

**CSCI 5410**

**Serverless Data Processing**

**Assignment 1**

**Name : Sagarkumar Pankajbhai Vaghasia**

**CSID : vaghasia**

**Banner ID : #B00878629**

**Gitlab Link :**

<https://git.cs.dal.ca/vaghasia/csci5410-f23-b00878629/-/tree/A1>

**PART-A Reading Task**

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

As the technology is evolving the need of infrastructure, storage, resources are increasing for organizations. These needs are fulfilled by serverless computing with the introduction of Function-as-a-Service (FaaS) but alongwith it also has some challenges related to performance. One of them is cold start delay which occurs due to preparation time require for a function to execute. There are some fixed mechanisms used by popular platforms to reduce cold start delay. However, they are not appropriate for dynamic environment. The authors have proposed an optimal approach which determines the best policy for keeping the containers warm according to the invocations of the functions over the time.

The major issue that encountered is preparation time required by the container which is also known as cold start delay. Alongwith this, another point to consider is resource consumption. There are many ways to reduce cold start delay such as reducing container preparation delay by pre-warmed stem cell containers, reducing cold start occurrence by reusing containers. This paper provides an approach which not only reduces the cold start delay but also reduces the resource consumption.

The Openwhisk platform method keeps the containers warm for a fixed period which is called as idle-container window. As the occurrence of the function invocation varies over time, taking a constant occurrence value is not feasible. A two-layer approach is proposed by the authors to make the platform adaptable. In first layer, using the reinforcement learning, the best time for the idle-container window is determined which can reduce the number of the cold start delays. As we know that the longer the window is available, the cold start delays will be shorter, but more resource consumption will take place. So, authors find a way to predict the time intervals between invocations in the past which determines the value of the idle-container window in the future. The first layer steps are done by completing 4 stages: 1) Receiving the function execution log which includes the time interval. 2) Predicting the value of the idle-container window by using an equation of Gaussian probability distribution in terms of mean and standard deviation. 3) Acting by updating the new value of the idle-container window. 4) At the end, Reward function is calculated to balance the number of delays and memory wastes. In second layer, LSTM (long short-term memory) approach is used which predicts the maximum number of concurrent invocations to arrange the pre-warmed containers.

Authors have performed an experiment to compare the efficiency of the proposed two-layer approach with the current approaches (Openwhisk). They took two types of datasets where one includes sequential invocations and the other includes concurrent invocations. They also took computing platform as Openwhisk and setup a training environment. In this experiment, they created 10,000 entries with different average entry rates of invocations per hour such as 5, 10 and 20. Based on results, they came to know that the proposed two-layer approach is more able to predict invocations and execute them in the pre-warmed containers Therefore, it is clear from the outcomes and the figures that the proposed approach is 22.65% more efficient than other approaches.

**References :**

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.

URL: https://ieeexplore-ieee-org.ezproxy.library.dal.ca/document/9749611

**PART-B AWS S3 Experiment**

A flowchart defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task. figure 1 is a flowchart which shows the steps for creating the bucket in Amazon S3 console and then uploading index.html file in the bucket.

The steps to create a bucket and host the html file is easier to understand by the following description :

Firstly, I logged in to the Amazon AWS Academy by student login. In the home screen, there is a modules tab which redirects to my modules section. In that, I selected Learner Lab – Foundational Services which took me to the AWS console where I clicked on the Start Lab button the start the lab. Once the lab is started the Red blink near to the AWS will turn into green. After starting the lab, I clicked on AWS details button to get aws\_access\_key,0020aws\_secret\_access\_key, aws\_session\_token to setup the connection of JAVA code with AWS. After setup, I clicked on the AWS button with a green blink which took me Console Home. This console home has different options such as IAM, EC2, S3, RDS, Lambda and many more. I simply clicked on S3 as we will be working on S3 for this assignment. This took me to the S3 console home. Here, we can manually create the S3 bucket and upload and host the files. But as we are instructed to do it so by JAVA program, so we are not going to create the bucket manually.

Then, I explored AWS SDKs[1] for java under AWS documentation which give me idea about coding part in java for connecting my java code with AWS and uploading and hosting index.html on AWS.

I write the code which is submitted to GitLab and at the end of this document. My JAVA code created the bucket named as freezingfire and then uploads a file[2] in that bucket which is named as index.html.

After uploading the file to S3 bucket, we have to edit the Bucket Policy[6] which makes the index.html file publicly available. After making changes to the Bucket Policy which I referred from AWS documentation to change bucket policy the index.html file will be available for public access.

At the end, I must edit the static website hosting options under Properties section of my bucket freezingfire. By default, this option is disable. I enabled this option, and selected Host a static website option where I specified index.html under the Index document section and clicked on save changes[3][4][5].

