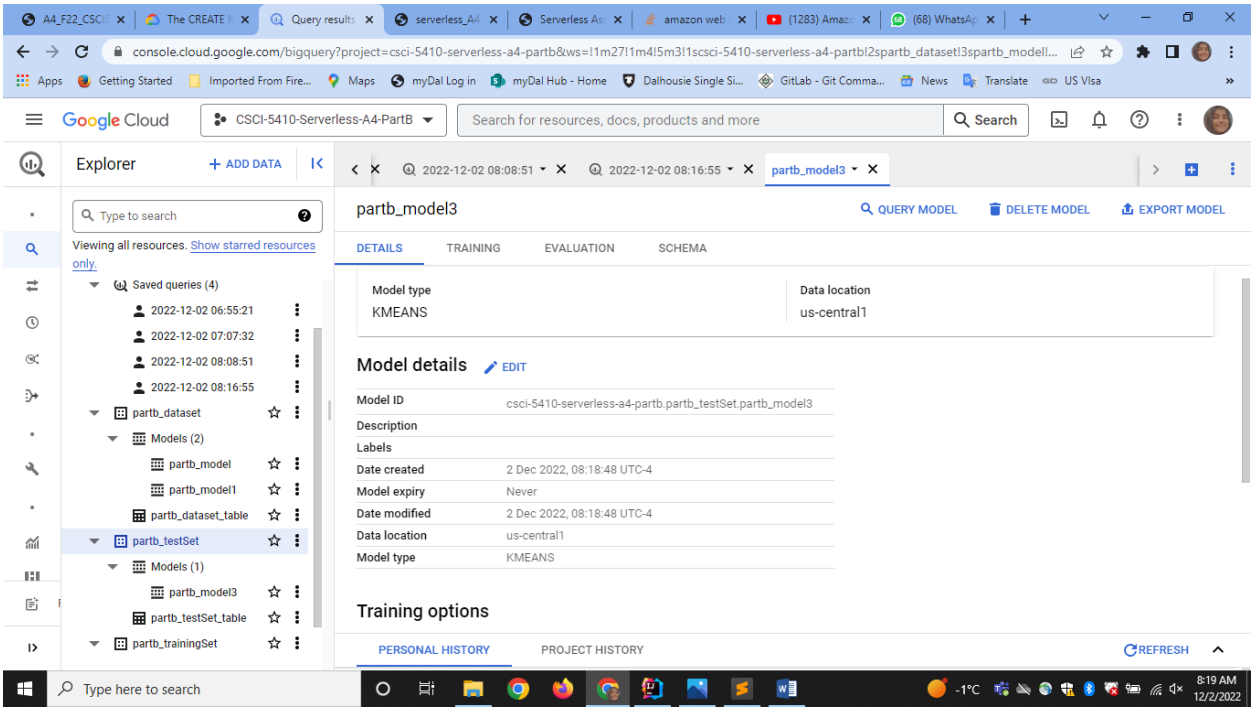


Fig 69: Model details for "partb_model3"

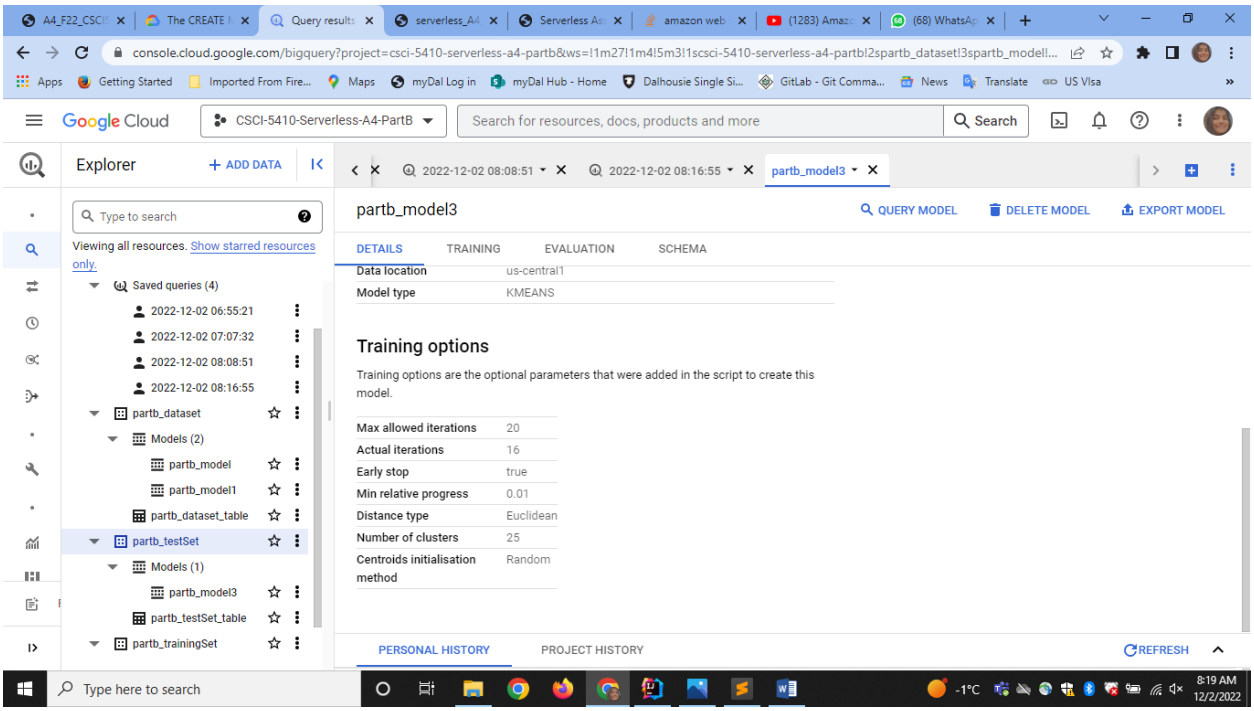


Fig 70: Model details for "partb_model3"

