ASSIGNMENT 1: PART A

TITLE: Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach

ACM Reference:

[1] P. Vahidinia, B. Farahani and F. S. Aliee, "Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2022.3165127.

Summary:

It became necessary to have computing resources, storage, and network-based technologies to manage growing data volumes because information technology has revolutionised the world [1]. The latest cloud computing architecture, known as serverless computing, helps to lower the cost and complexity of system development, which supports corporate growth and offers a variety of services to implement software maintenance issues [1]. The cold start delay, which is seen as a delay in setting up the environment in which functions are executed, is one of the most important challenges of serverless computing discussed in the study [1]. To lessen the impact of the cold start delay, many techniques are employed, such as using a pool of warm containers, but they are ineffective due to memory waste [1]. Therefore, we propose two methods for keeping the containers warm according to the function invocations over time [1].

The first technique is "Reduce cold start delay," and it uses the OpenWhisk platform to pre-warm the stem cell containers depending on memory and programming language [1]. Different platforms, such as SAND, which separates applications at different application levels, and SOCK, which interacts with the Linux kernels used in containers, accomplish reductions of the container's preparation time and cold start delay [1]. In addition, prebaking makes use of the snapshots made for previous functions [1]. Another approach to reduce function occurrence delays is "Reduce Cold Start Occurrence" [1]. To achieve this, the container should be prepared prior to the function request [1]. In some platforms, notably AWS Lamba, containers are halted for a short period of time known as the idle-container window, which reduces the delay because it takes longer to build a new container than it does to restart the existing one [1]. The Openwhisk platform is made flexible to forecast future invocation times utilising machine learning methods employing a two-layer approach [1]. The first layer's goal is to have a larger window, which decreases cold start delays and uses more resources [1]. The second layer, on the other hand, uses Long Short Term Memory (LSTM) to predict how many pre-warmed containers are needed for future use and tries to delay the number of requests [1].

The evaluation was performed for the Cold Start Reduction Layer, in which the test datasets included 200 requests [1]. The Openwhisk platform's idle-container window uses the proposed Reinforcement Learning method to recognise the entry pattern for invocations [1]. Three different hourly average invocation patterns were displayed (5, 10, and 20) [1]. The findings indicated that 73% of invocations have an idle-container window of less than 10 minutes, indicating that a container typically does not need to keep warm for up to 10 minutes [1]. Additionally, the cumulative distribution functions calculate cold start occurrences and they found that 75% of invocations have fewer than 4 cold starts during an hour [1]. This demonstrates that there is no improvement in the cold start delays because of memory control [1]. However, the most crucial factor is memory utilisation, which improved by 11.11%, 12.73%, and 4.05%, respectively, for average invocations per hour of 5, 10, and 20 [1]. The end findings show that 56 out of 106 invocations were properly predicted, yet the Openwhisk platform only carried out 32 requests on pre-warmed containers, showing a clear improvement of 22.65% in the approach's ability to carry out invocations on pre-warmed containers [1]. One of the biggest problems with serverless computing, which has an impact on performance and user satisfaction, is cold start delay [1]. Reducing memory utilisation and cold start latency must therefore be balanced [1].

ASSIGNMENT 1: PART B - AWS S3 Experiment

Gitlab Link: https://git.cs.dal.ca/umatiya/csci5410-f23-b00899642.git

Steps followed to complete AWS S3 Experiment:

Step 1:

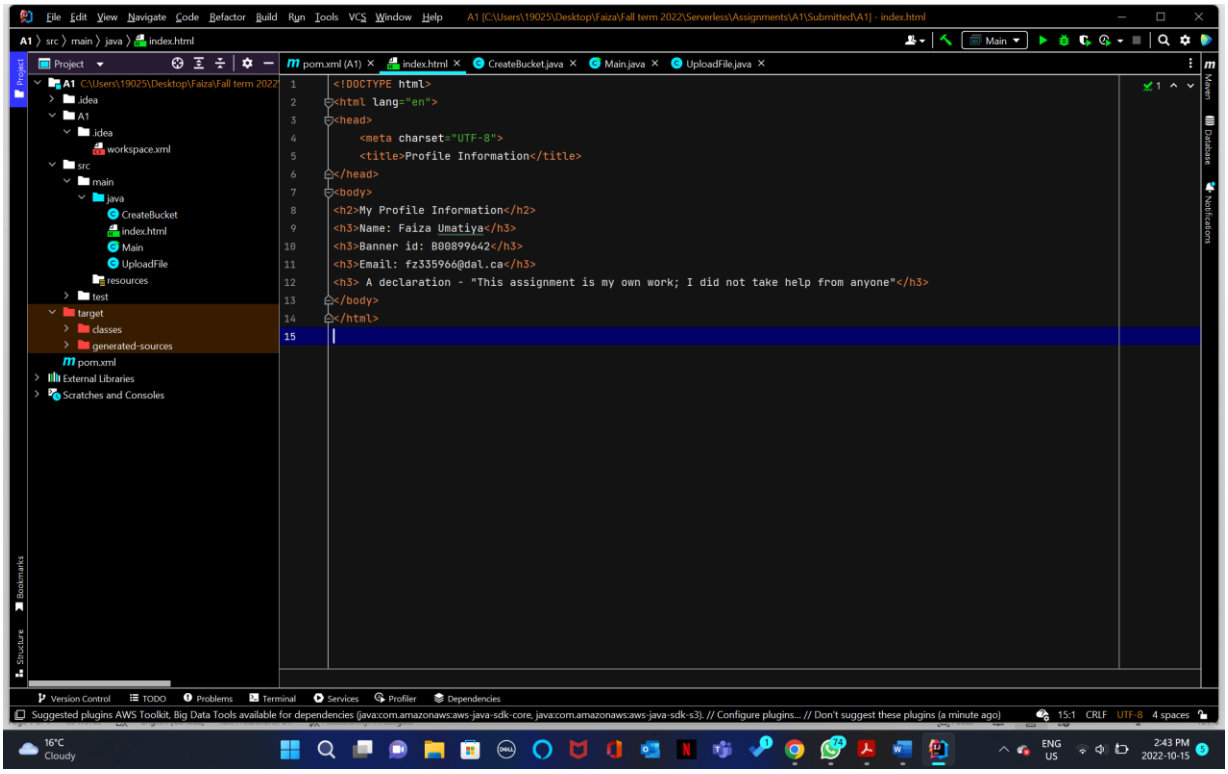


Fig 1: Code for "index.html" file in Intellij JAVA IDE

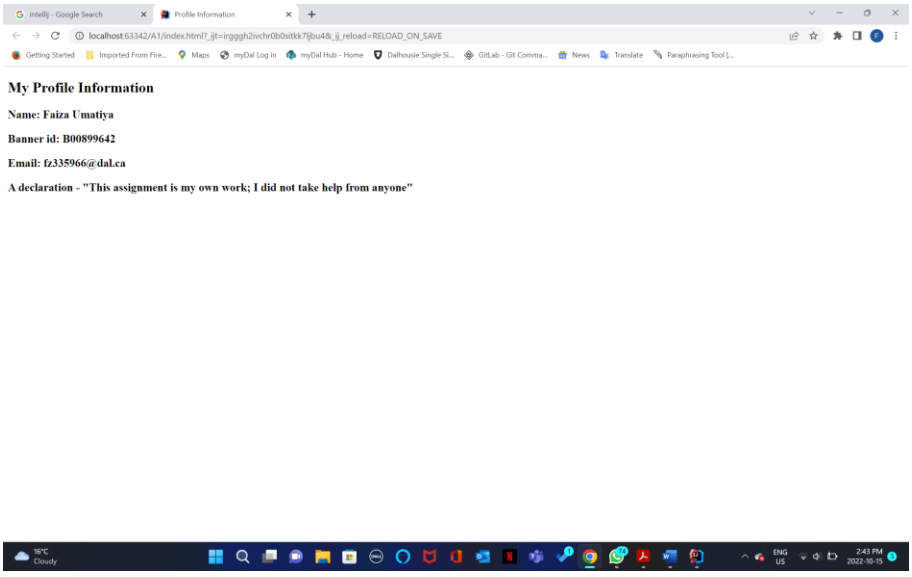


Fig 2: Contents of the "index.html" file

I created an "**index.html**" file in JAVA IDE (Refer Fig 1) that contains the profile information as the content of the page (Refer Fig 2).

The file contains:

- My full name
- My banner number
- My email
- A declaration – "This assignment is my own work; I did not take help from the anyone"

Step 2:

Setting up the AWS SDK for JAVA:

To make requests to AWS using the AWS SDK for JAVA [3], one should do the following:

1. Create an Account and login to the AWS Academy as shown in Fig 3 & 4.

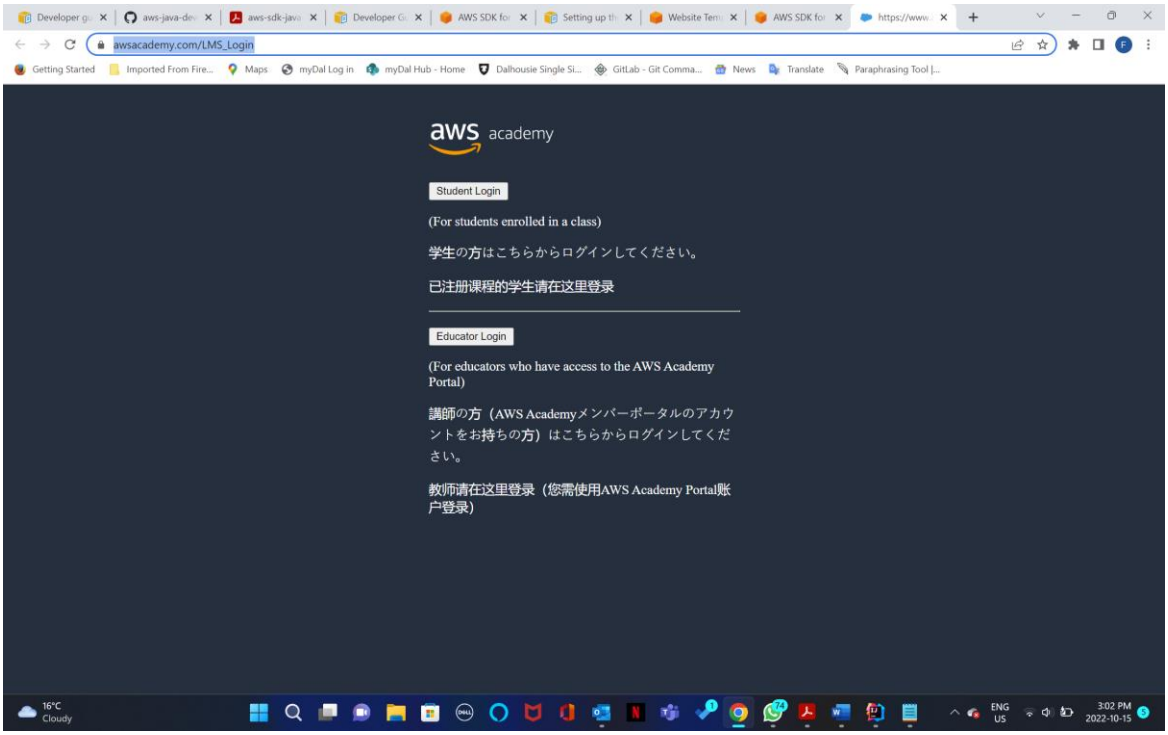


Fig 3: AWS Academy account login page

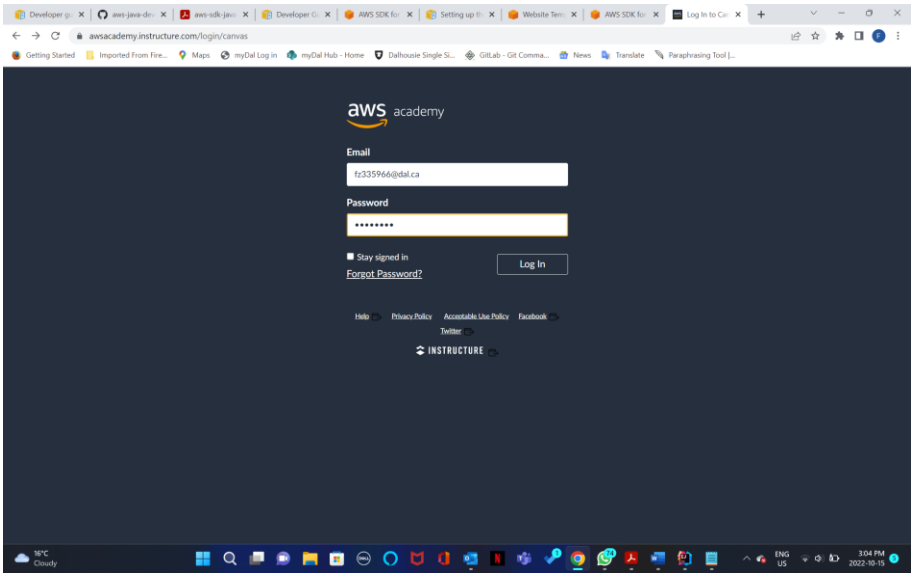


Fig 4: Student Account login page

2. Install JAVA and Build Tool [3]:
    - The development environment must have JAVA 8 or later [3]. Java works with the Oracle JAVA SE Development Kit and with Open Java Development Kit (Open JDK) such as Amazon Corretto[3].
    - A build tool or IDE supports Maven central such as Apache Maven, Gradle, or IntelliJ [3].

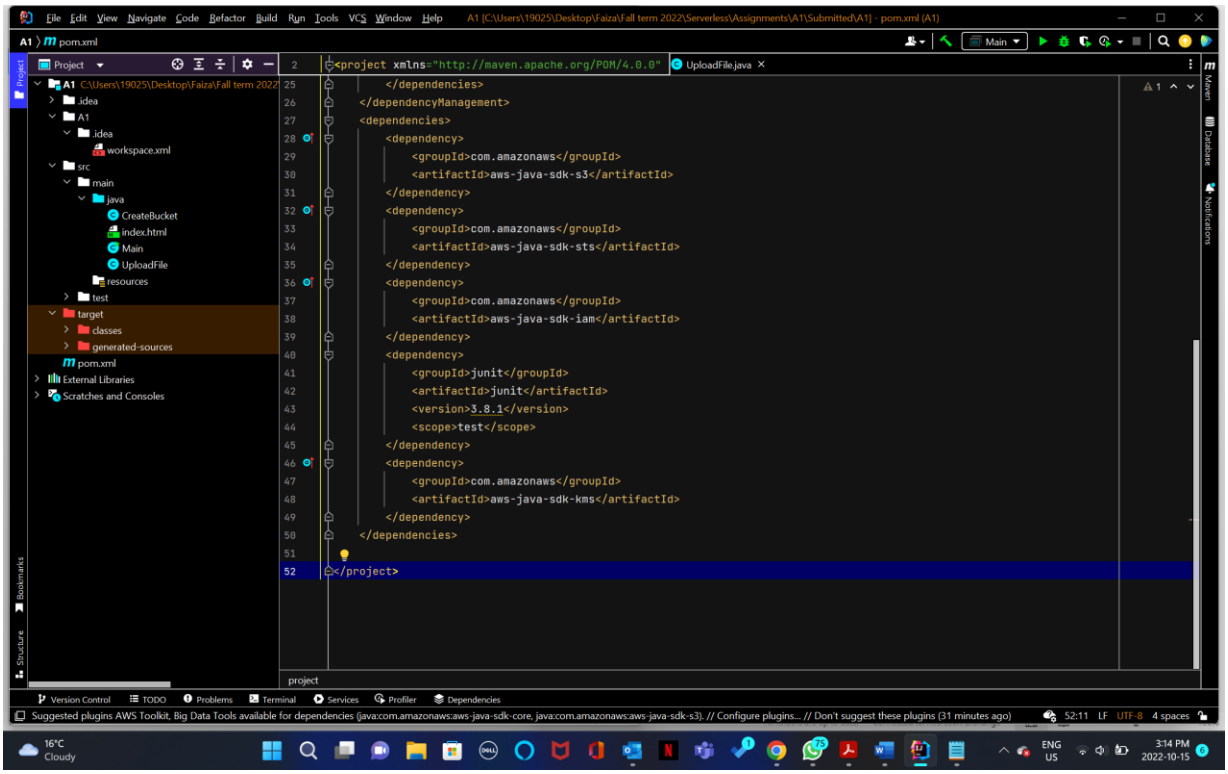


Fig 5: Dependencies in the "pom.xml" file

3. Once the environment and the tool is set, create a JAVA project and add dependencies for the services used in the application [3] as shown in the fig. 5.

4. <u>After login to the AWS academy, Go to AWS Academy Learner Lab and follow the below steps:</u>

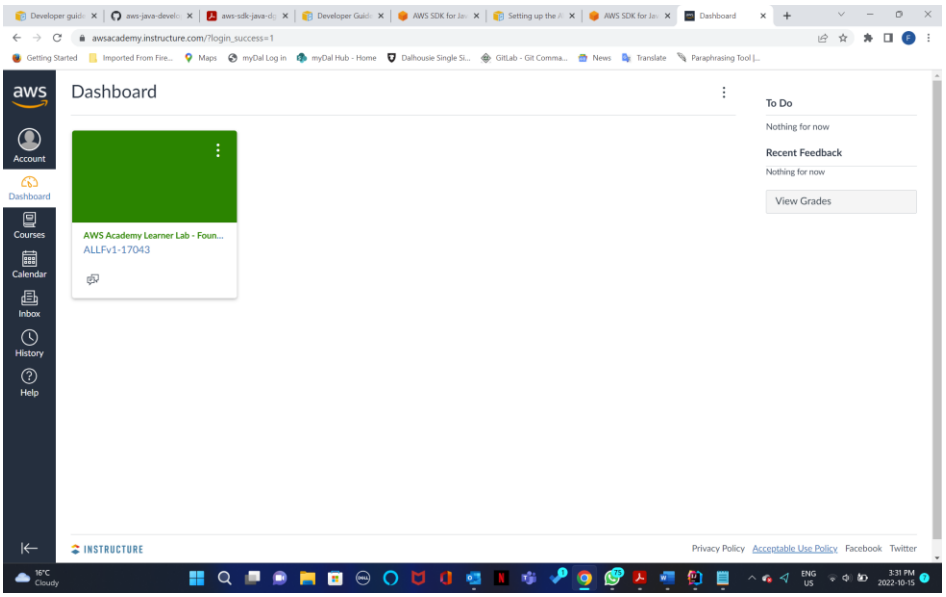


Fig 6: AWS Dashboard

- The first window you'll see open is the AWS Dashboard as shown in the fig. 6

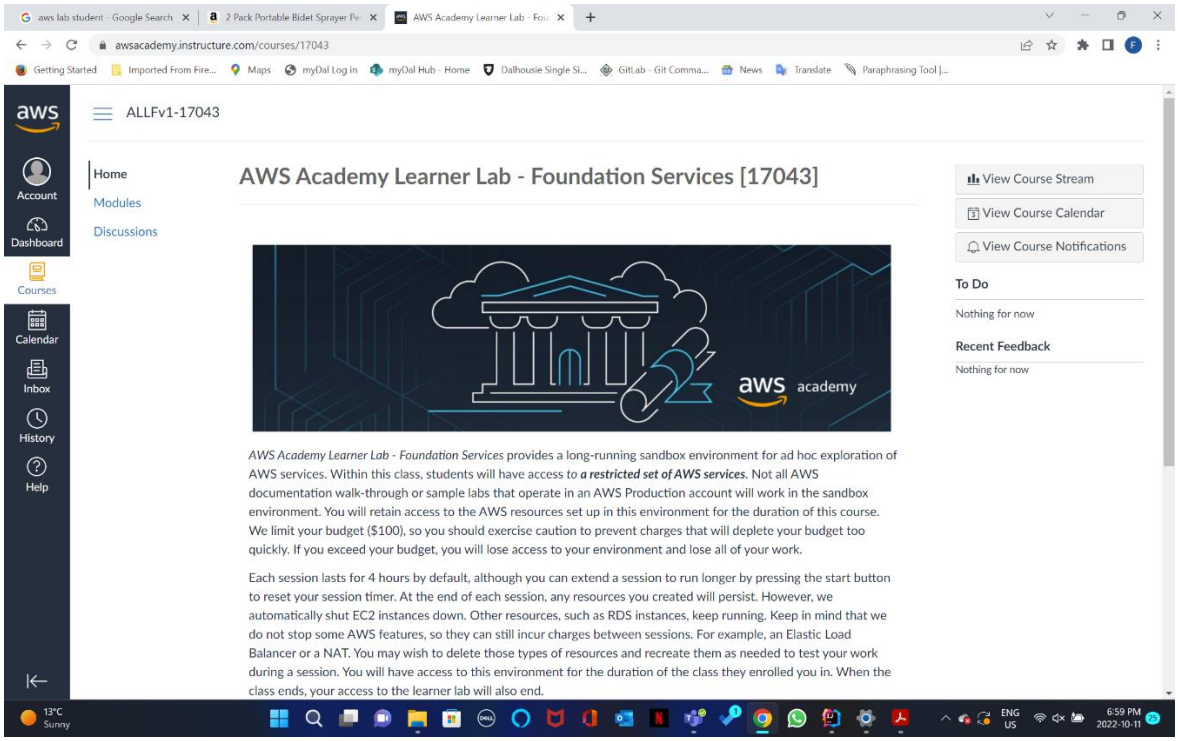


Fig 7: AWS home page

- Click on the AWS Academy Learner Lab shown in the fig. 6 and the new window will be open which is shown in the fig 7.
- Go to modules as shown in the fig 7. After clicking on the modules, AWS modules page will open as shown in the fig 8.

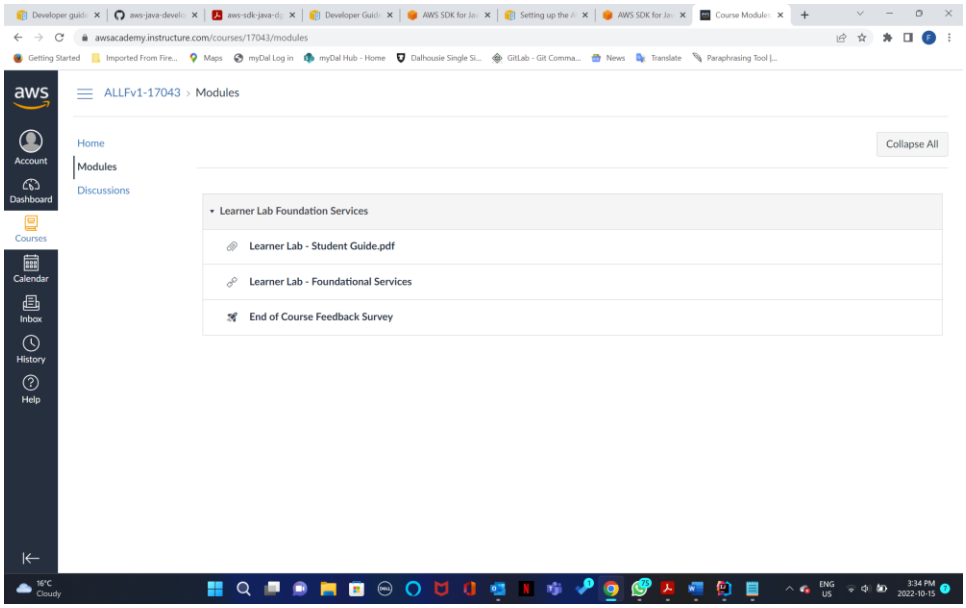


Fig 8: AWS Modules Page

- Go to “Learner Lab – Foundational Services” as shown in fig 8. After clicking on it, the start lab and end lab page will open as shown in the fig 9.