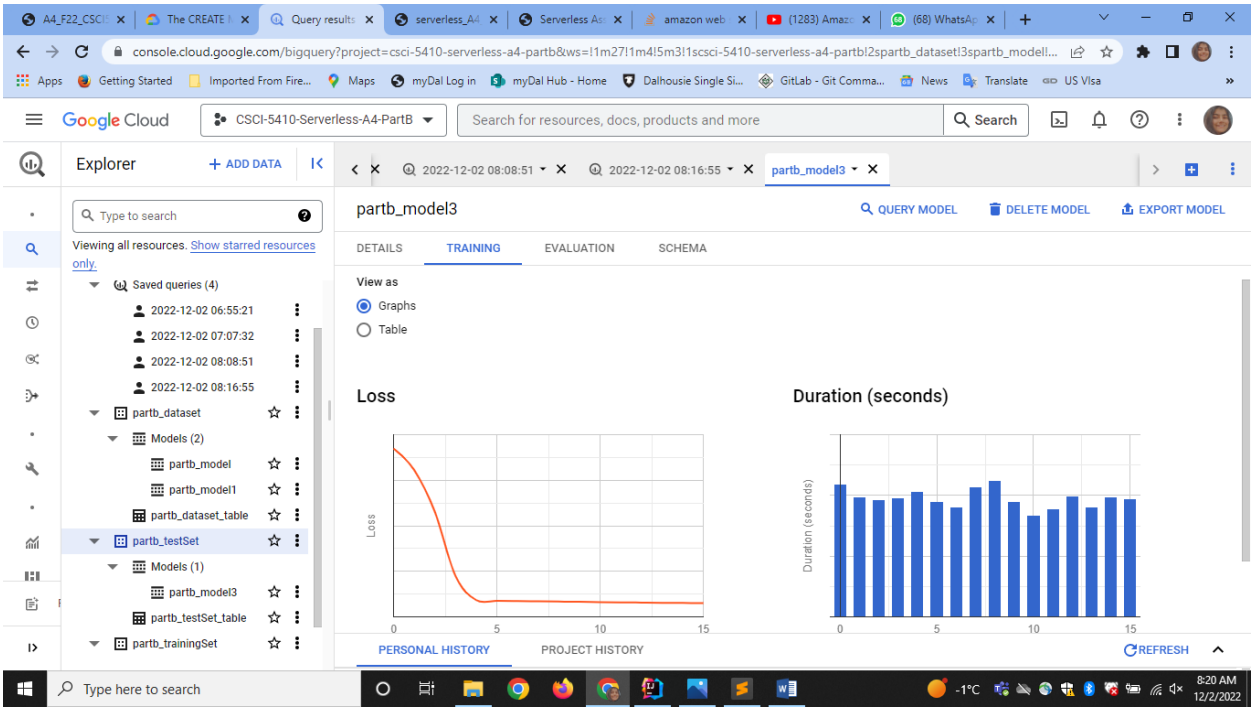


Fig 71: Training details for "partb_model3" – Graph view

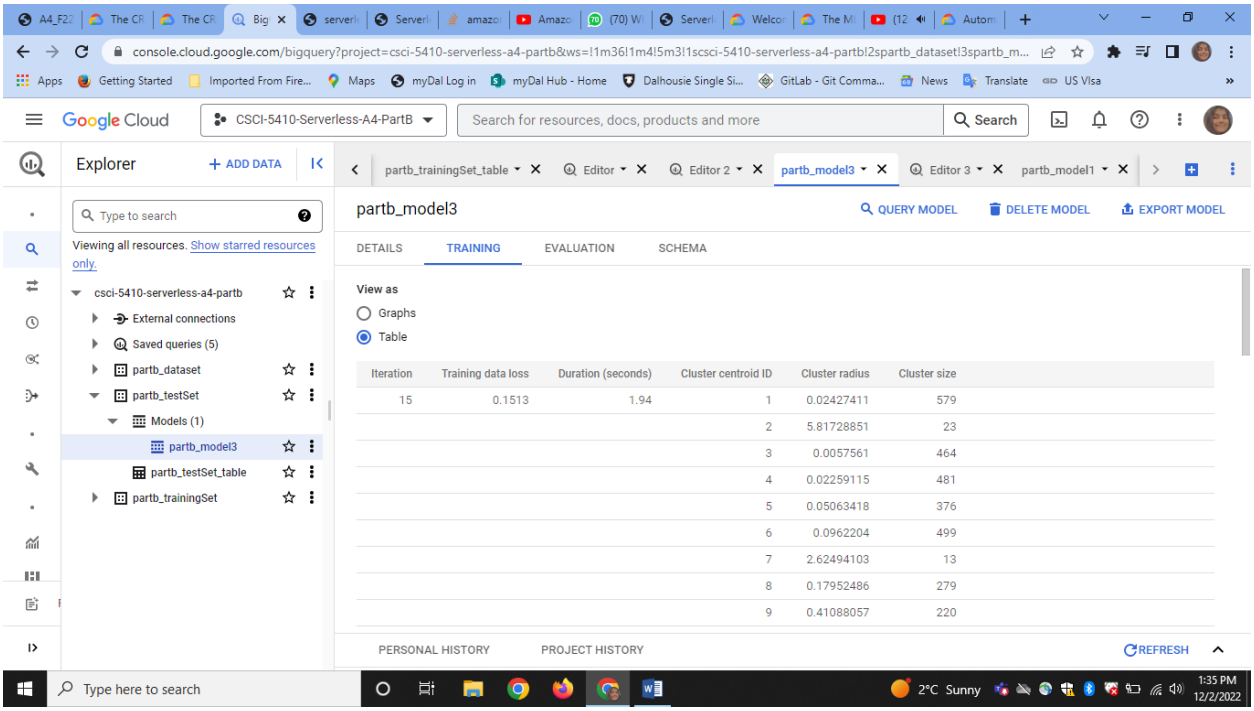


Fig 72: Training details for "partb_model3" – Table view

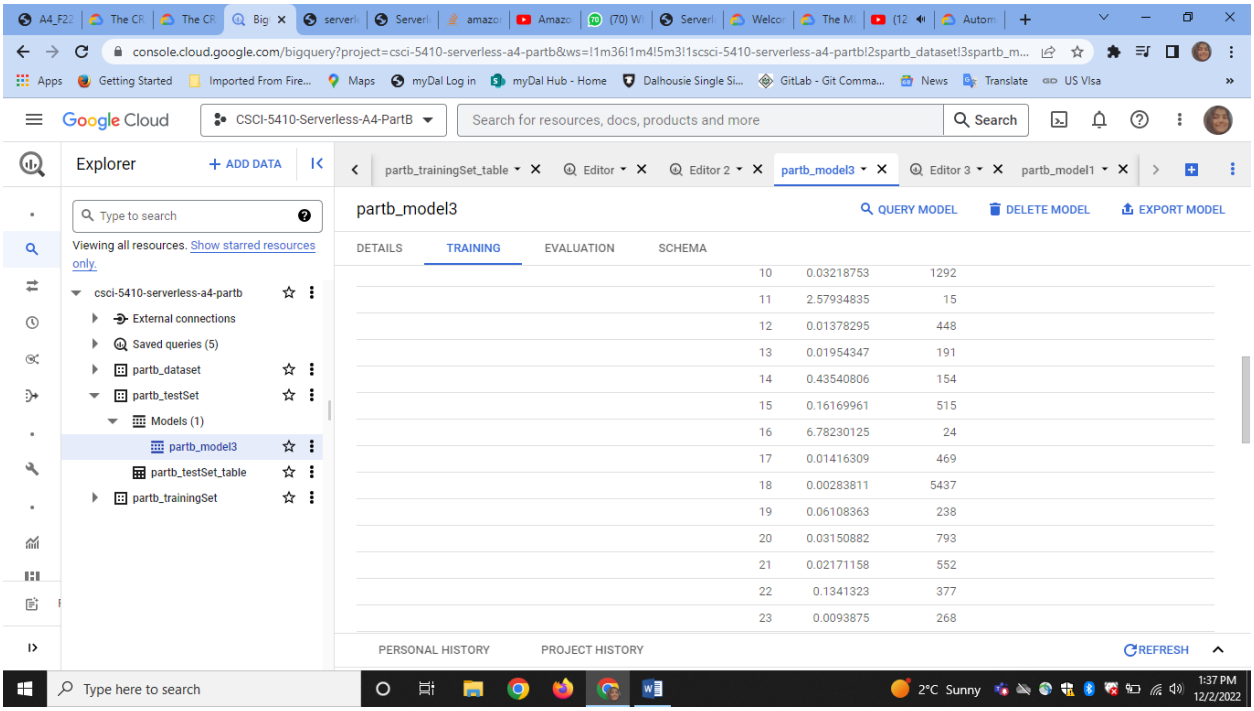


Fig 73: Training details for "partb_model3" – Table view

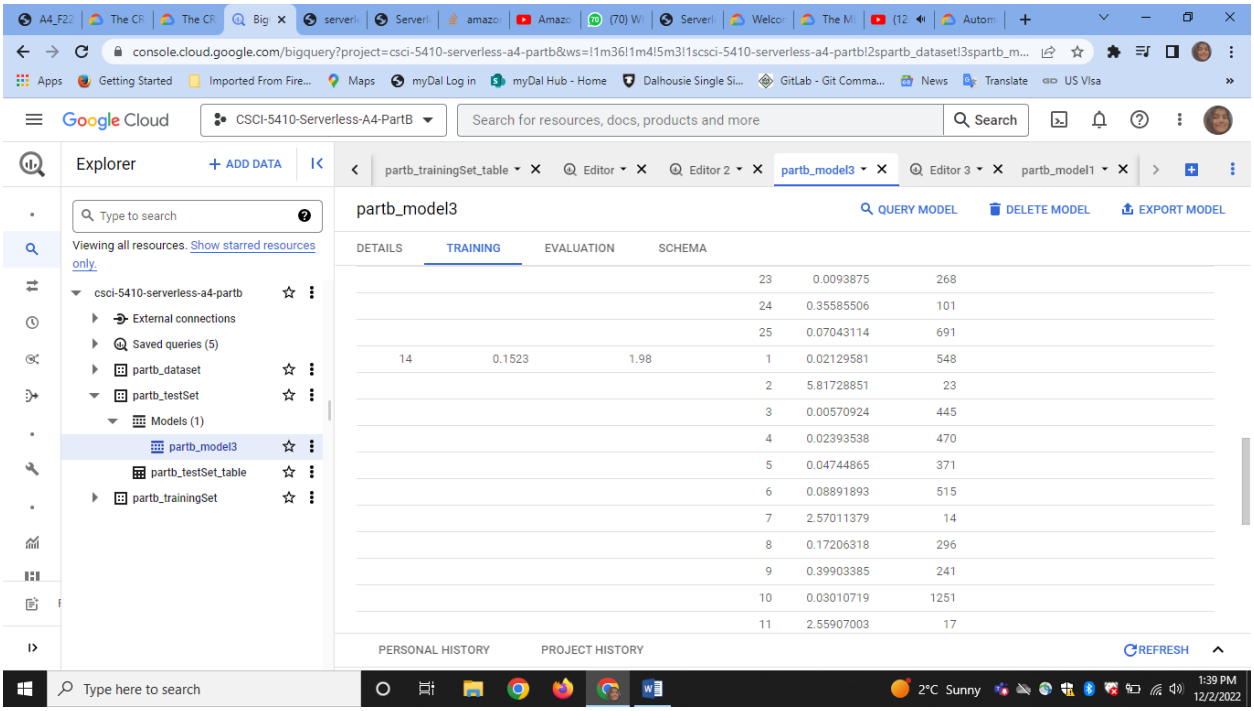


Fig 74: Training details for "partb_model3" – Table view

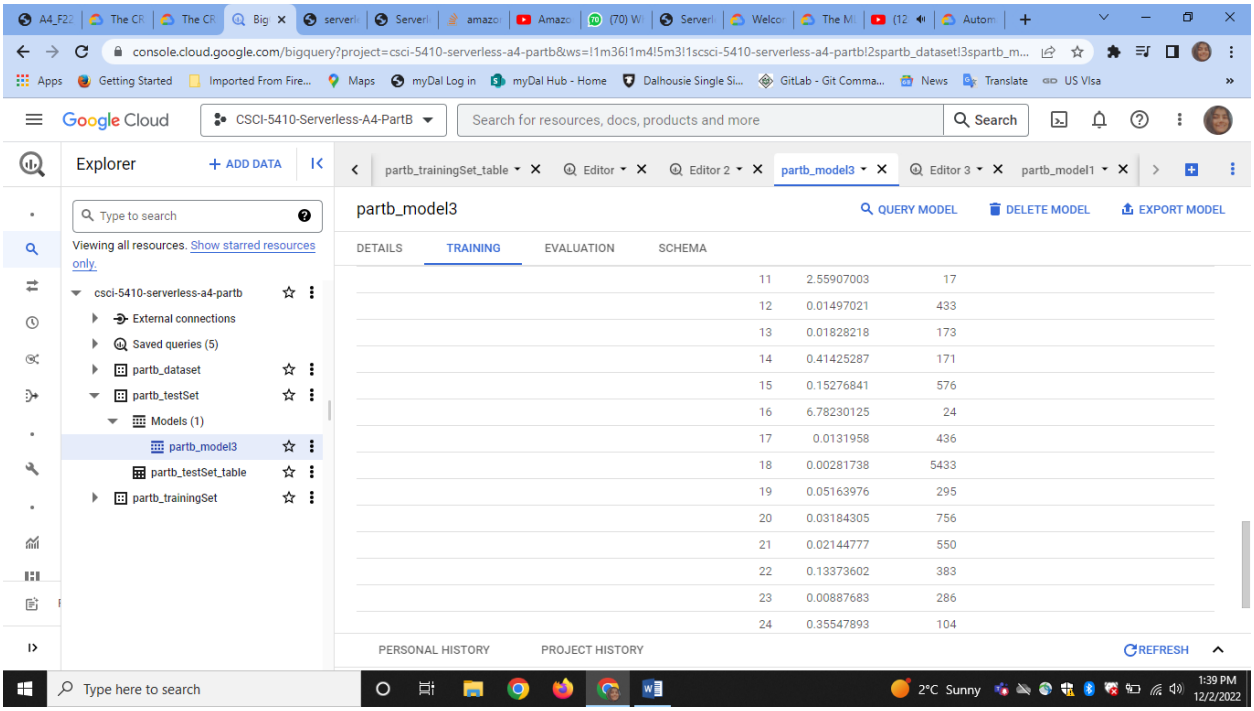


Fig 75: Training details for "partb_model3" – Table view

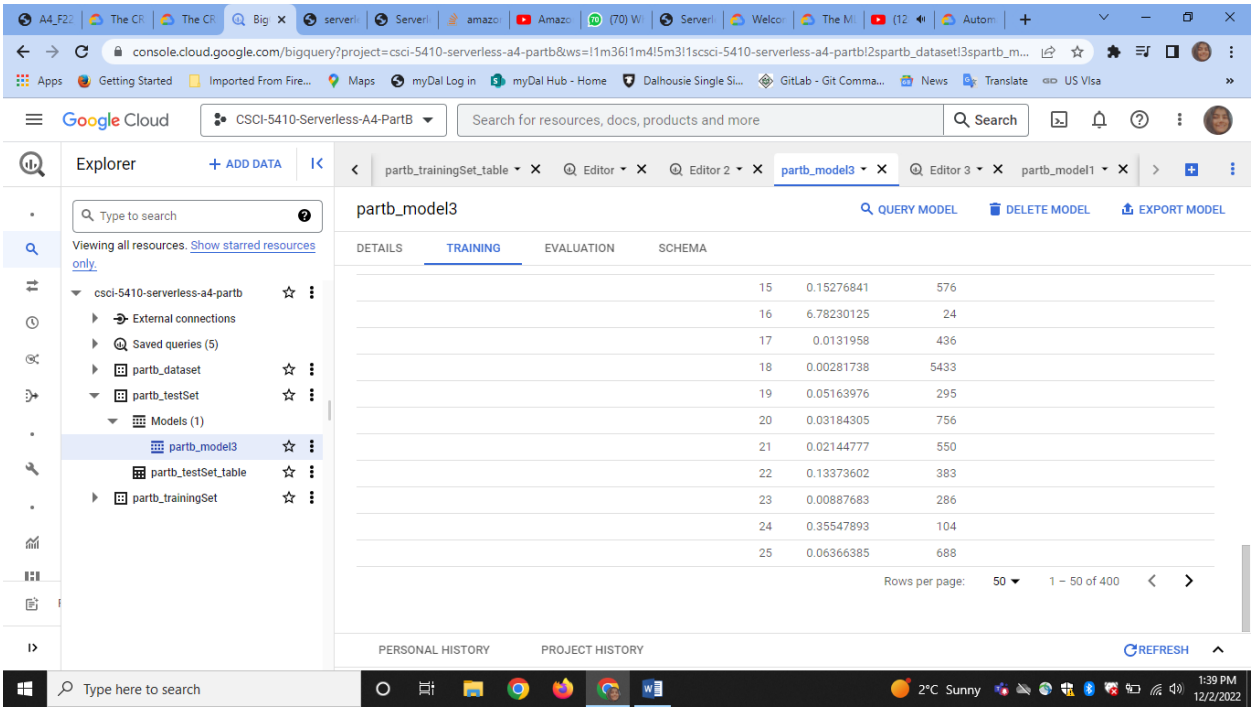


Fig 76: Training details for "partb_model3" – Table view

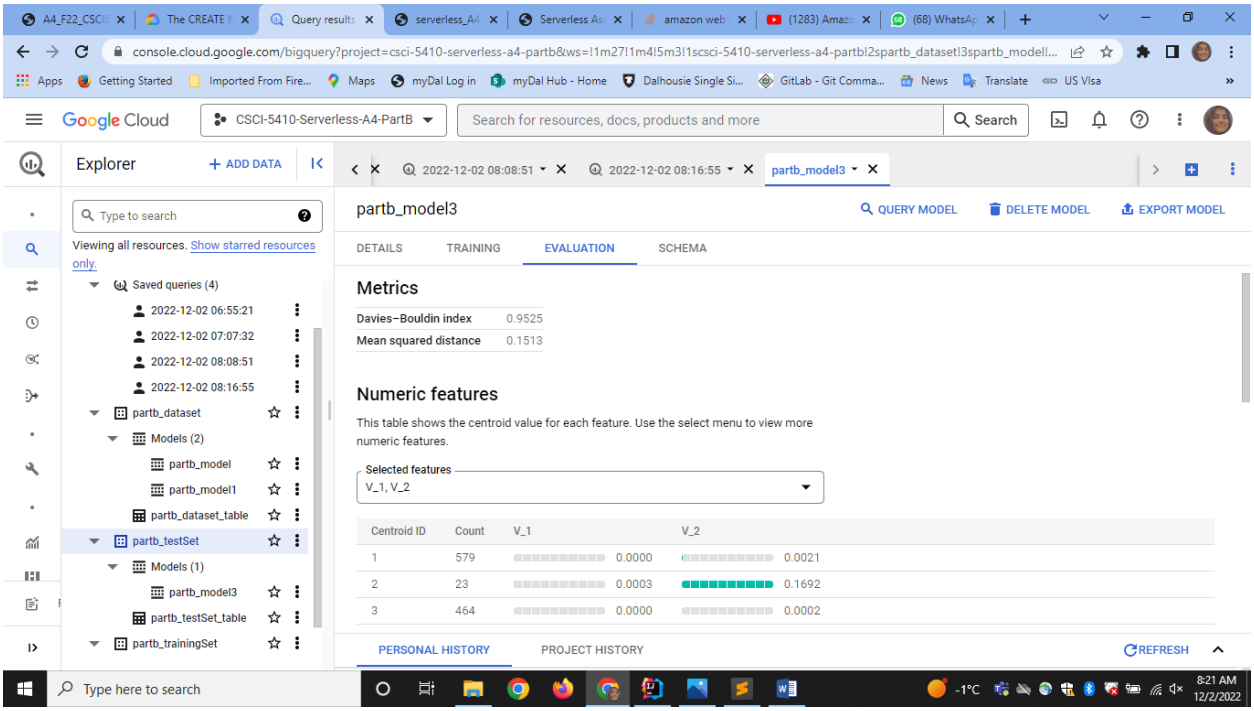


Fig 77: Evaluation details for "partb_model3"

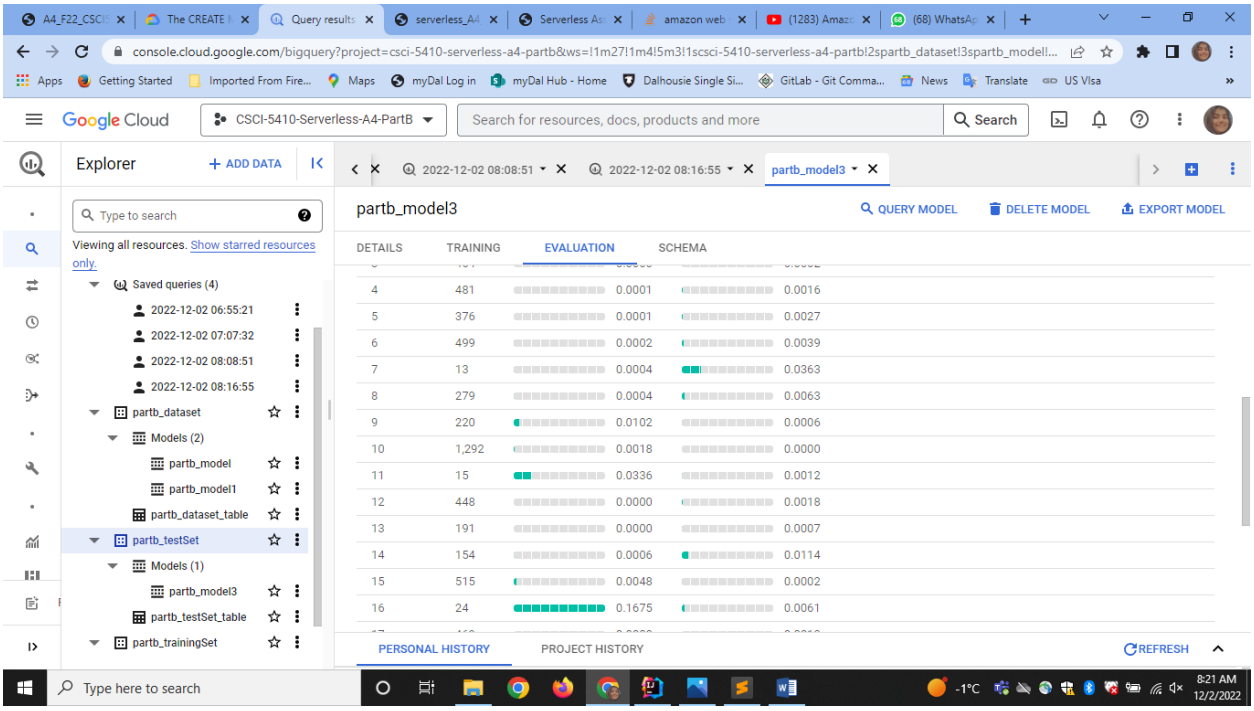


Fig 78: Evaluation details for "partb_model3"

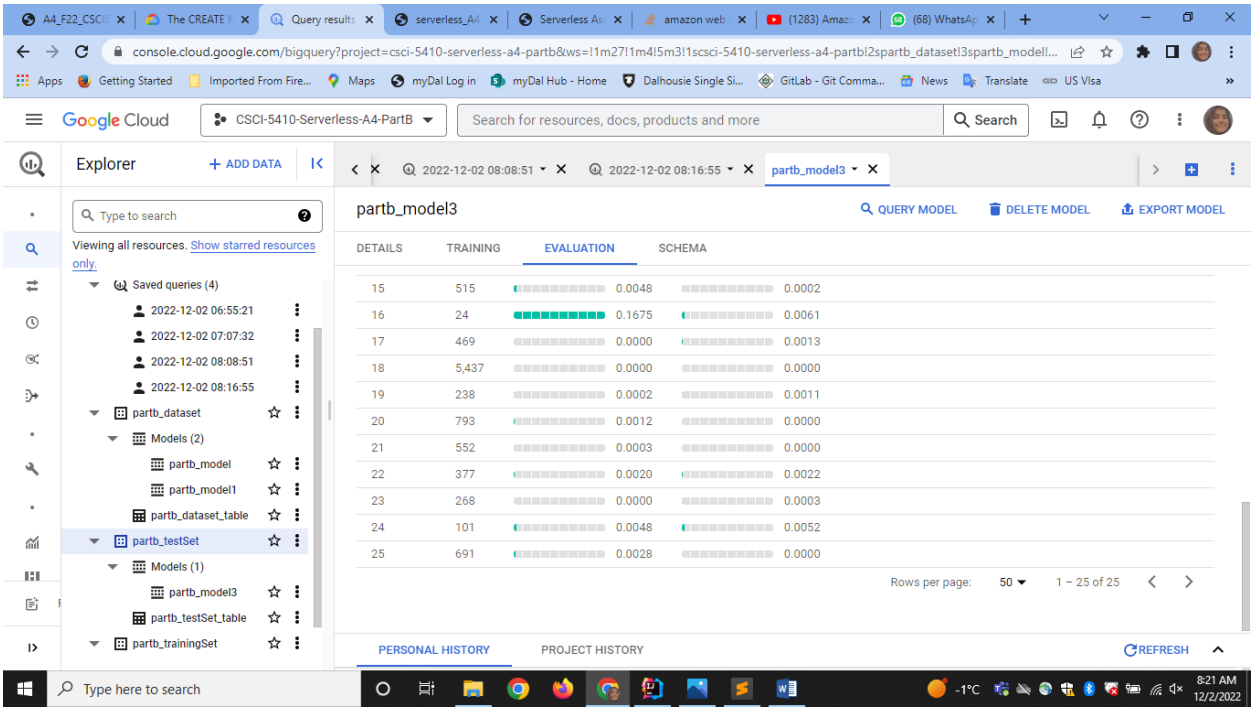


Fig 79: Evaluation details for "partb_model3"

Step 14:

- **Create a dataset with 75% dataset** as test set by filling the details as shown in the fig 80 and 81.
- The dataset "partb_trainingSet" created as shown in the fig 82.
- I used the "SDey_FTP_input_75PercentTrainingSet" dataset as shown in the fig 84.
- Now we need to add data. But in order to add data we need to create a table to upload file as shown in the fig 83, 84, 85 and 86.
- After clicking on the "create table" as shown in the fig 86, the partb_trainingSet_table is created.