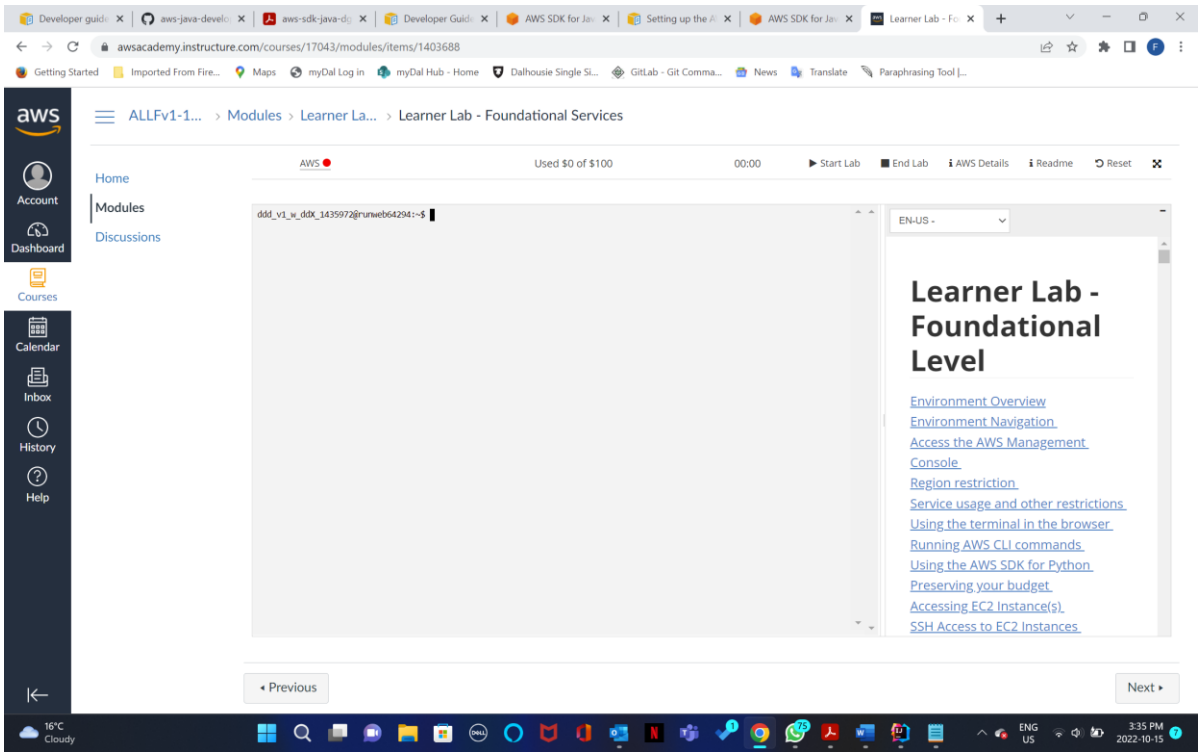


Fig 9: AWS End lab

- Click on the Start Lab to start the lab (The red dot represents the lab is not started yet) as shown in the fig 9.

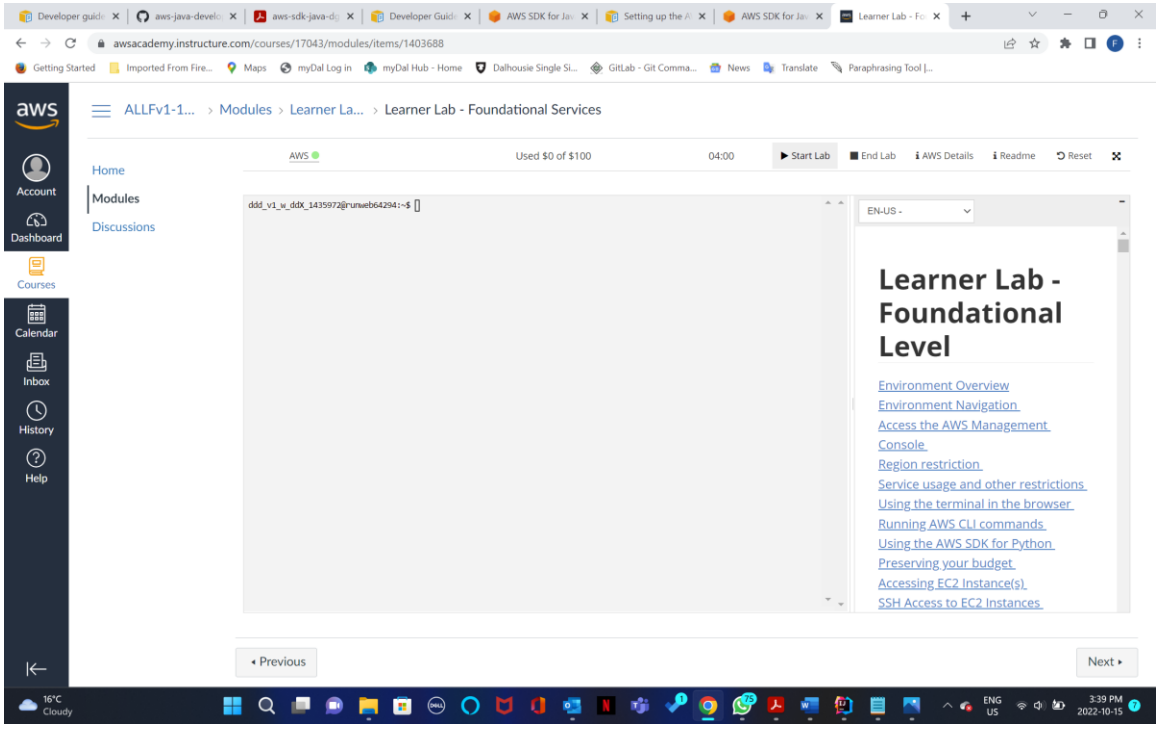


Fig 10: AWS start lab

- Lab has started because the green dot indicates the lab has started as shown in the fig 10.

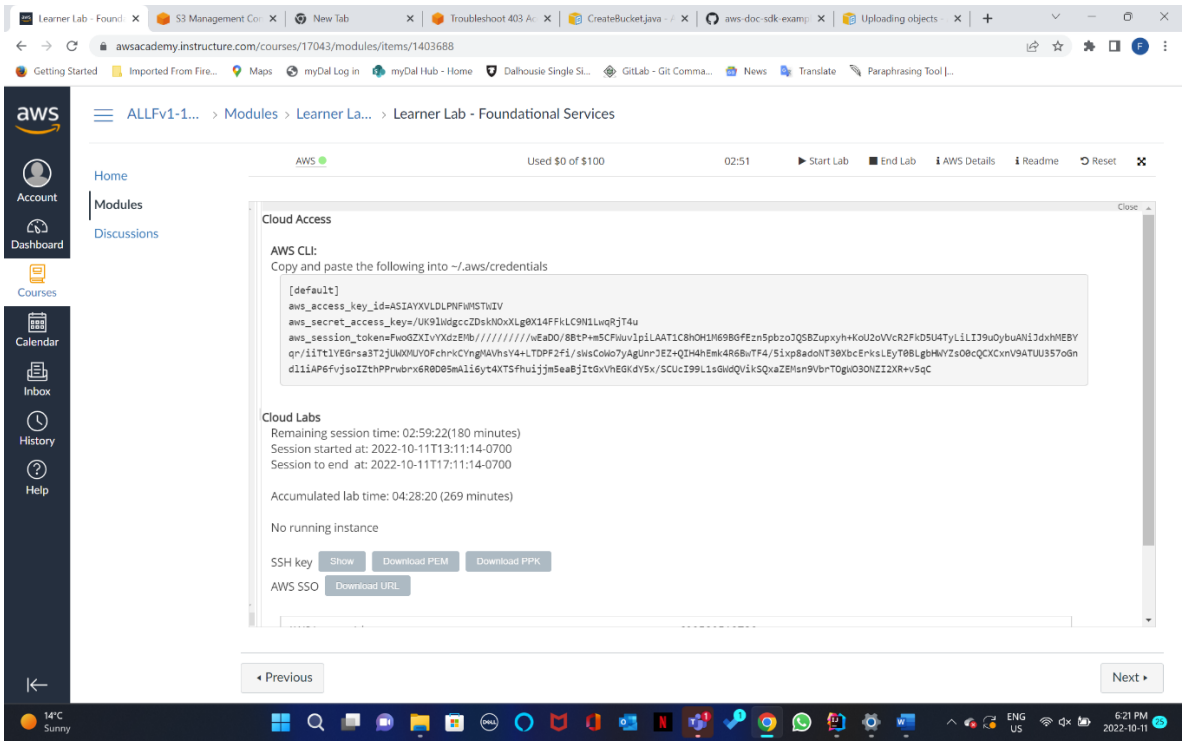


Fig 11: Cloud Access details

- Click on the “AWS details” (Refer fig 10) to get the AWS access key, secret key and session token. This is used to make the connection to the AWS S3 as shown in the fig 11.

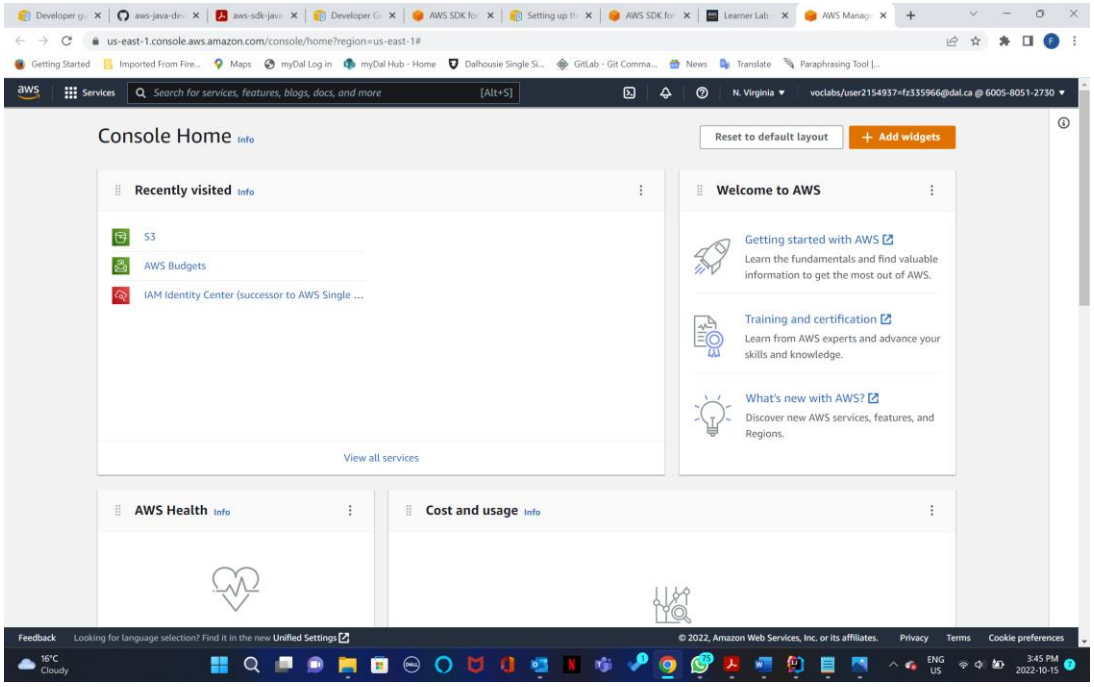


Fig 12: AWS Console home

- Click on AWS green icon (Refer fig 11) and it will redirect to the Console Home as shown in the fig 12.