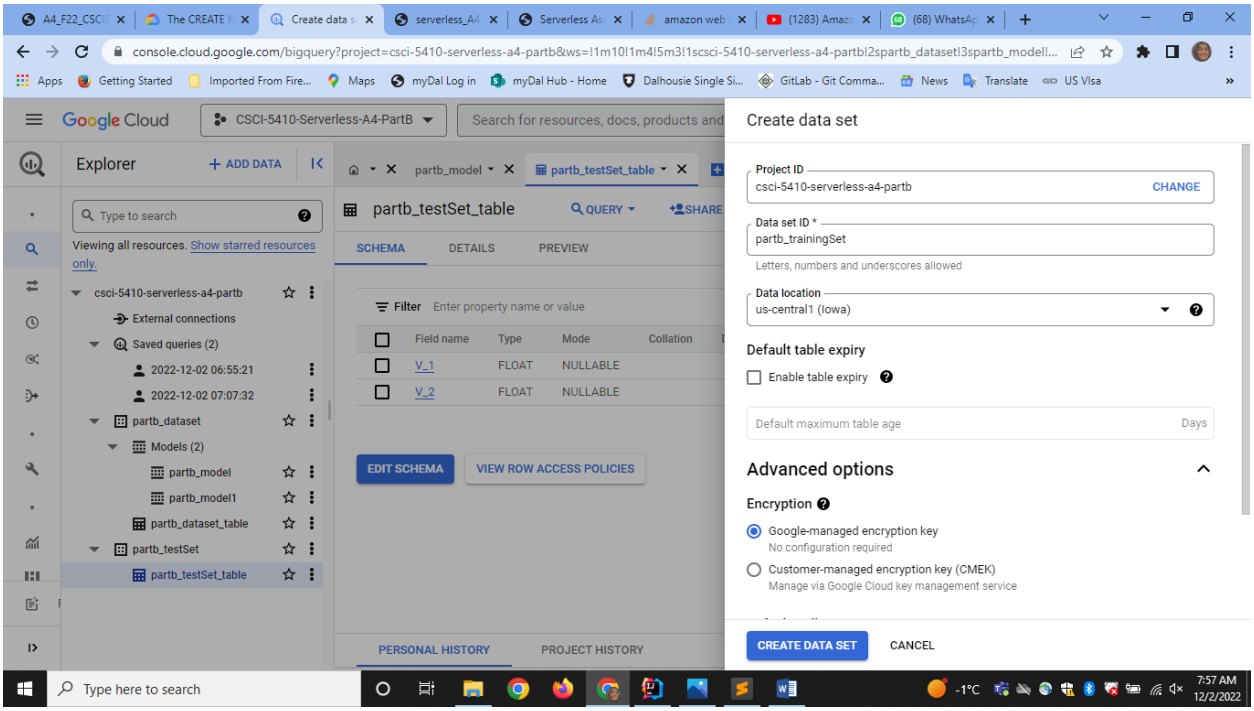


Fig 80: "Create data set" form

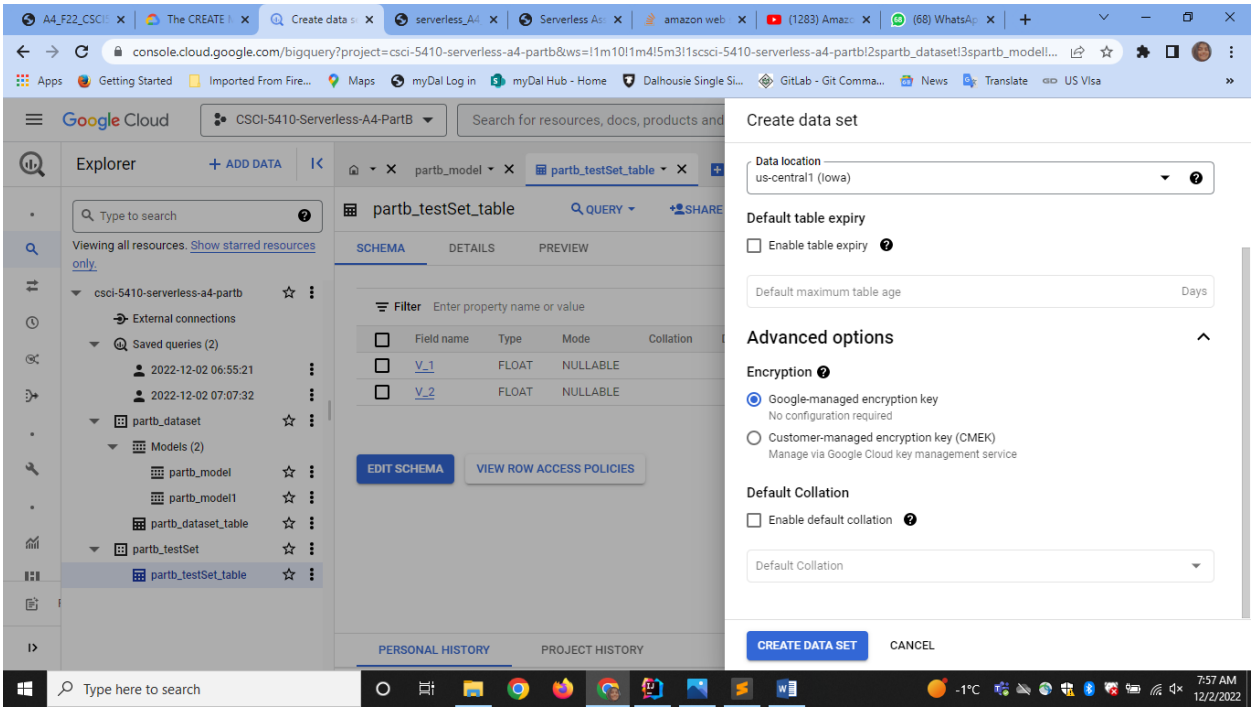


Fig 81: "Create data set" form

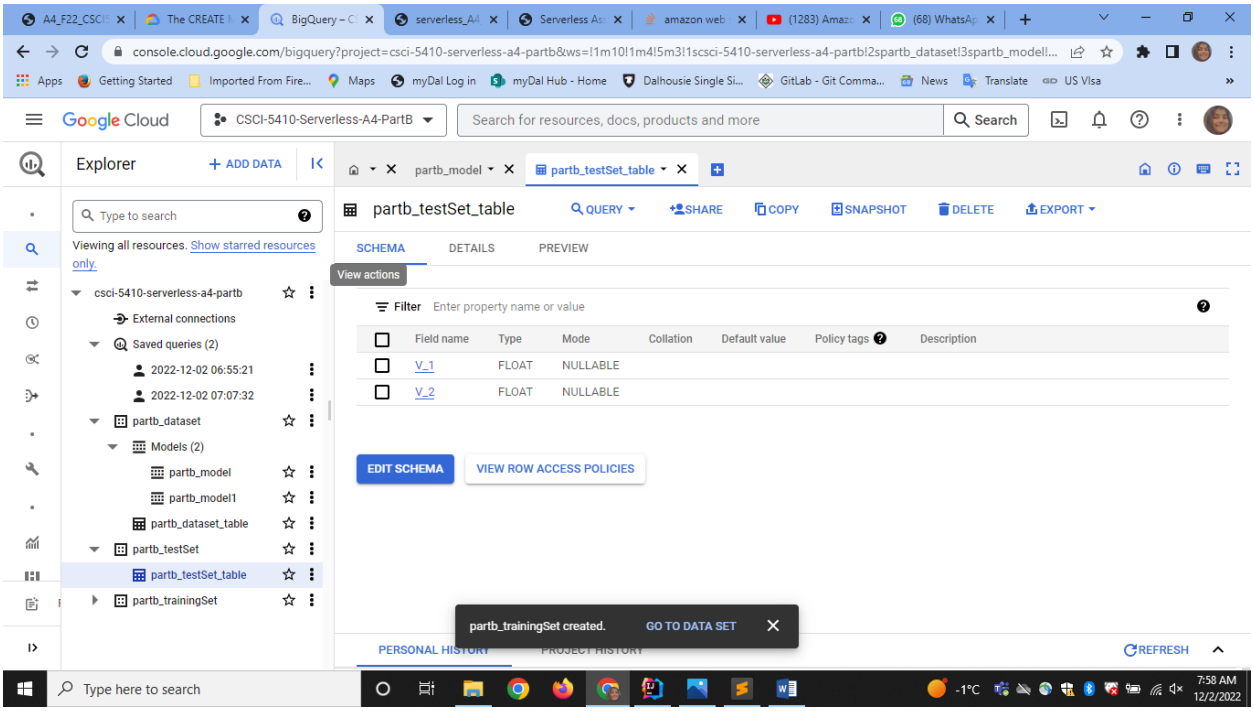


Fig 82: "partb_trainingSet" dataset created

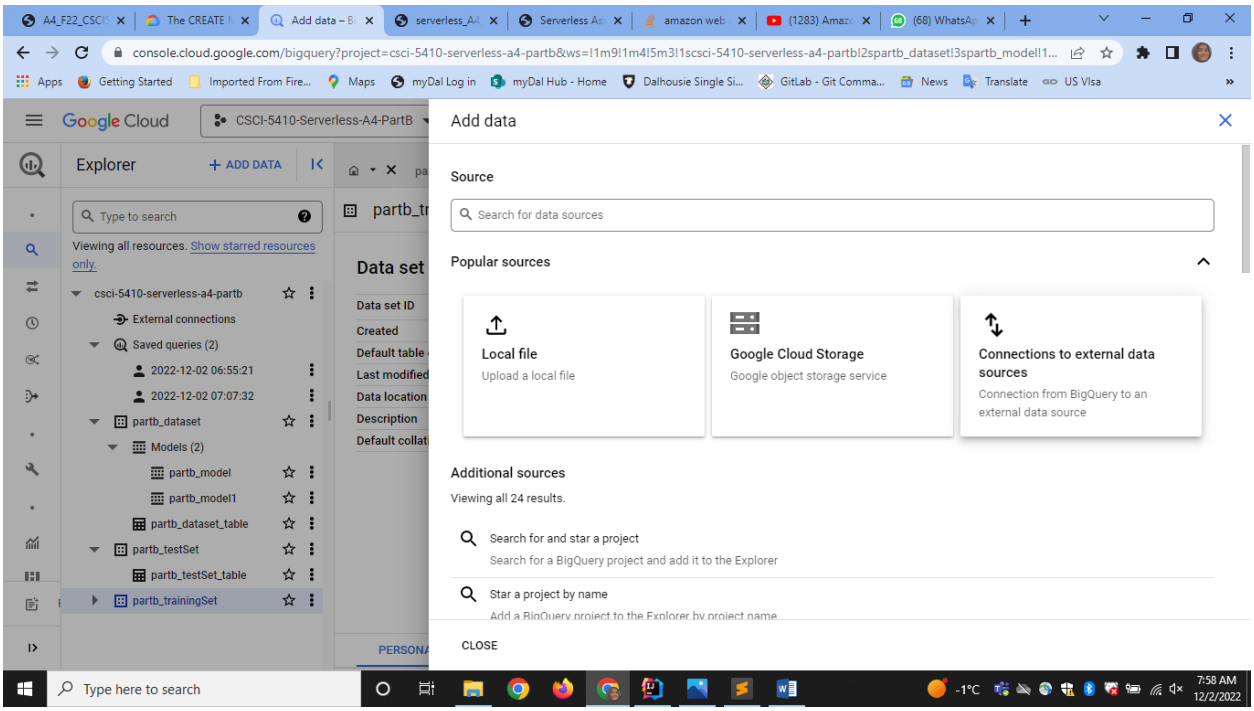


Fig 83: "Add data" form

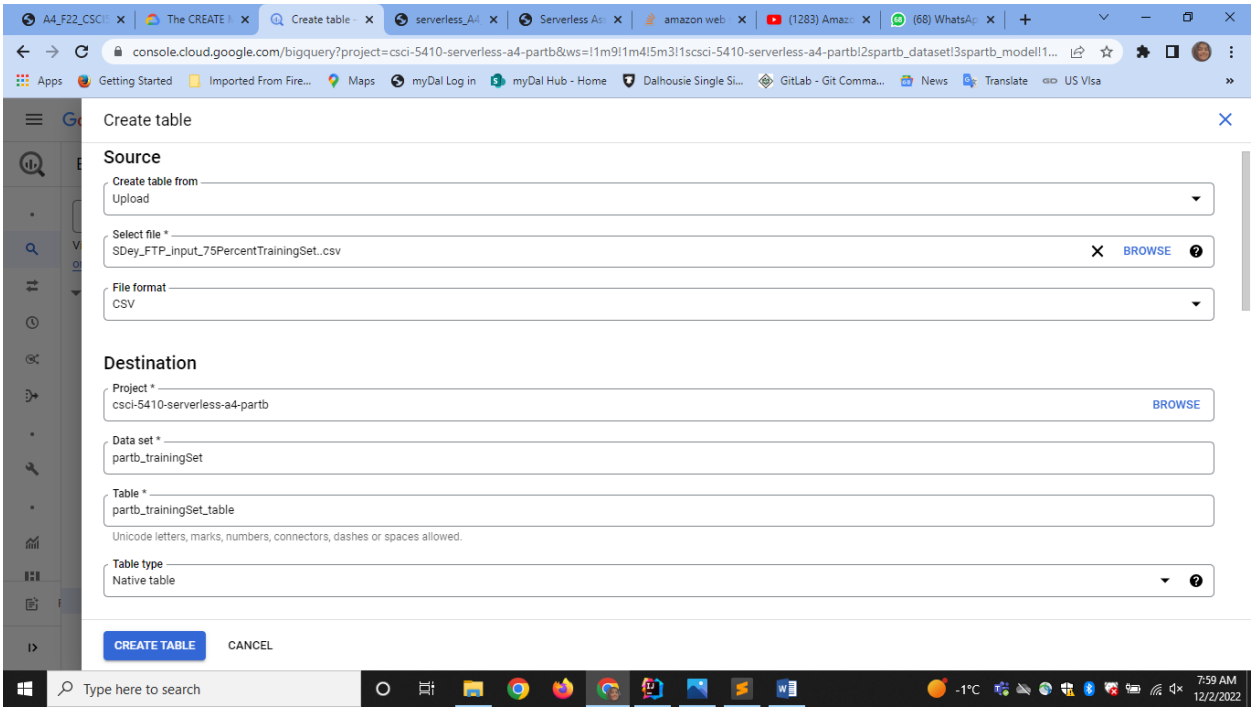


Fig 84: "Add data" form

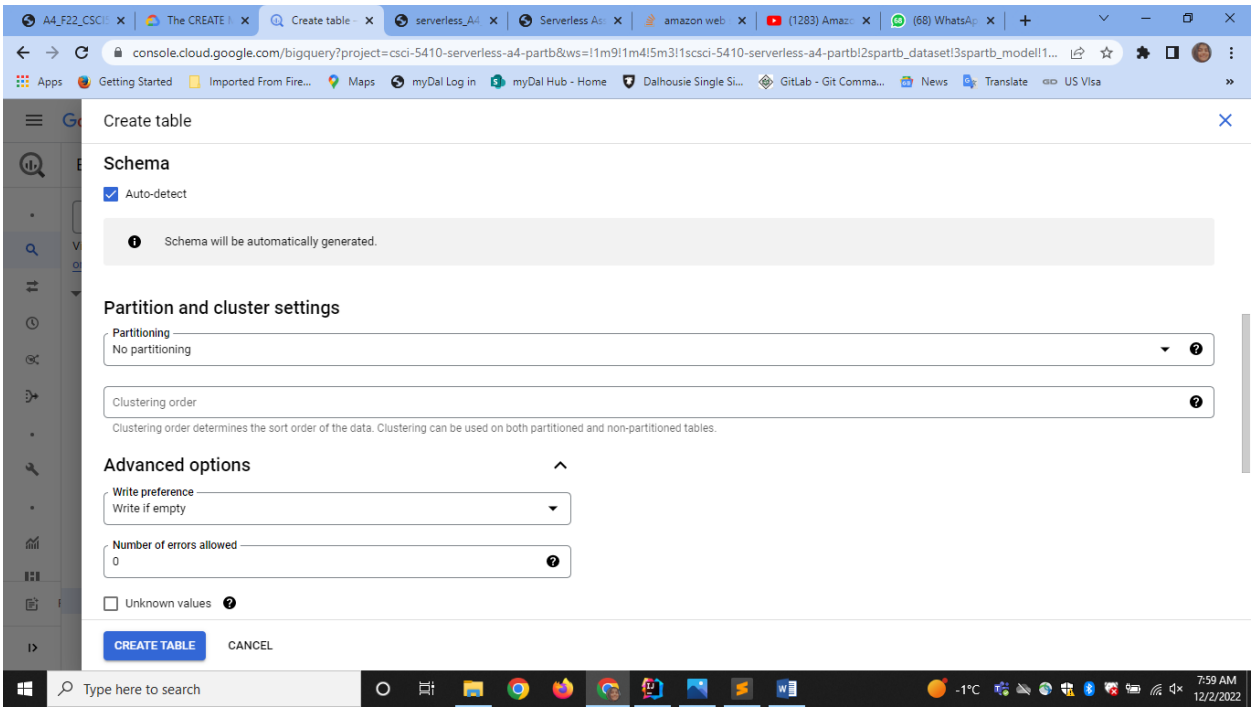


Fig 85: "Add data" form

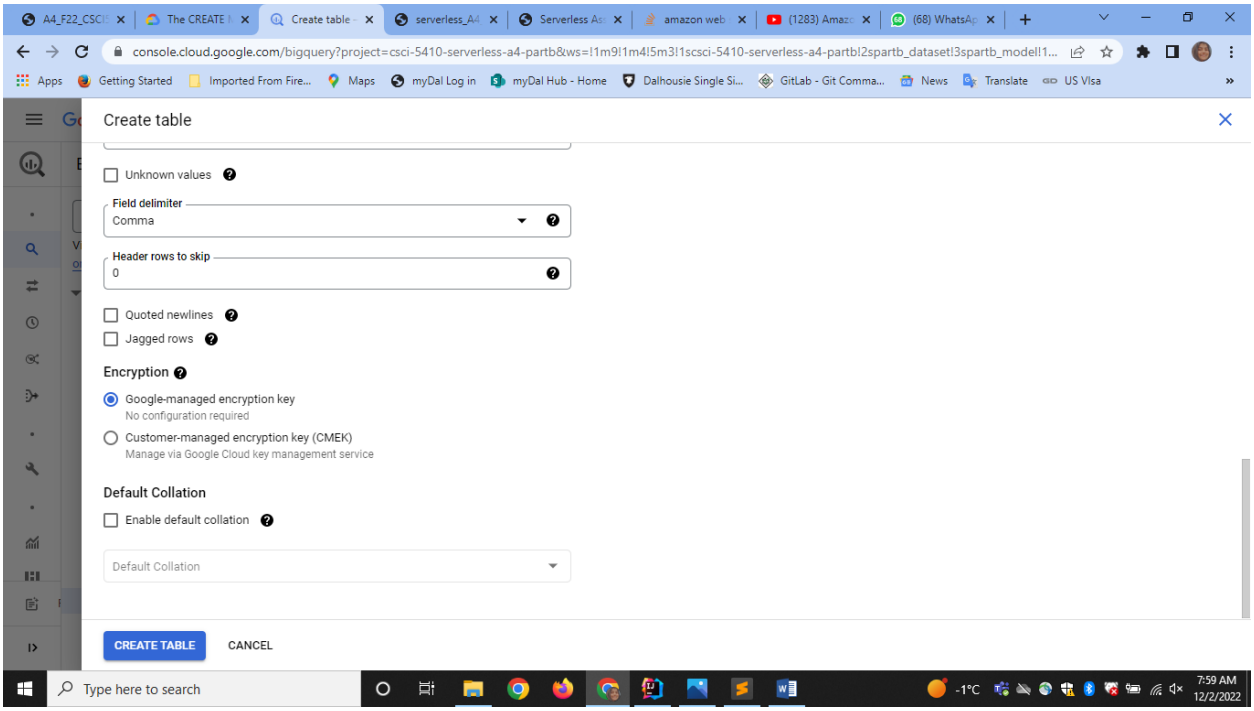


Fig 86: "Add data" form

Step 15:

- Keep the location same as the "partb_trainingSet" which is "us-central-1 (Iowa) as shown in the fig 87".
- I wrote a query to train the model as shown in the fig 88. The query creates a k-means model with 25 clusters using the "DISTANCE_TYPE" value of "EUCLIDEAN_DISTANCE" and standardize features [1].
- Fig 89 shows query running with the elapsed time.
- Fig 90 and 91 provides the model details which is been created after query completes its execution.
- Fig 92, 93, 94, 95, 96 and 97 gives the graphical view, table view of the results.
- Fig 98 and 99 represents evaluation details of the "partb_model4".

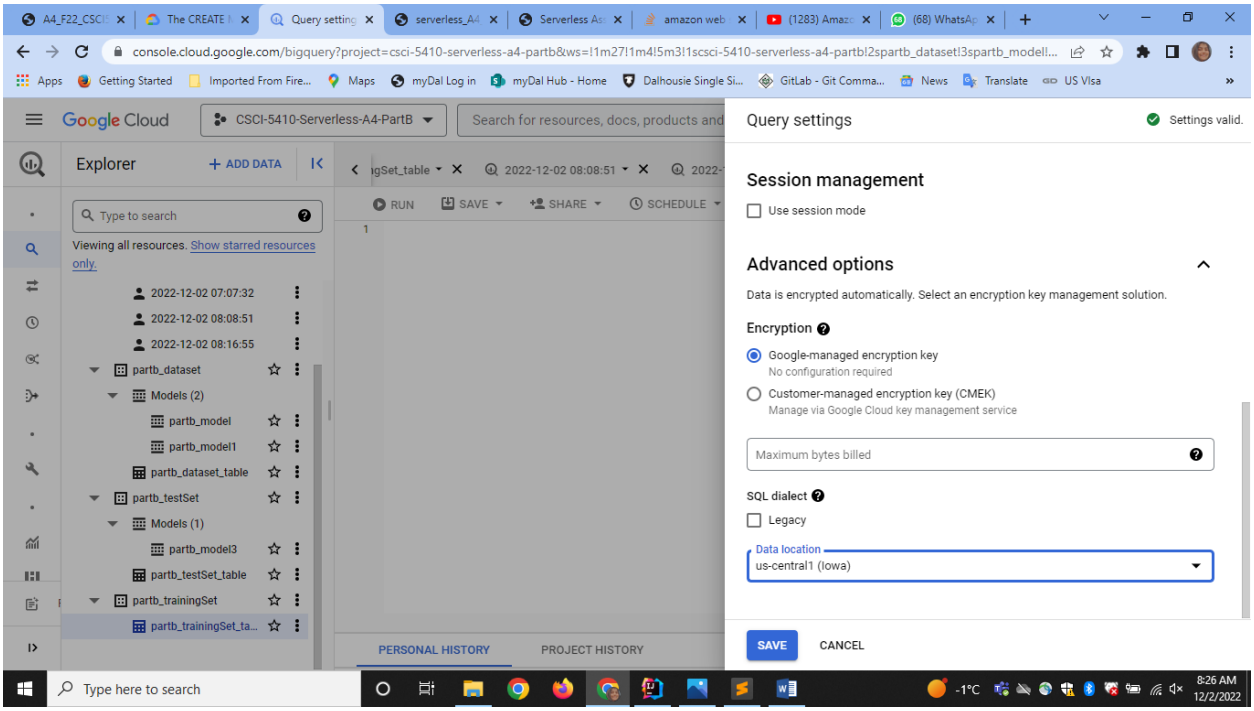


Fig 87: Query settings

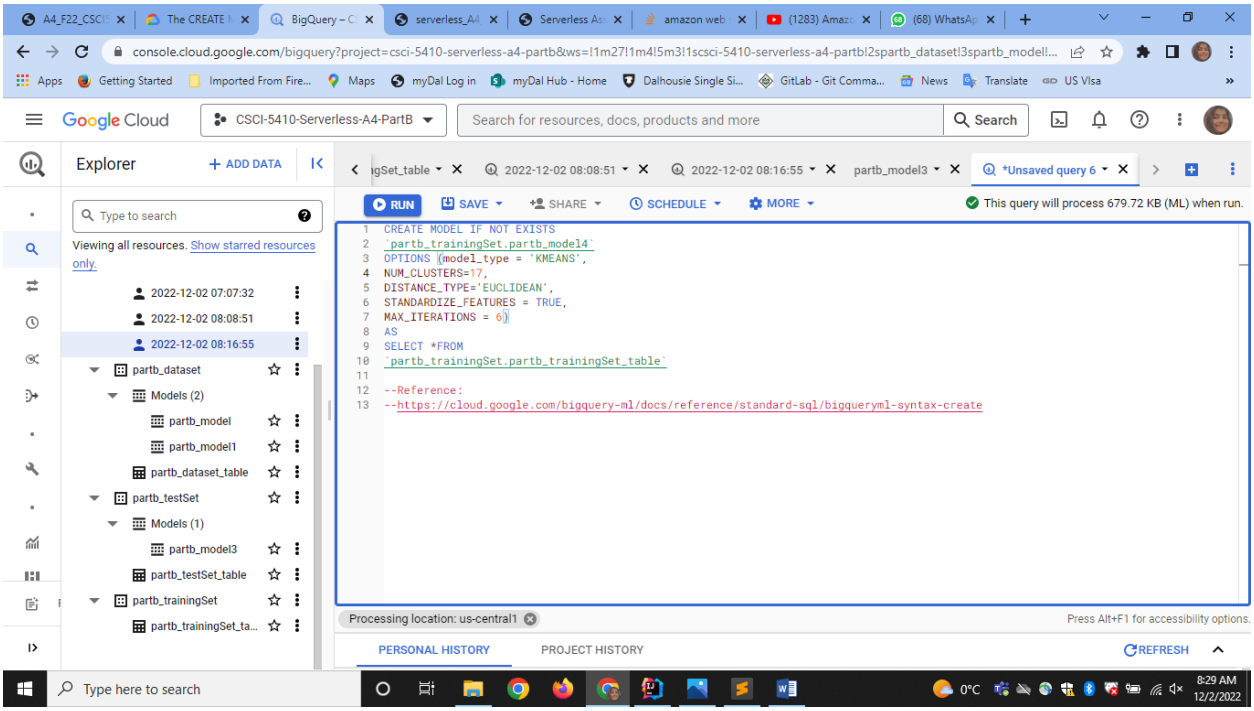


Fig 88: Query to create model for training set

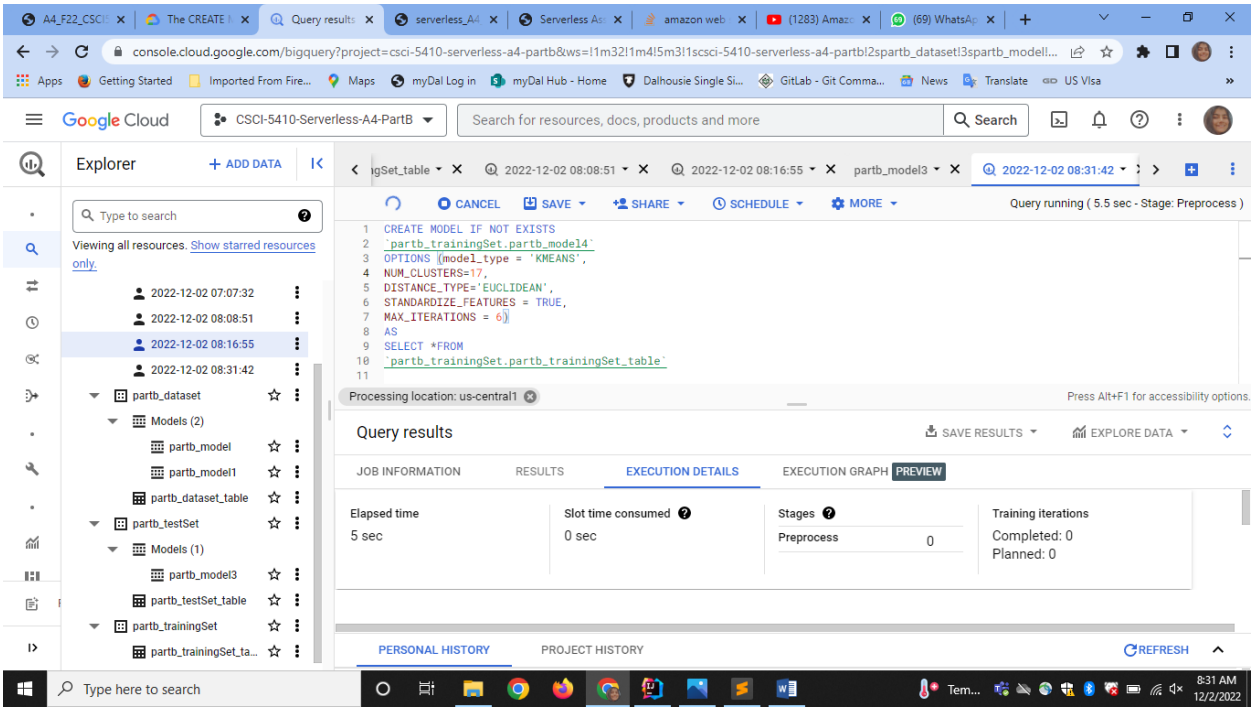


Fig 89: Query running

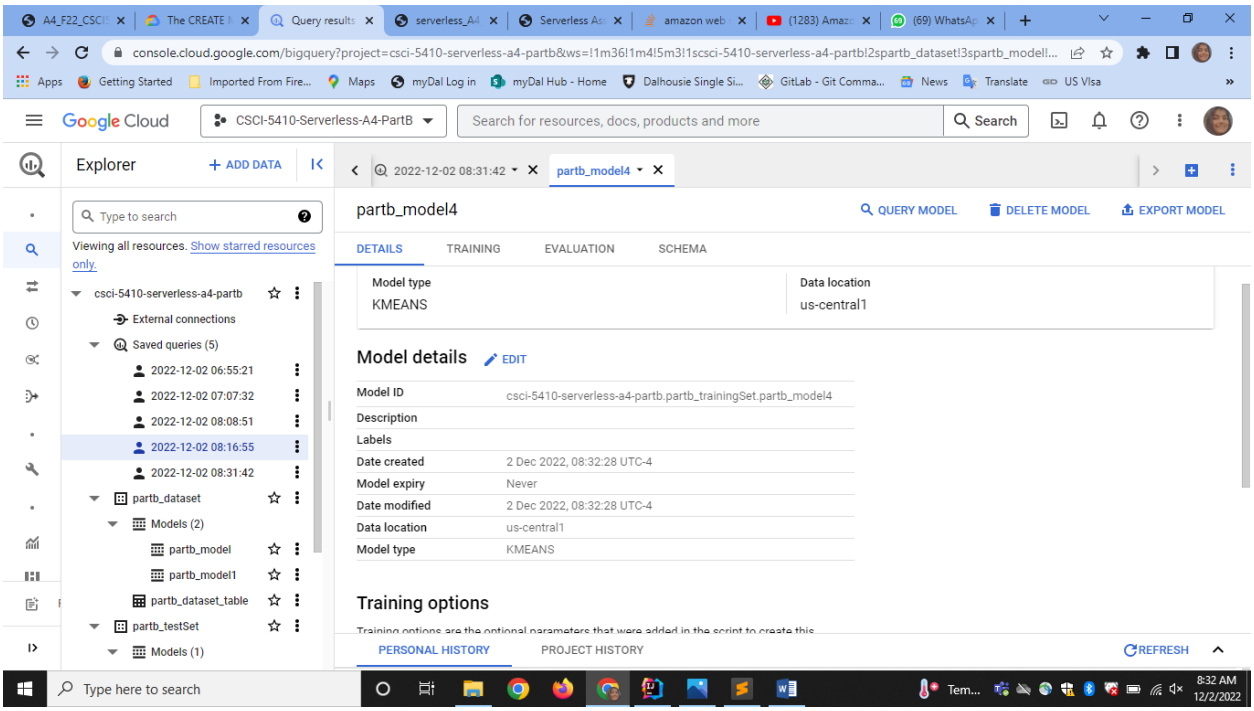


Fig 90: Model details for "partb_model4"

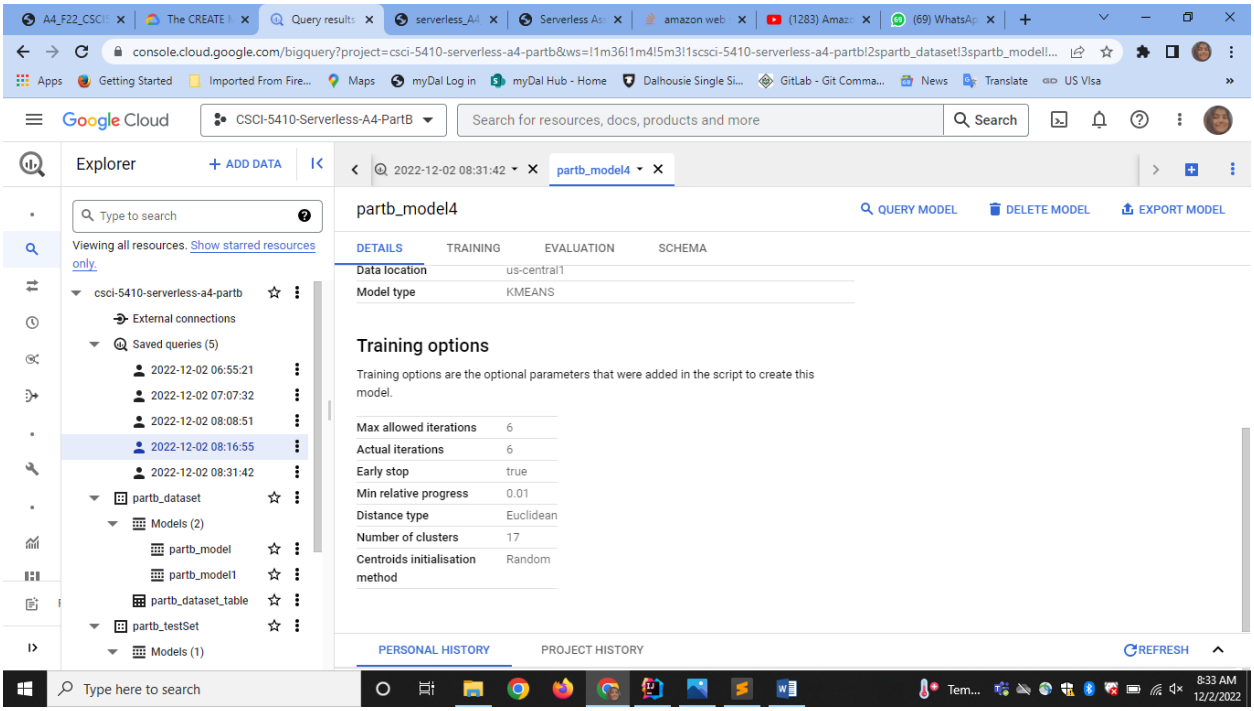


Fig 91: Model details for "partb_model4"