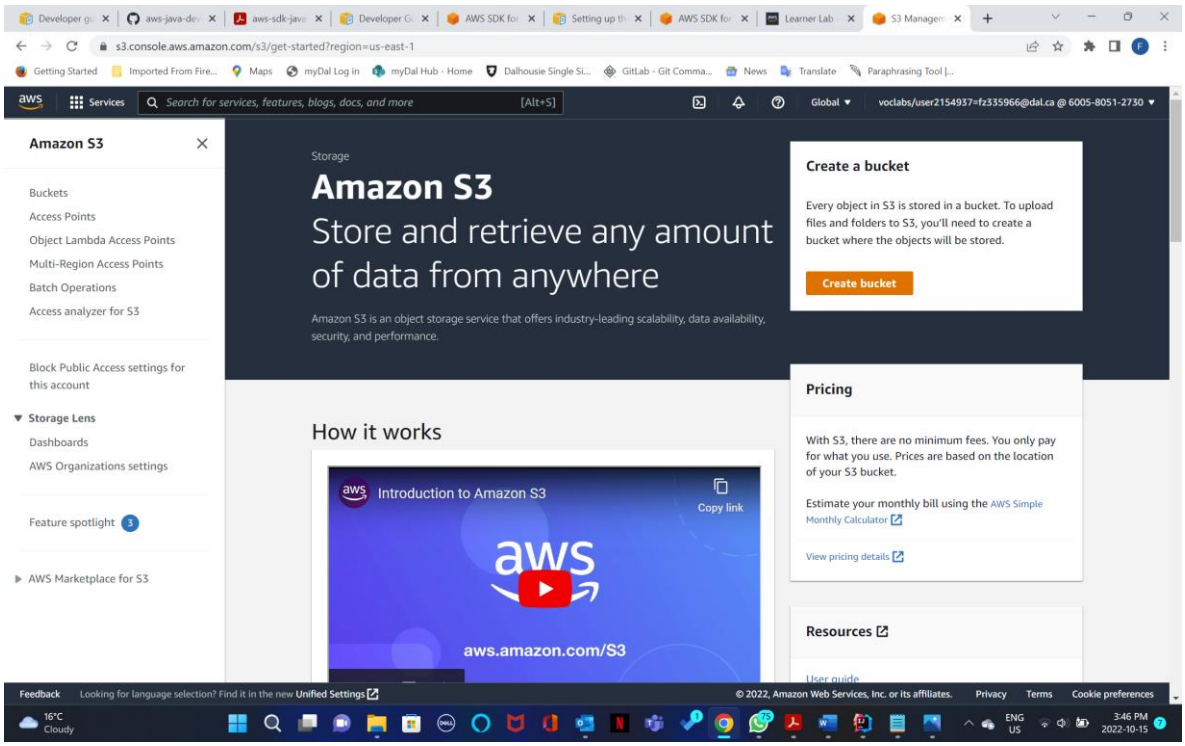


Fig 13: Amazon S3 page

- Select S3 which is shown in fig 12 and it will open Amazon S3 page.
- Go to Buckets which is shown in fig 13 to check if there are any buckets presents.

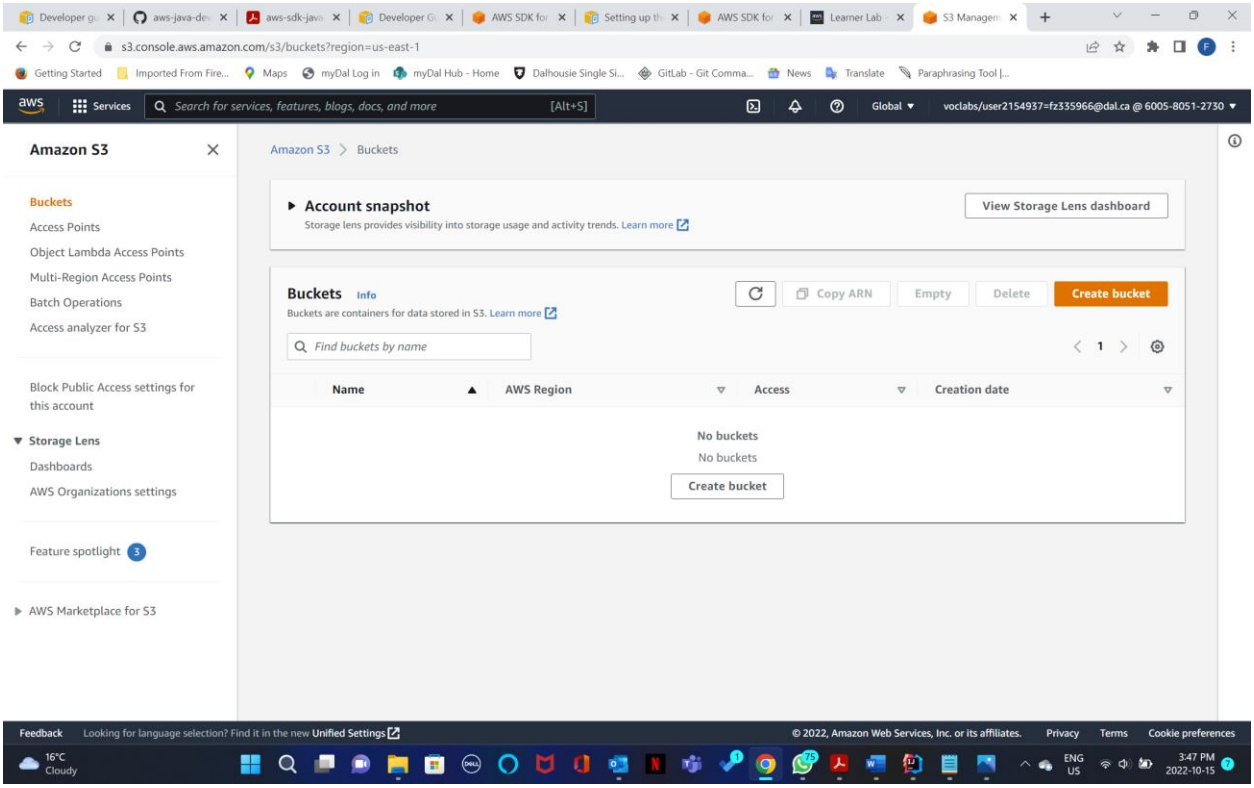


Fig 14. Amazon S3 Buckets

- The fig shows there are no buckets so create a bucket using the below steps
- Follow the below steps.

5. Write a JAVA program to create a S3 bucket using SDK specification.
   - I created a class name “CreateBucket.java” which consists of 3 functions i.e connectToAWSS3(), getBucket() and createAwsBucket().

```
4 usages
public class CreateBucket {
    3 usages
    public AmazonS3 connectToAWSS3() {
        return AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).withCredentials(new AWSStaticCredentialsProvider(new BasicSessionCredentials(
                awsAccessKey: "ASIAYXVLDLPNFWMSTWIV",
                awsSecretKey: "/UK9lWdgccZDskNOxXLg0X14FFkLC9N1LwqRjT4u",
                sessionToken: "FwoGZXIvYXdzEMb/////////wEaDO/8BtP+m5CFWuvlpiLAAT1C8hOH1M69B6fEzn5pbzoJQSBZupxyh+KoU2oVVcR2FkD5U4TyLiLIJ9uOybuANiJdxhMEBYqr/iiTtlYEGrsa3T2j
        ))).build();
    }
```

Fig 15: connectToAWSS3 function

- connectToAWSS3() – establishes the connection to the AWS S3 using the AWS access key, AWS secret key and session Token (Refer Fig 15).

```
1 usage
public Bucket getBucket(String bucket_name) {
    final AmazonS3 s3 = connectToAWSS3();
    Bucket named_bucket = null;
    List<Bucket> buckets = s3.listBuckets();
    for (Bucket b : buckets) {
        if (b.getName().equals(bucket_name)) {
            named_bucket = b;
        }
    }
    return named_bucket;
}
```

Fig 16: getBucket()

- getBucket() – gets the bucket from the list if its present (Refer Fig 16).

```
1 usage
public Bucket createAwsBucket(String bucket_name) {
    final AmazonS3 s3 = connectToAWSS3();
    if (s3.doesBucketExistV2(bucket_name)) {
        System.out.format("Bucket %s already exists.\n", bucket_name);
        return getBucket(bucket_name);
    } else {
        try {
            return s3.createBucket(bucket_name);
        } catch (AmazonS3Exception e) {
            System.err.println(e.getErrorMessage());
            return null;
        }
    }
}
}
```

Fig 17: createAwsBucket()

- createAwsBucket() – it will check whether the bucket with the name exists in the list of bucket. If yes, it will return that bucket else will create a new bucket (Refer Fig 17).

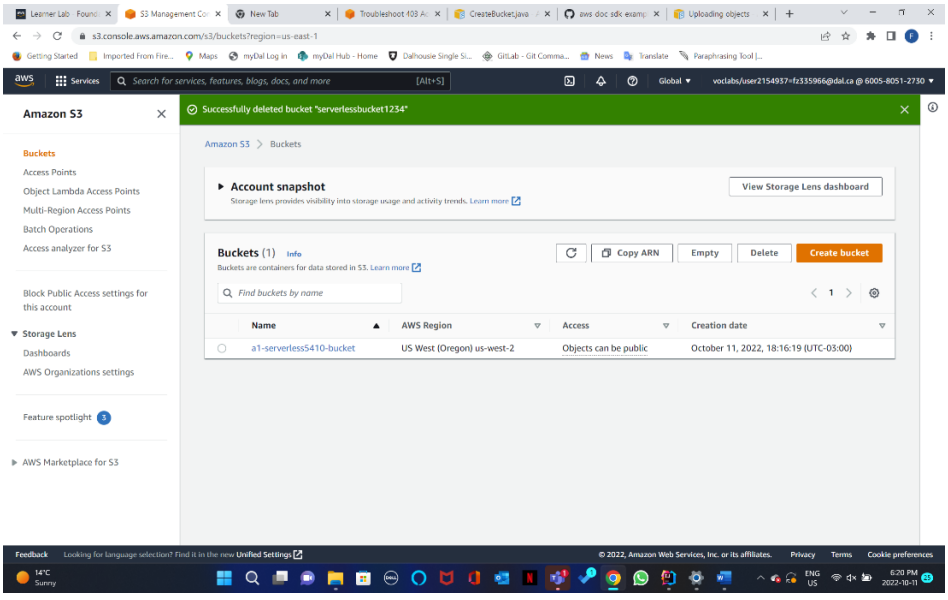


Fig 18: Bucket creation

6. After running the code, the bucket gets created in the Amazon S3 buckets. Refer fig 18 to see the buckets that has been created. In the fig 18, “a1-serverless5410-bucket” has been created.

Step 3:

I wrote a JAVA program to upload the “index.html” file from my computer to the S3 bucket that I created i.e. “a1-serverless5410-bucket”.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.AmazonS3;
import java.io.File;

1 usage
public class UploadFile {
    1 usage
    public static void uploadObject() {
        CreateBucket cb = new CreateBucket();
        final AmazonS3 s3 = cb.connectToAWSS3();