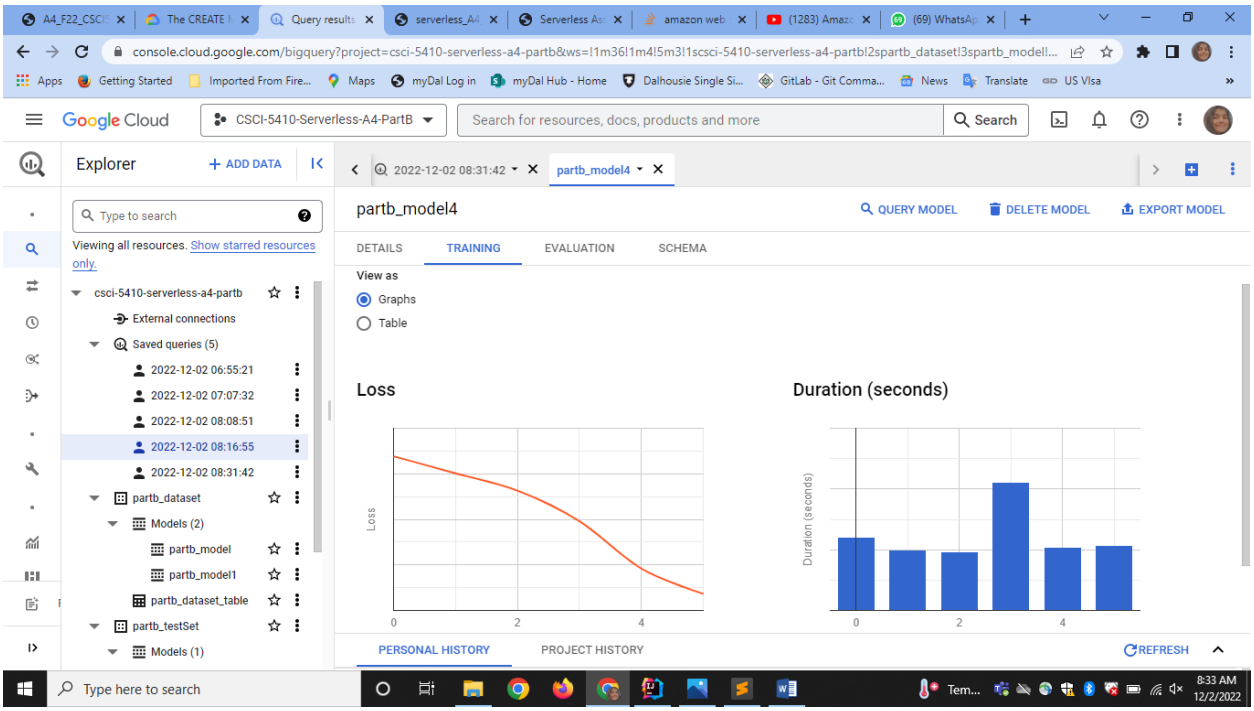


Fig 92: Training details for "partb_model4" – Graph view

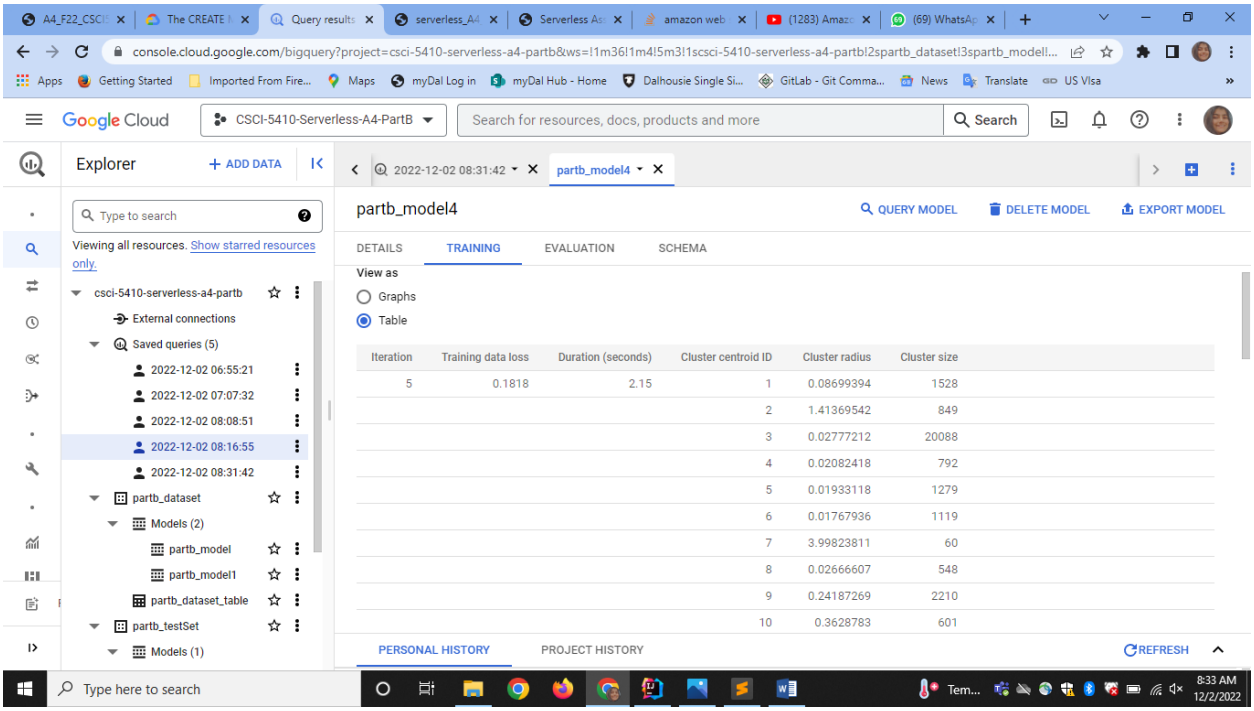


Fig 93: Training details for "partb_model4" – Table view

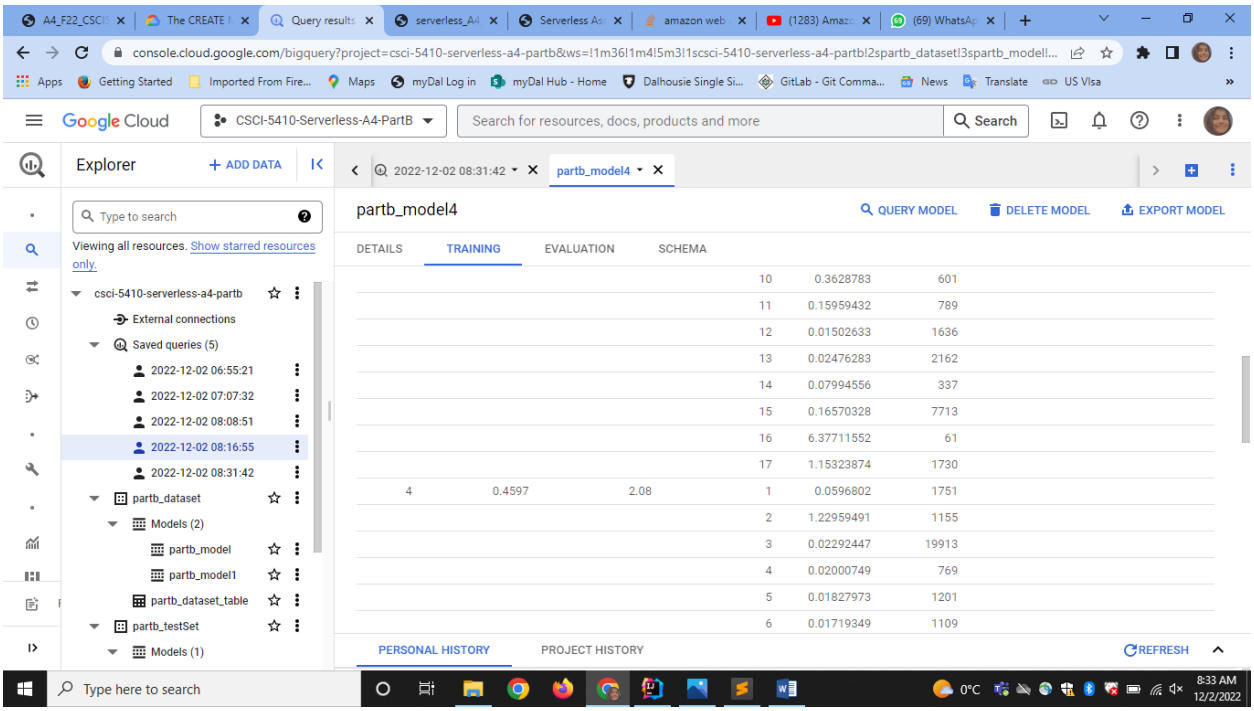


Fig 94: Training details for "partb_model4" – Table view

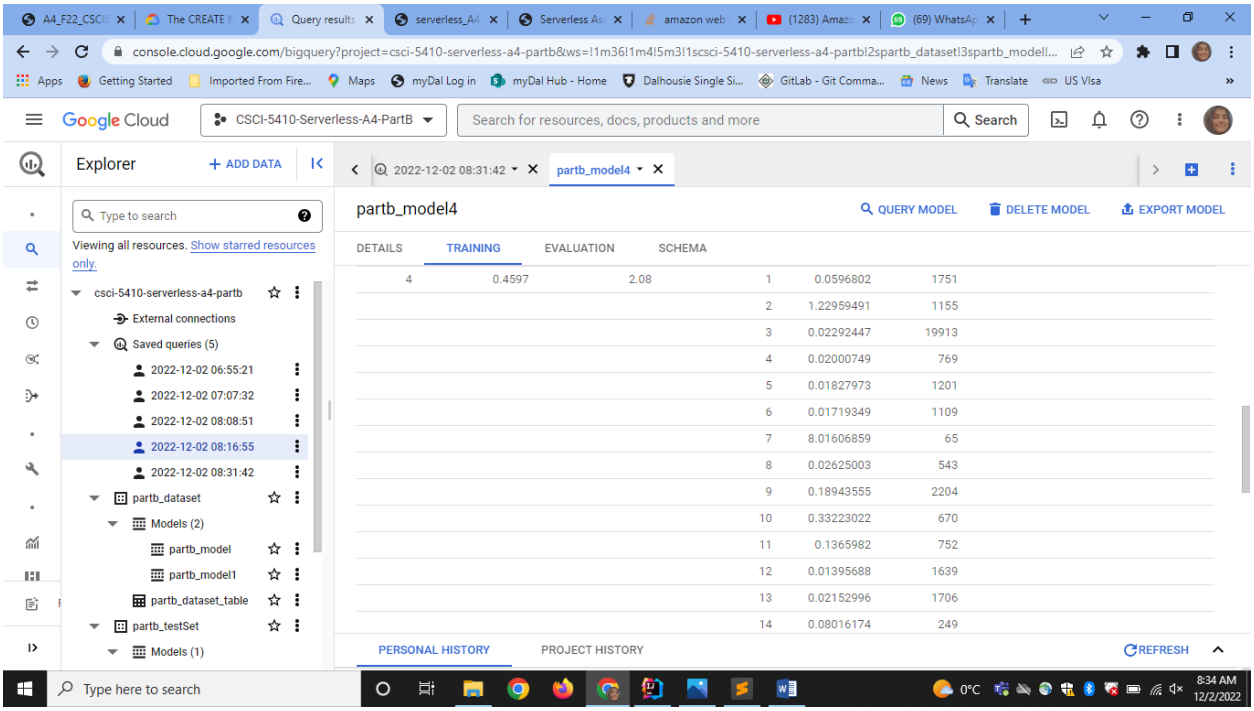


Fig 95: Training details for "partb_model4" – Table view

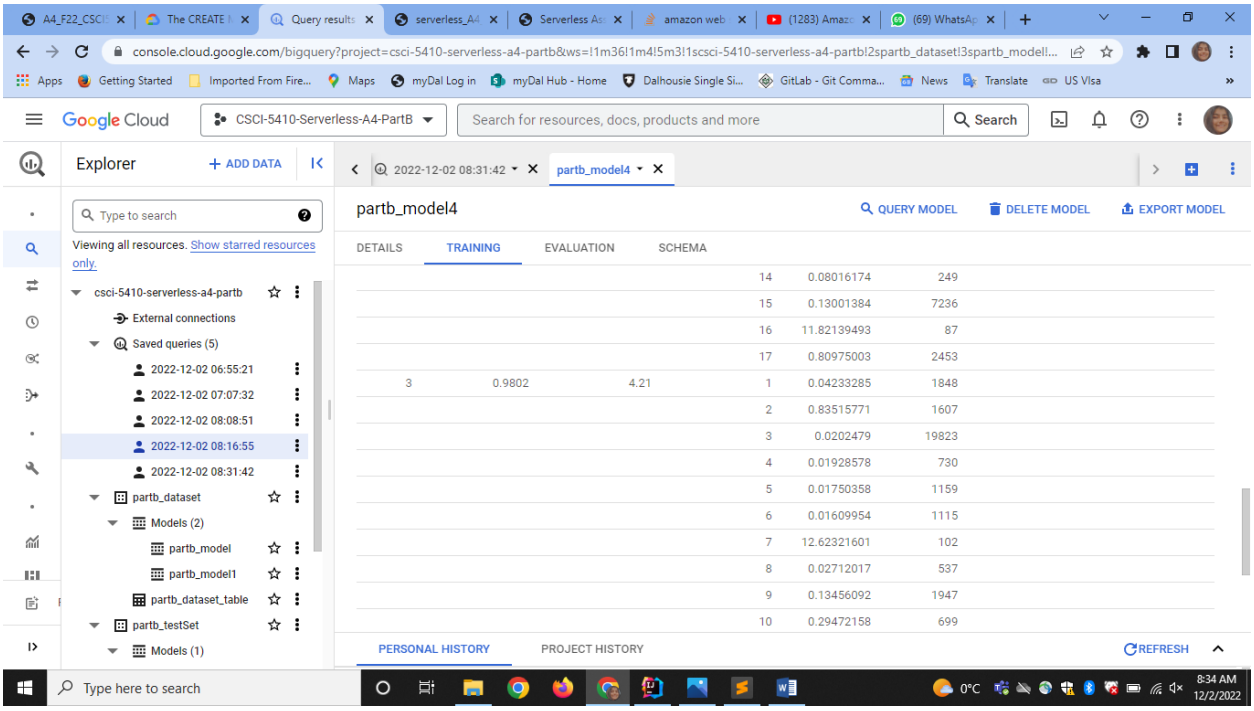


Fig 96: Training details for "partb_model4" – Table view

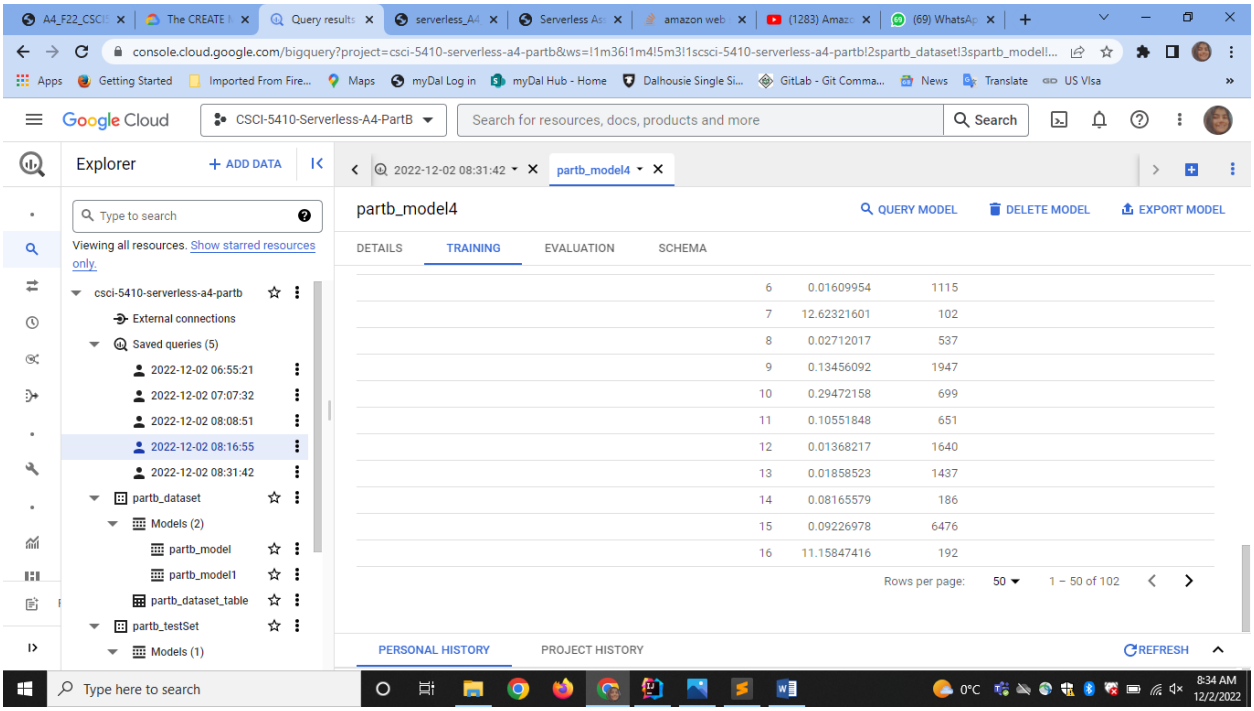


Fig 97: Training details for "partb_model4" – Table view

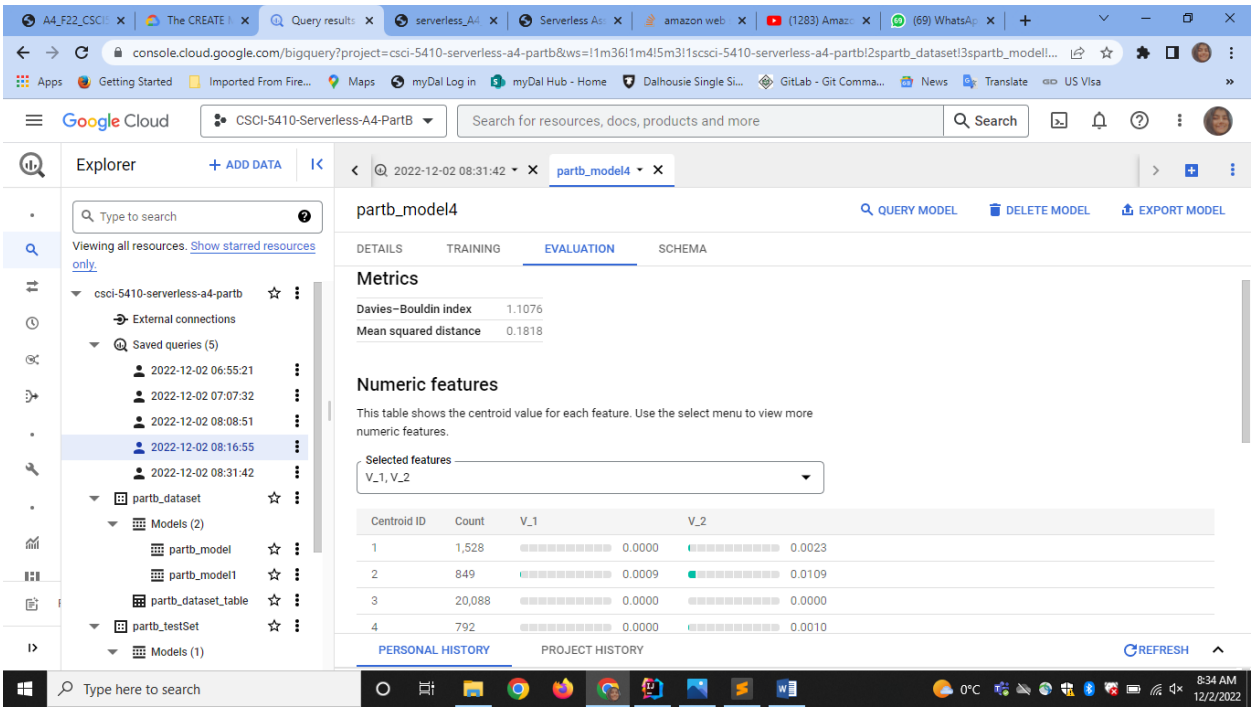


Fig 98: Evaluation details for "partb_model4"

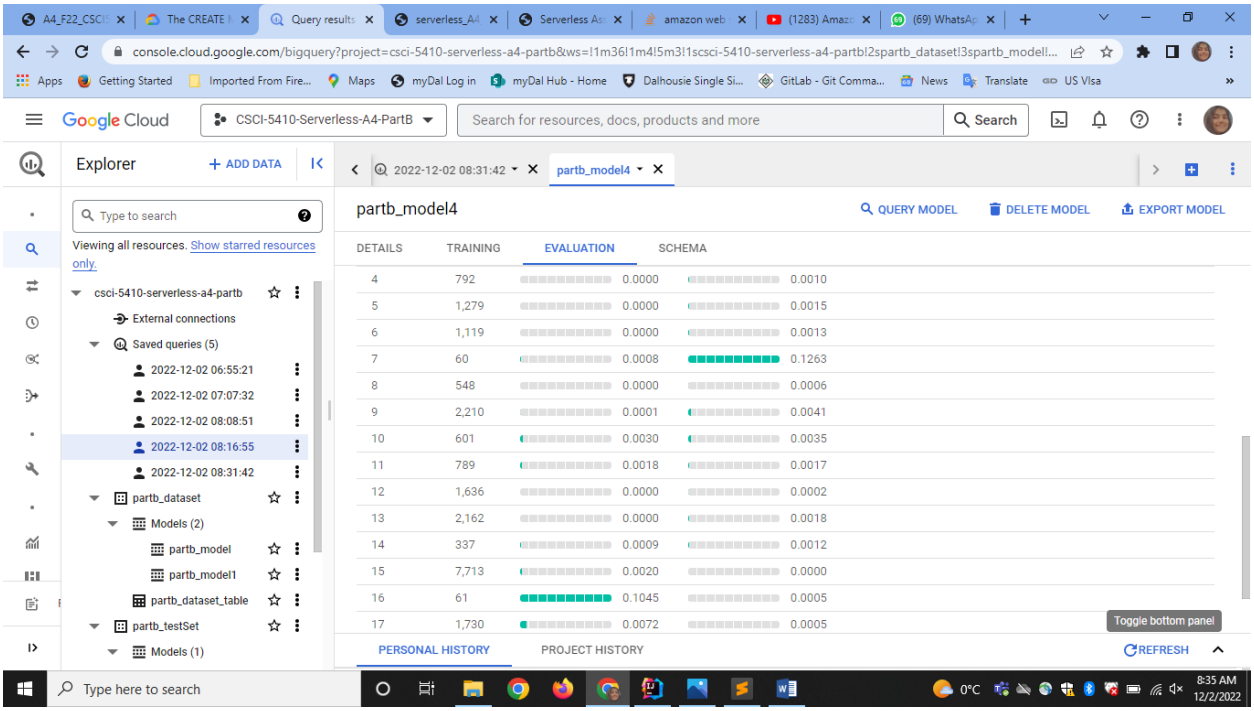


Fig 99: Evaluation details for "partb_model4"

Code/Query Script:

1) Creation of Model (partb_model) having dataset "SDey_FTP_input.csv"

```
CREATE MODEL IF NOT EXISTS
`partb_dataset.partb_model`
OPTIONS (model_type = 'KMEANS',
NUM_CLUSTERS=5,
KMEANS_INIT_METHOD='RANDOM')
AS
SELECT *FROM
`partb_dataset.partb_dataset_table`
```

2) Evaluate model using "SDey_FTP_input.csv" dataset

```
SELECT *
FROM
ML.TRAINING_INFO(MODEL `partb_dataset.partb_model`)