        try {
            System.out.println("file being uploaded");
            s3.putObject( s: "a1-serverless5410-bucket", s1: "index.html", new File( pathname: "C:\\Users\\19025\\Desktop\\Faiza\\Fall term 2022\\Serverless\\Assignments\\A1
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit( status: 1);
        }
    }
}
//https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-s3-objects.html
```

Fig 19: UploadFile class

- I created a class name “UploadFile” that contains a function called uploadObject() which uploads the file “index.html” in the S3 bucket by providing the file path refer fig 19.

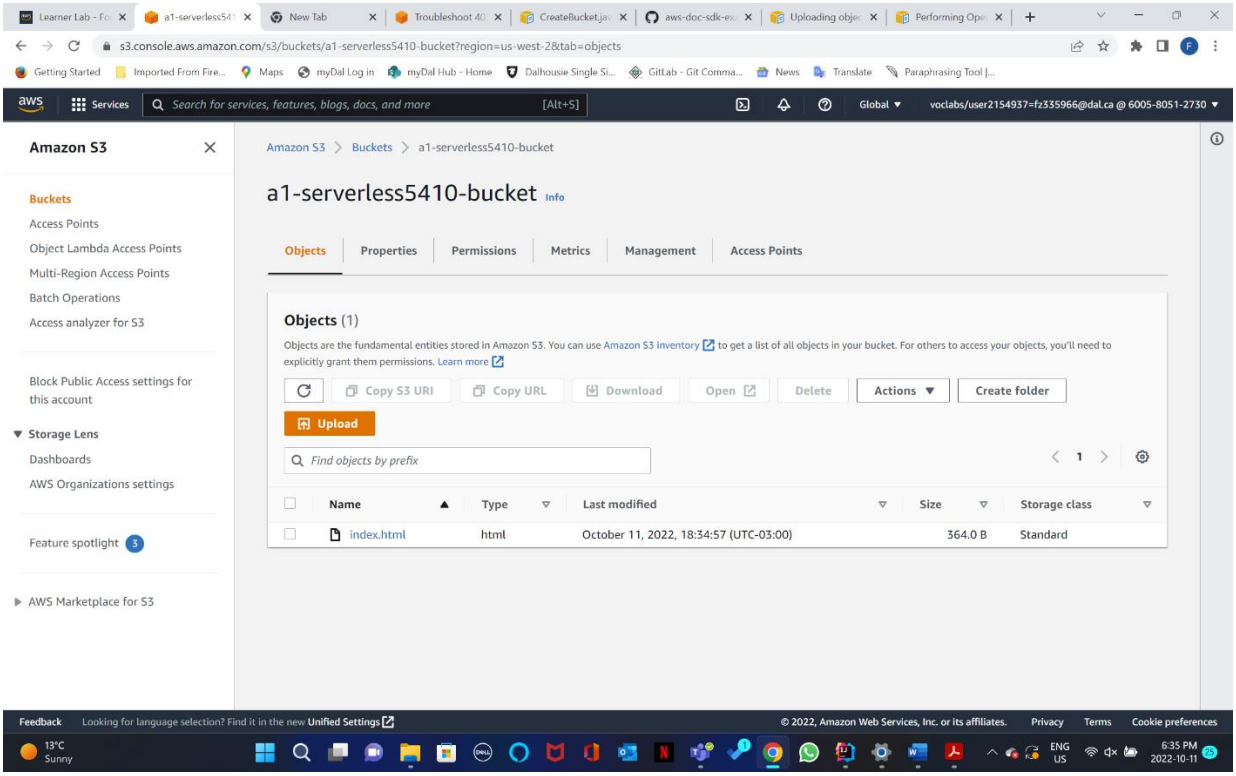


Fig 20: File uploaded in the bucket

- The “index.html” has been uploaded in the S3 bucket as shown in the fig 20.

Step 4:

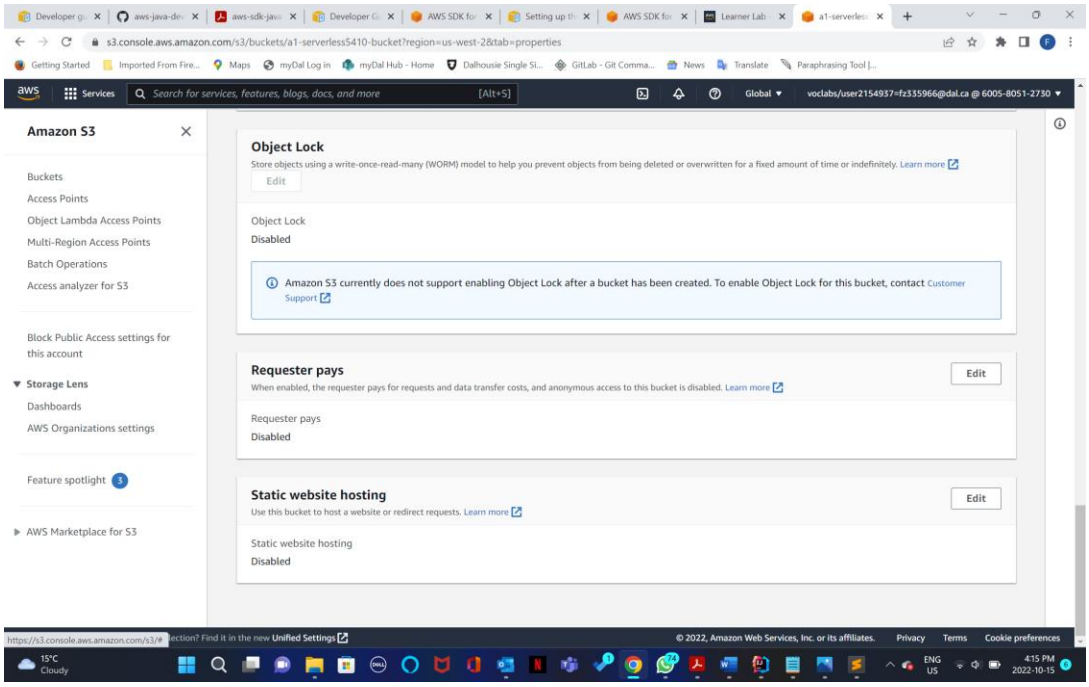


Fig 21: Static website hosting

- After uploading the “index.html” in S3 bucket, go to static website hosting option (Refer fig 21) and click on the option to edit the hosting settings as shown in the fig 22.

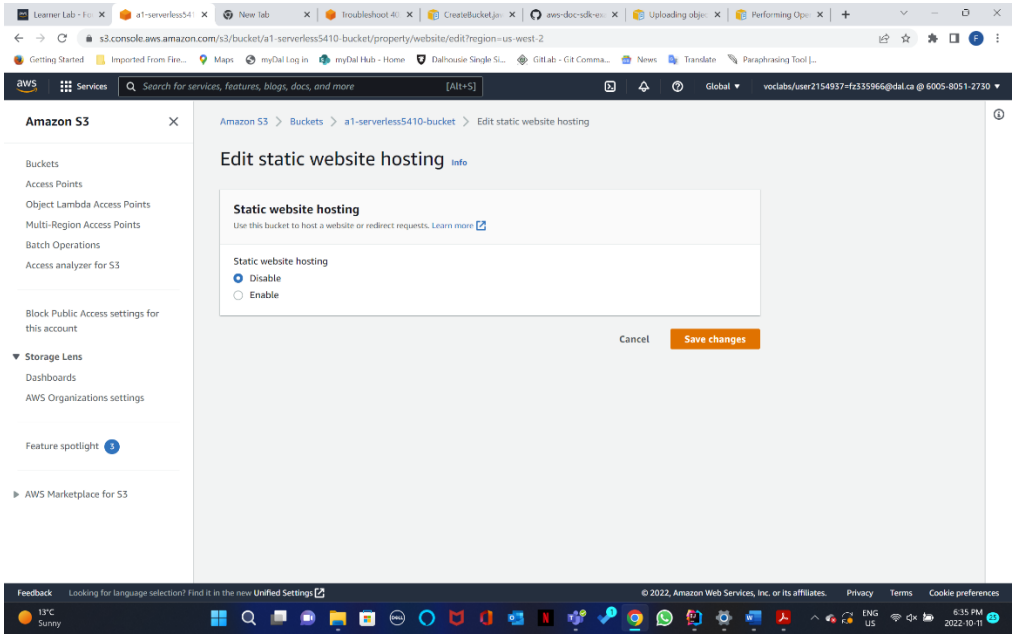


Fig 22: Edit static website hosting page

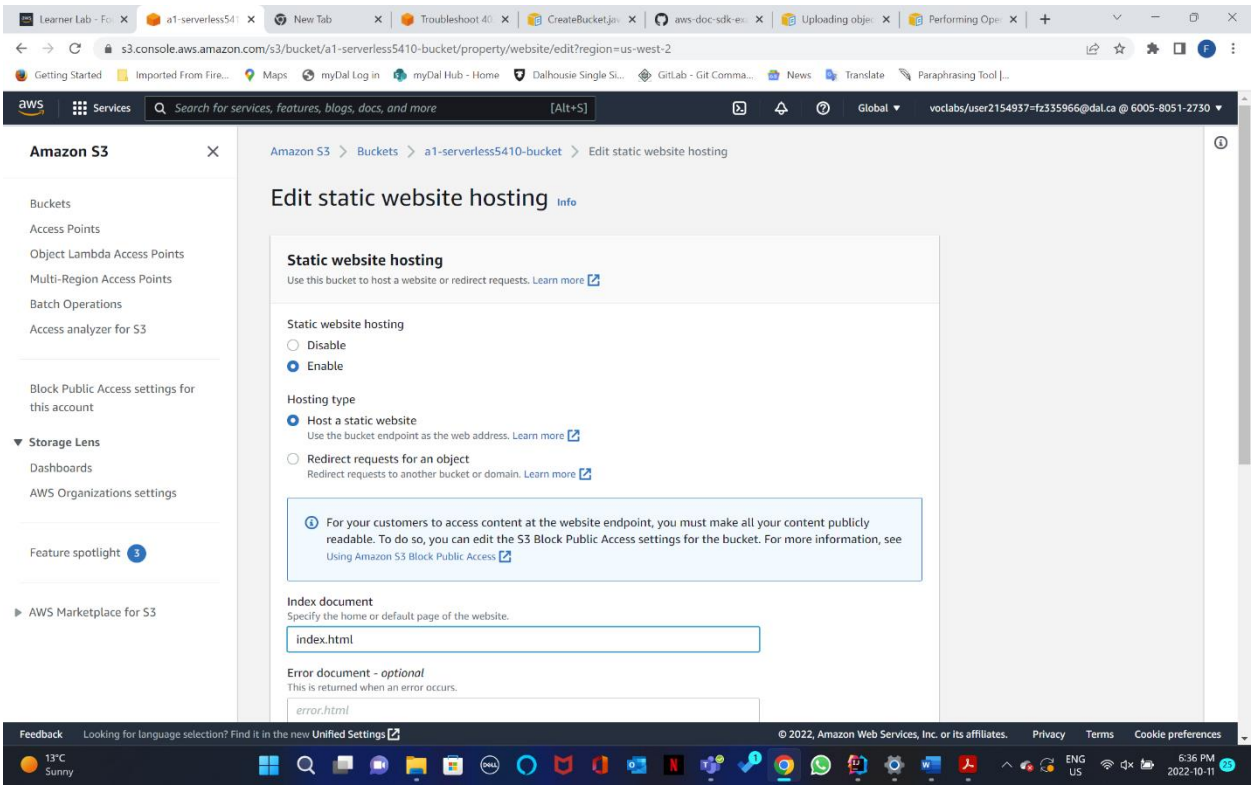


Fig 23: Edit static website hosting page

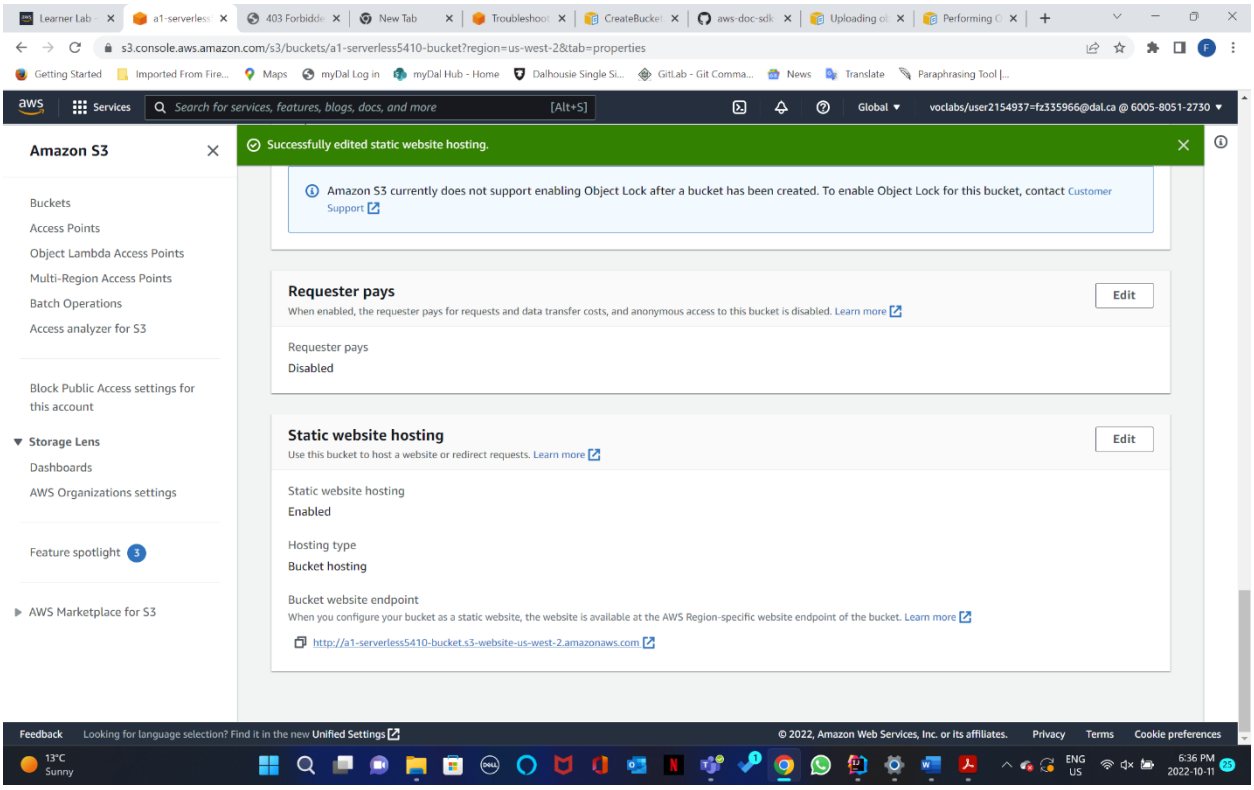


Fig 24: Saved changes for the static website hosting

- To enable the static hosting website, select the “enable” option and “host a static website option” as shown in the fig 23 and save the changes.
- The fig 24 shows the changes edited for the static website hosting has been successful.

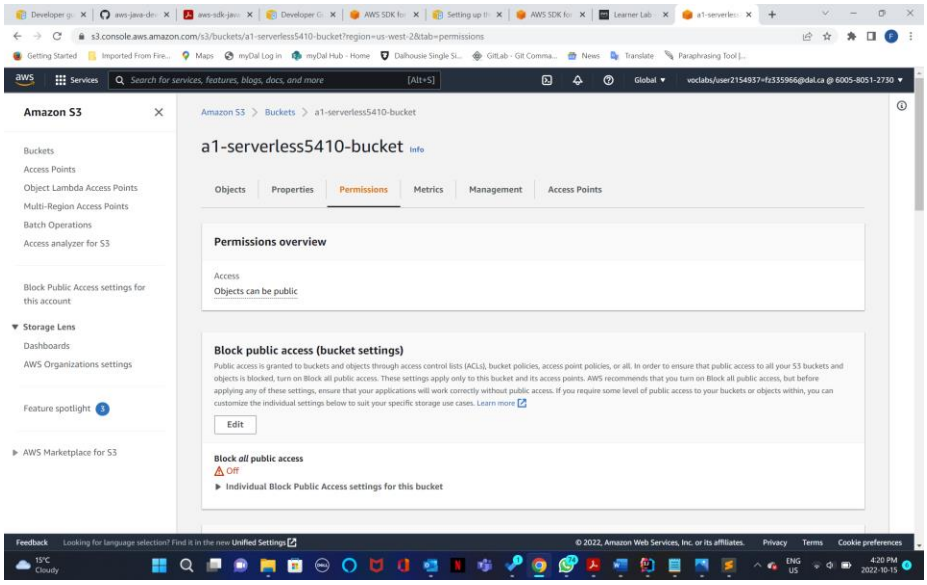


Fig 25: Saved changes for the static website hosting

- To make changes in the policy of S3 bucket for hosting the static webpage, go to “Permissions” section under the bucket “a1-serverless5410-bucket” as shown in the fig 25.

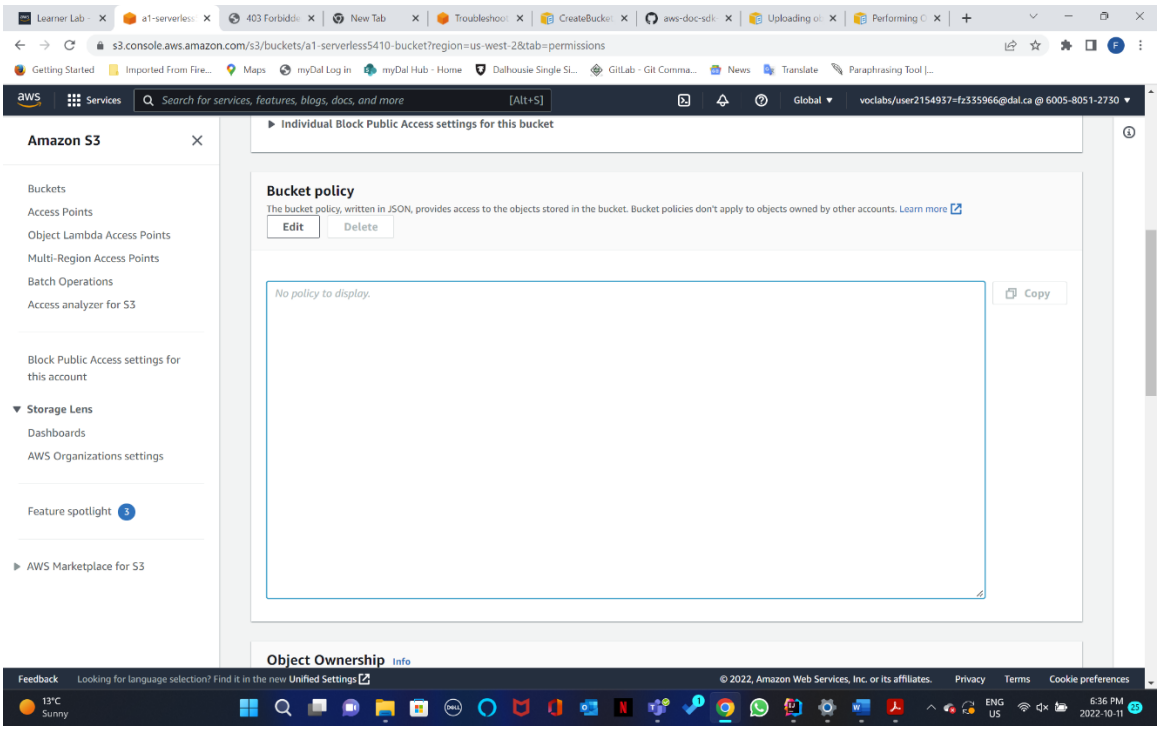


Fig 26: Edit Bucket policy

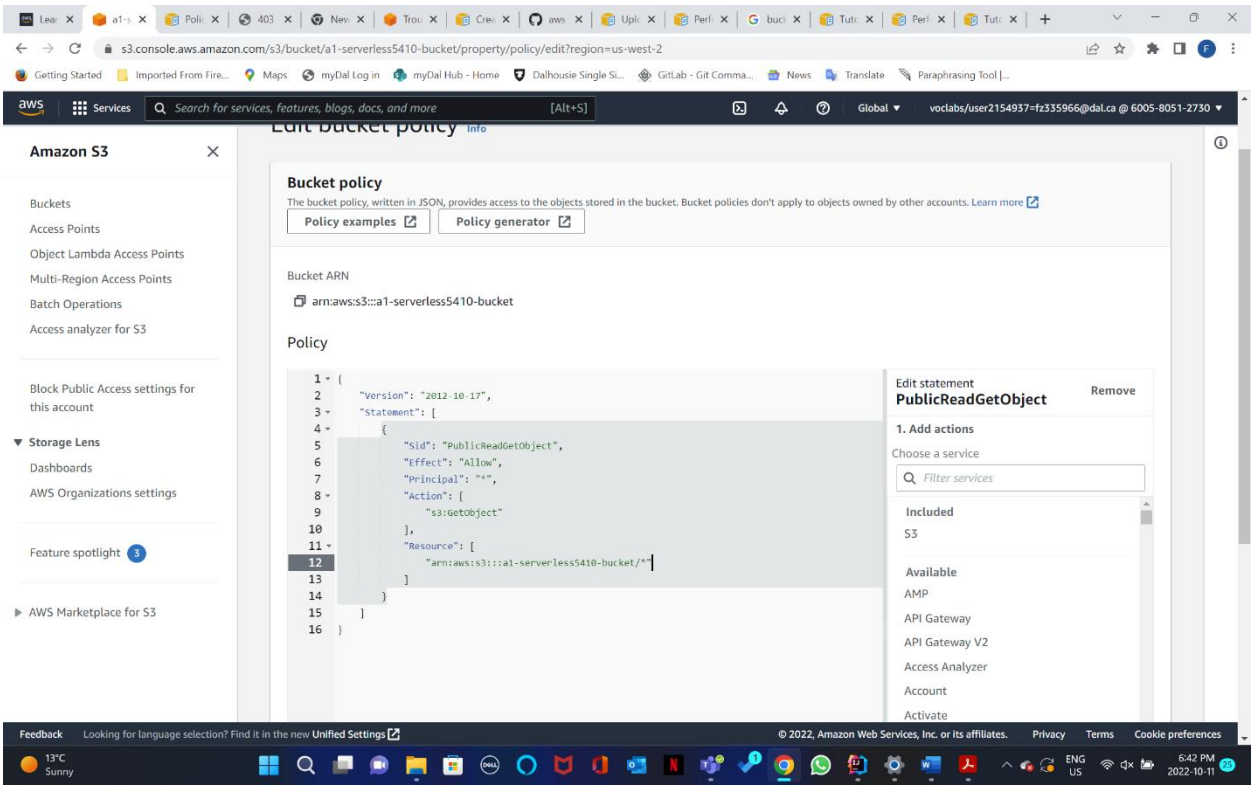


Fig 27: Updated Bucket policy

- Under “Permissions” go to “bucket policy” to make changes in the policy as shown in the fig 26. The updated policy is written in the JSON format as shown in the fig 27.

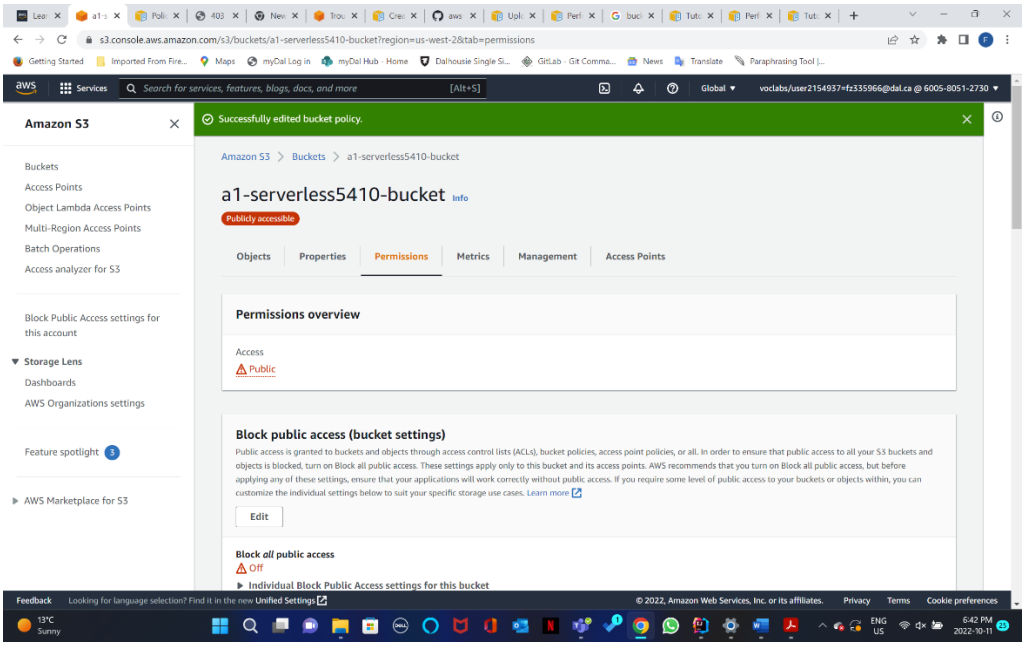


Fig 28: Bucket publicly accessible

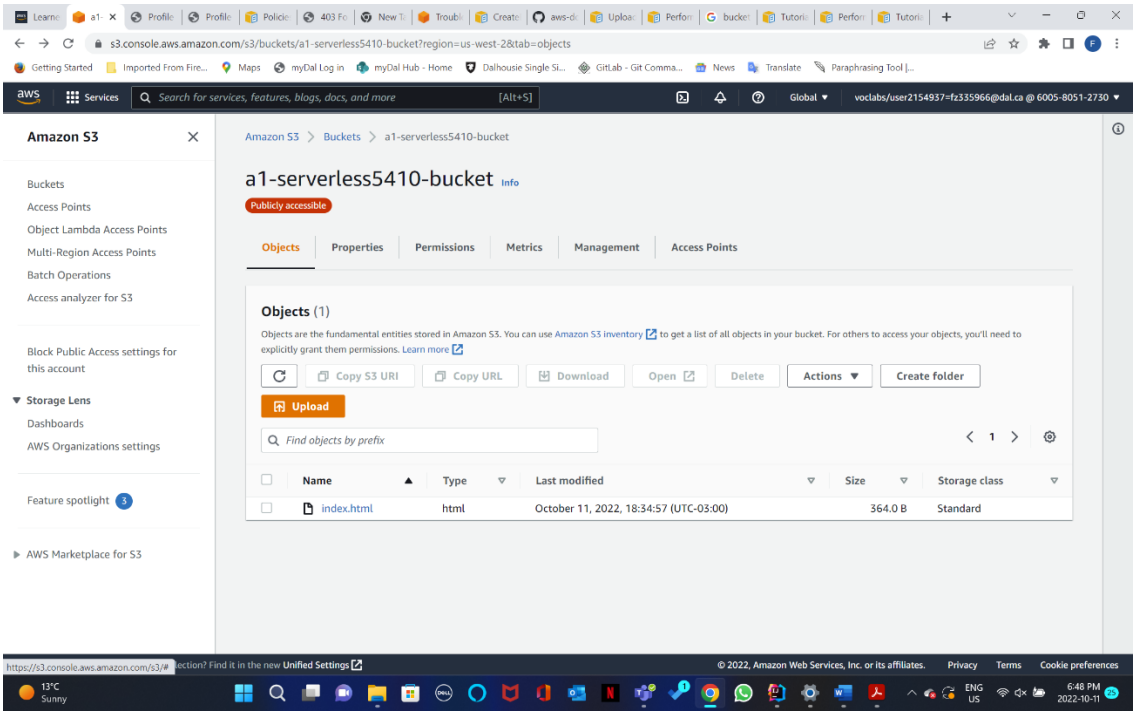


Fig 29: Bucket publicly accessible

- Now the static website hosted is publicly accessible (Refer fig 28 & 29) by changing the bucket policy (Refer fig 27).

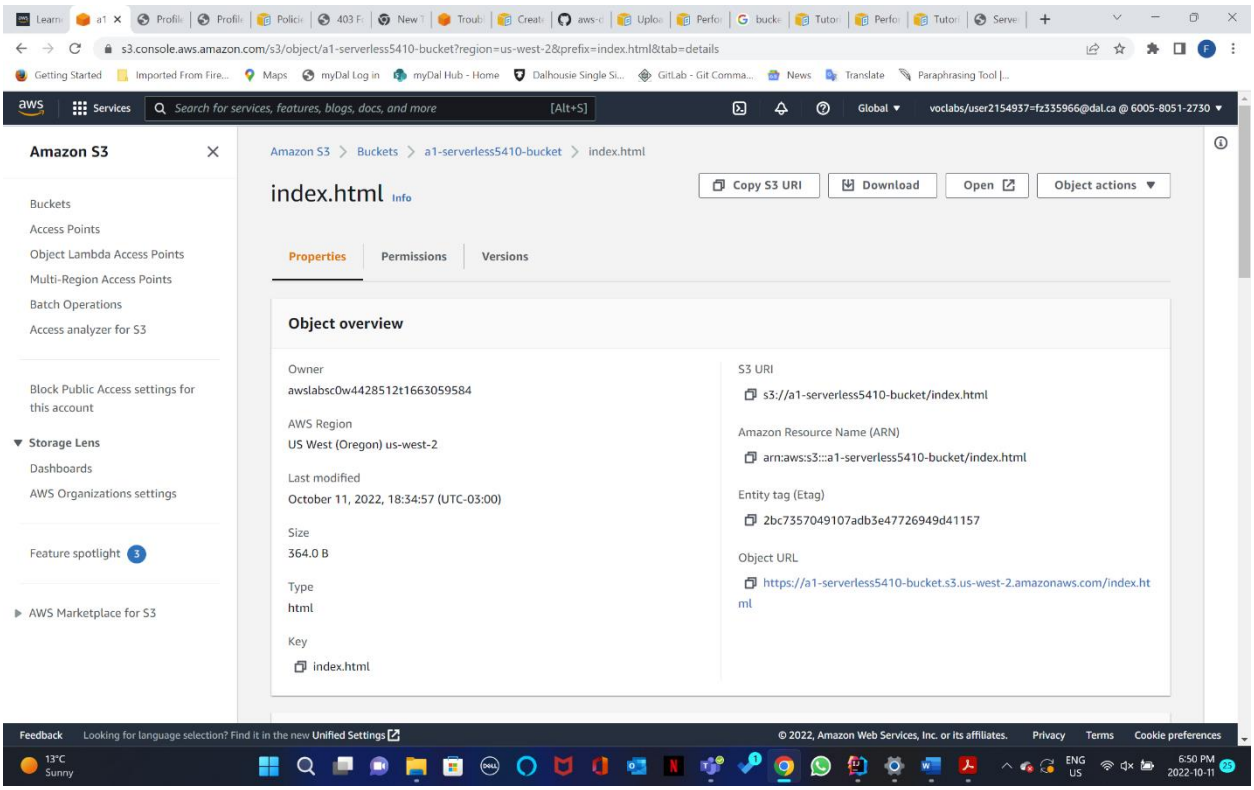


Fig 30: index.html info page

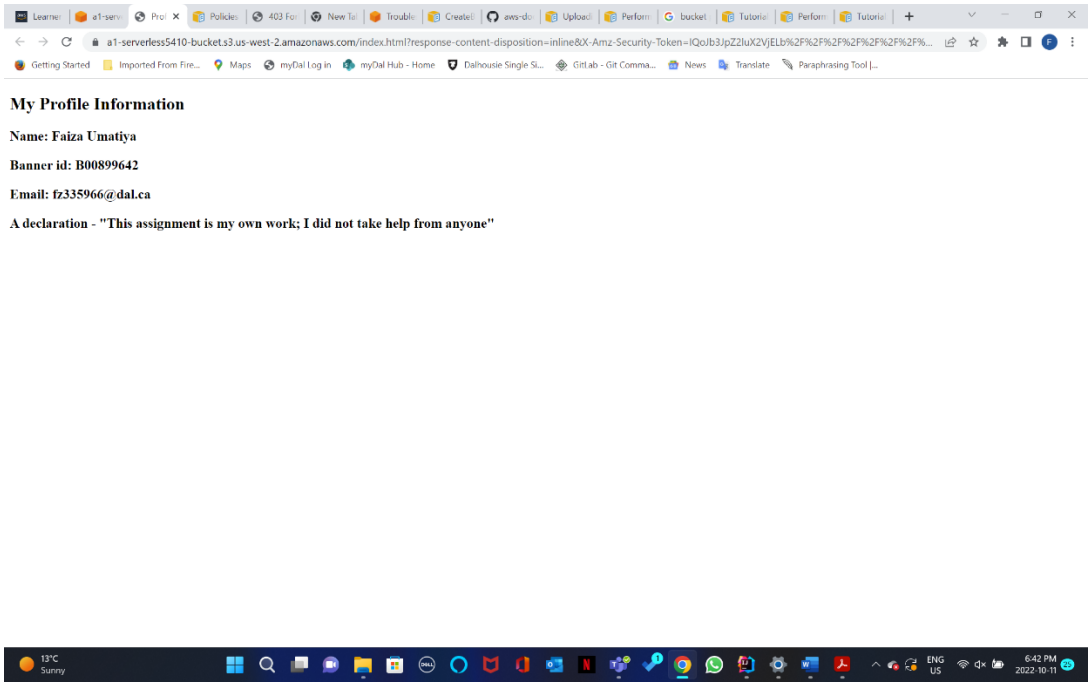


Fig 31: Static website hosted

- Go to the Object URL (Refer fig 30) and browse it to see the website that has been hosted (Refer fig 31).

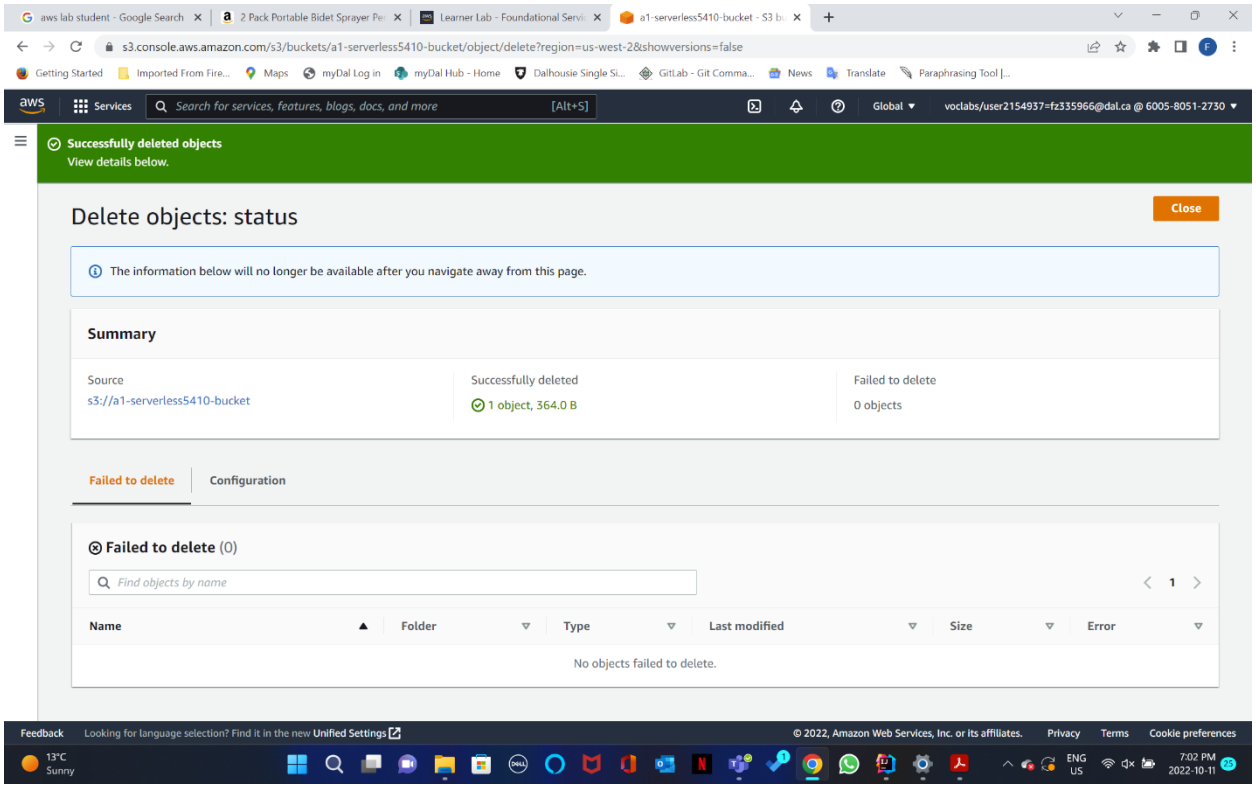


Fig 32: Object (index.html) deletion

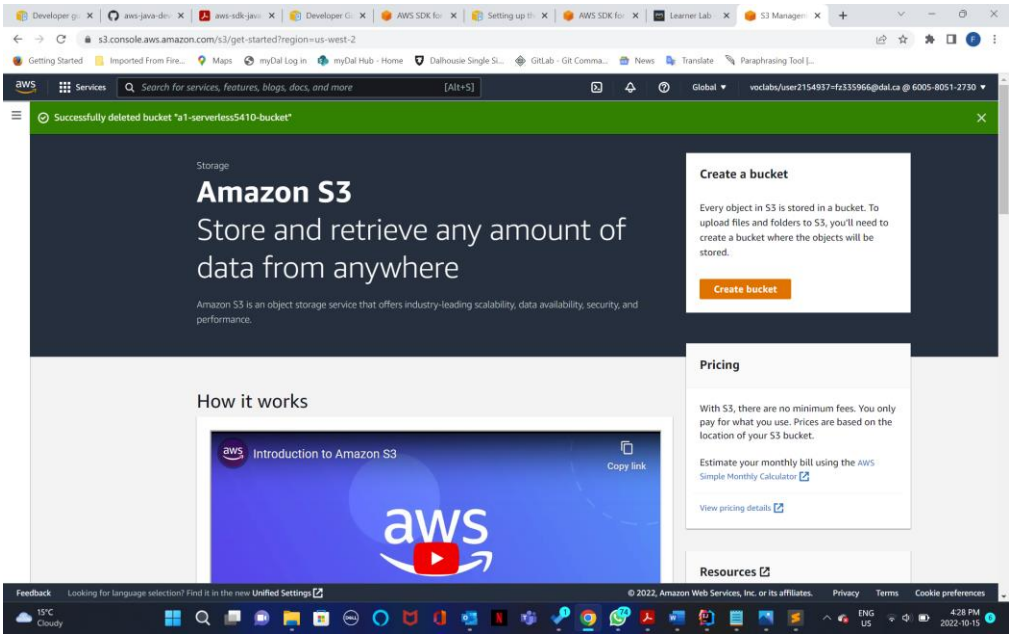


Fig 33: Bucket deletion

- The last step is to stop the AWS instance. To do that delete the object (Refer fig 32) bucket (Refer fig 33).

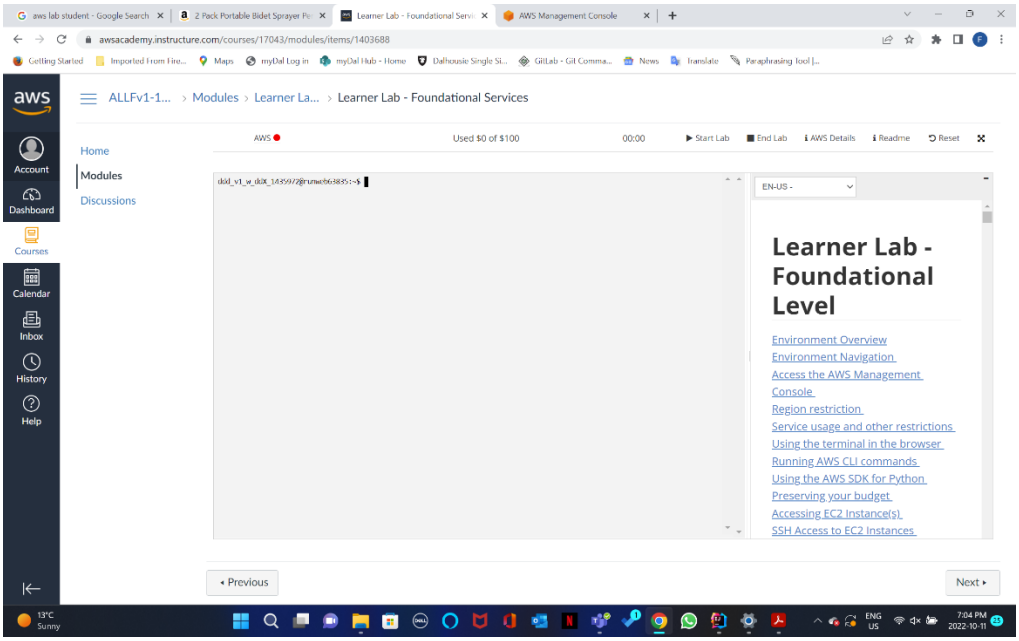


Fig 34: AWS End Lab

- Stop the AWS instance by clicking on the “End Lab”. If the dot turns red means the AWS lab has ended as shown in the fig 34.

Program Script:

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>A1</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>18</maven.compiler.source>
        <maven.compiler.target>18</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-bom</artifactId>
                <version>1.11.837</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-s3</artifactId>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-sts</artifactId>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-iam</artifactId>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-kms</artifactId>
        </dependency>
    </dependencies>

</project>
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Profile Information</title>
</head>
<body>
<h2>My Profile Information</h2>
<h3>Name: Faiza Umatiya</h3>
<h3>Banner id: B00899642</h3>
<h3>Email: fz335966@dal.ca</h3>
<h3> A declaration - "This assignment is my own work; I did not take help
from anyone"</h3>
</body>
</html>
```

CreateBucket.java:

```
import com.amazonaws.auth.*;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.Bucket;
import java.util.List;

/*
Code Reference:
[1] https://docs.aws.amazon.com/code-samples/latest/catalog/java-s3-src-main-
java-aws-example-s3-CreateBucket.java.html
For dependencies:
[2] https://github.com/awsdocs/aws-doc-sdk-
examples/blob/main/java/example_code/s3/pom.xml
*/

public class CreateBucket {
    public AmazonS3 connectToAWSS3() {
        return
AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).withCrede
ntials(new AWSStaticCredentialsProvider(new BasicSessionCredentials(
                "ASIAYXVLDLPNAQ5L4AT7",
                "C5B5seDkMS44MXjmzjrwRThHx2U7OTZujAGAZzI9",

"FwoGZXIvYXdzECQaDMKhBHOblio2tTpO/SLAAYrTLTiuS20CgZKSyBNOU4ll8N2AcZBefwRj2pAZ
2d0mV2A99QBAlZmTHrobQpdAmc588CG5LUrGTEgfeLKfDLVlJg8mKSFZnMsGVjUK/nRs4UoeVe2Pp
ZBhtMqsa87MmQAf6LFX7QbSSuqtja2QJXOX2HfxSyktTe5o4DO3ExsgbKJhd1yweAnD4qMeDGttH1
m6dmEFUZyWW2JnLS2LLRoLd9HULxXuqpUQ+nbi+FxpWrCjOaylLQUXEMSH8cldFSj6/quaBjItis9
SdQ5oJc4O6wFJLAPGOm1Zu/bruc3yP3pGwUgOX6WWKGgBvWex0lSFM53J"
        ))).build();
    }

    public Bucket getBucket(String bucket_name) {
        final AmazonS3 s3 = connectToAWSS3();
        Bucket named_bucket = null;
```

```
        List<Bucket> buckets = s3.listBuckets();
        for (Bucket b : buckets) {
            if (b.getName().equals(bucket_name)) {
                named_bucket = b;
            }
        }
        return named_bucket;
    }

    public Bucket createAwsBucket(String bucket_name) {
        final AmazonS3 s3 = connectToAWSS3();
        if (s3.doesBucketExistV2(bucket_name)) {
            System.out.format("Bucket %s already exists.\n", bucket_name);
            return getBucket(bucket_name);
        } else {
            try {
                return s3.createBucket(bucket_name);
            } catch (AmazonS3Exception e) {
                System.err.println(e.getErrorMessage());
                return null;
            }
        }
    }
}
```