ORDER BY ITERATION
```

3) Creation of Model (partb_model1) having dataset "SDey_FTP_input.csv"

```
CREATE MODEL IF NOT EXISTS
`partb_dataset.partb_model1`
OPTIONS (model_type = 'KMEANS',
NUM_CLUSTERS=11,
KMEANS_INIT_METHOD='RANDOM',
DISTANCE_TYPE= 'EUCLIDEAN')
AS
SELECT *FROM
`partb_dataset.partb_dataset_table`
```

4) Creation of model using test dataset i.e. "SDey_FTP_input_25PercentTestSet.csv":

```
CREATE MODEL IF NOT EXISTS
`partb_testSet.partb_model3`
OPTIONS (model_type = 'KMEANS',
NUM_CLUSTERS=25,
KMEANS_INIT_METHOD='RANDOM',
DISTANCE_TYPE='EUCLIDEAN',
STANDARDIZE_FEATURES = TRUE)
AS
SELECT *FROM
`partb_testSet.partb_testSet_table`
```

5) Creation of model using training dataset "SDey_FTP_input_75PercentTestSet":

```
CREATE MODEL IF NOT EXISTS
`partb_trainingSet.partb_model4`
OPTIONS (model_type = 'KMEANS',
NUM_CLUSTERS=17,
DISTANCE_TYPE='EUCLIDEAN',
STANDARDIZE_FEATURES = TRUE,
MAX_ITERATIONS = 6)
AS
SELECT *FROM
`partb_trainingSet.partb_trainingSet_table`
```

Reference:

[1] "The CREATE MODEL statement for K-means models," *Google Cloud*. [Online]. Available: https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-kmeans. [Accessed: 02-Dec-2022].