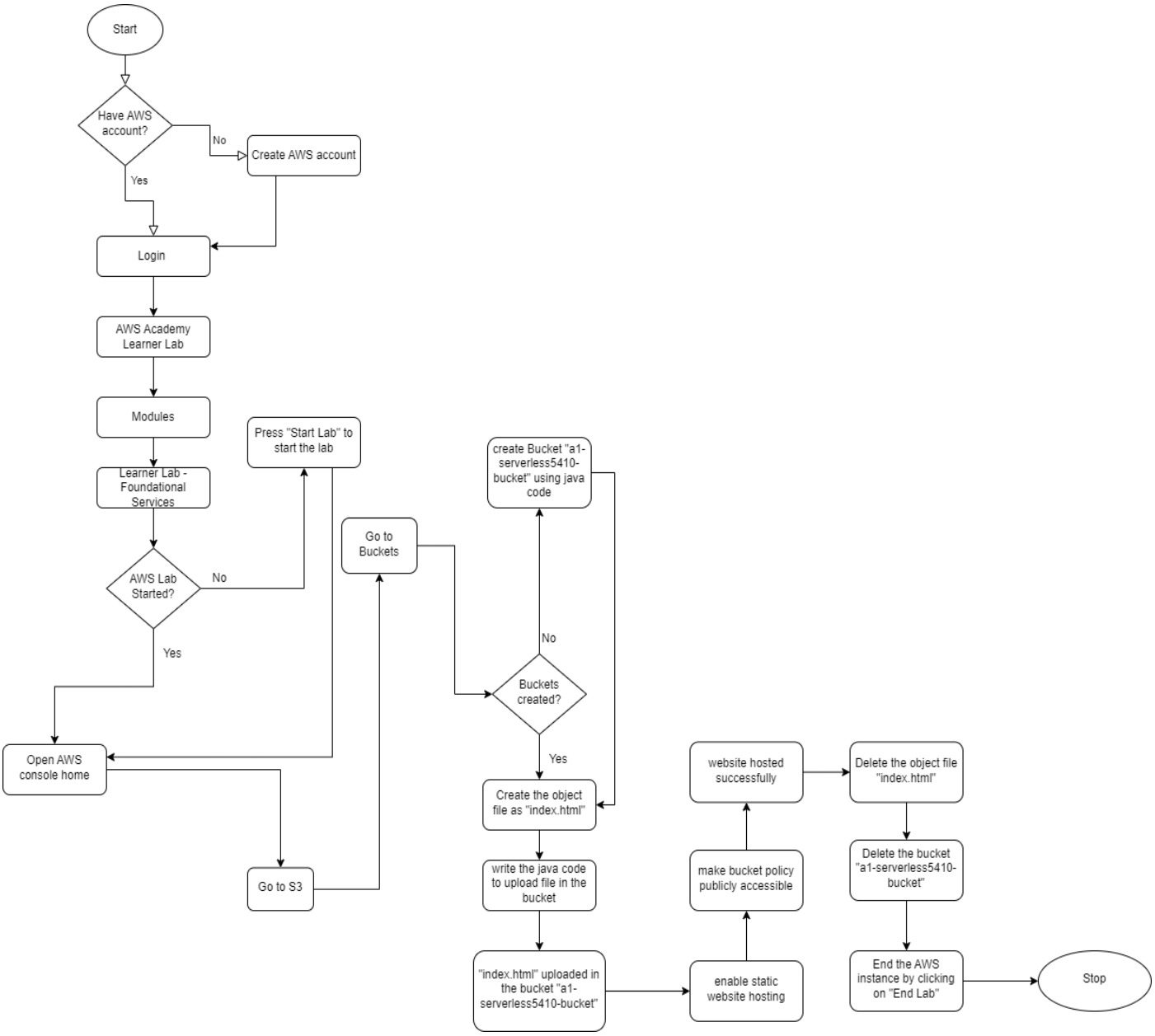
UploadFile.java:

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.AmazonS3;
import java.io.File;

/*
Reference:
[1] https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-s3-
objects.html
*/

public class UploadFile {
    public static void uploadObject() {
        CreateBucket cb = new CreateBucket();
        final AmazonS3 s3 = cb.connectToAWSS3();

        try {
            System.out.println("file being uploaded");
            s3.putObject("a1-serverless5410-bucket", "index.html", new
File("C:\\Users\\19025\\Desktop\\Faiza\\Fall term
2022\\Serverless\\Assignments\\A1\\Submitted\\A1\\src\\main\\java\\index.html
"));
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }
    }
}
```

Main.java:

```
public class Main {
public static void main(String[] args) {
        String bucket_name = "a1-serverless5410-bucket";
        System.out.format("\nCreating S3 bucket: %s\n", bucket_name);
        CreateBucket cb = new CreateBucket();
        cb.createAwsBucket(bucket_name);
        UploadFile.uploadObject();
        }
}
```

Flowchart of Operations:



Observation of the AWS JAVA SDK:

The AWS SDK for Java provides a Java API for AWS services. Using the SDK, one can easily build Java applications that work with Amazon S3, Amazon EC2, DynamoDB, and more [1]. The AWS SDK for Java simplifies use of AWS Services by providing a set of libraries that are consistent and familiar for Java developers [2]. It provides support for API lifecycle consideration such as credential management, retries, data marshaling, and serialization [2]. The AWS SDK for Java also supports higher level abstractions for simplified development [2].

References:

[1] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/welcome.html. [Accessed: 15-Oct-2022].

[2] *Amazon.com*. [Online]. Available: https://aws.amazon.com/sdk-for-java/. [Accessed: 15-Oct-2022].

[3] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/setup.html. [Accessed: 15-Oct-2022].