Diagram

Description automatically generated

Figure–1 Flowchart of operations

**Overall observation of the JAVA SDK :**

To access the well-known and standard libraries the JAVA developers need to access the AWS services and utilize it by using AWS SDK for JAVA[1]. This SDK offers assistance for API lifecycles like retries, data sharding, serialization and credential management. AWS SDK for JAVA allows higher level abstractions for development. It also supports many key features Non-Blocking I/O, Automatic Pagination and many more. Many JAVA tools are also supported by AWS such as IntelliJ, Visual Studio, Eclipse and so on. AWS provides the toolkit for popular IDE’s.

**Steps and Screenshots of the S3 buckets and operations :**

Firstly, I have created the AWS academy student account and logged in by using student credentials. The below given figure-2 represents the courses tab in AWS account. From the home screen I have selected the courses tab and in that I have selected the current course named as ALLFv1-17043.



Figure-2 Courses tab in AWS

After getting into the courses tab, I have selected the Modules section. figure-3 attached figure represents the modules tab in AWS. This tab contains 3 options namely : Learner’s Lab – Student Guide.pdf, Learner’s Lab – Foundational Services and End of course Feedback Survey. We must select Learner’s Lab – Foundational Services for our module work.

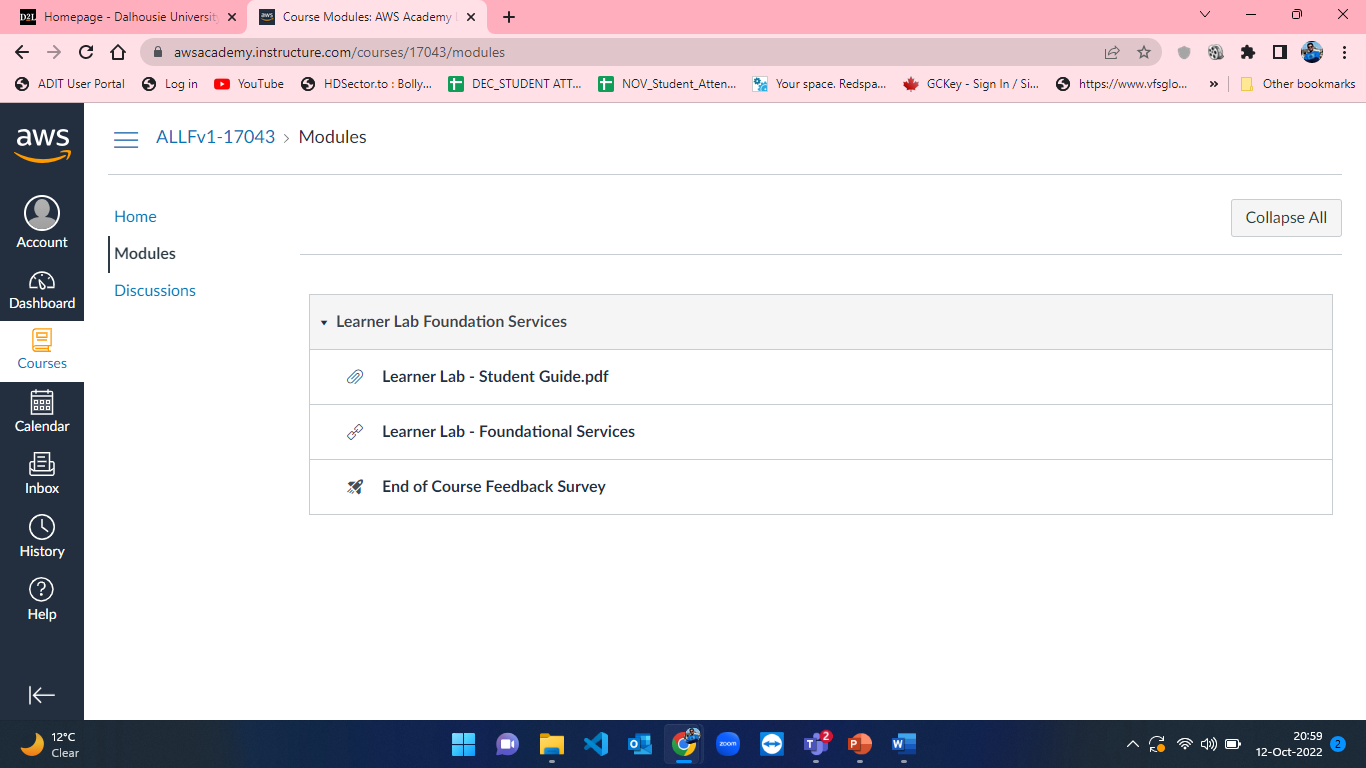


Figure-3 Module tab having Learner Lab in AWS

After clicking on Learner’s Lab – Foundational Services a basic console is opened with many options. In this I clicked on the Start Lab, which starts the AWS console. Once it is turned on the red blink near AWS will change into green. The below given two snapshots represents AWS stopped and start lab.

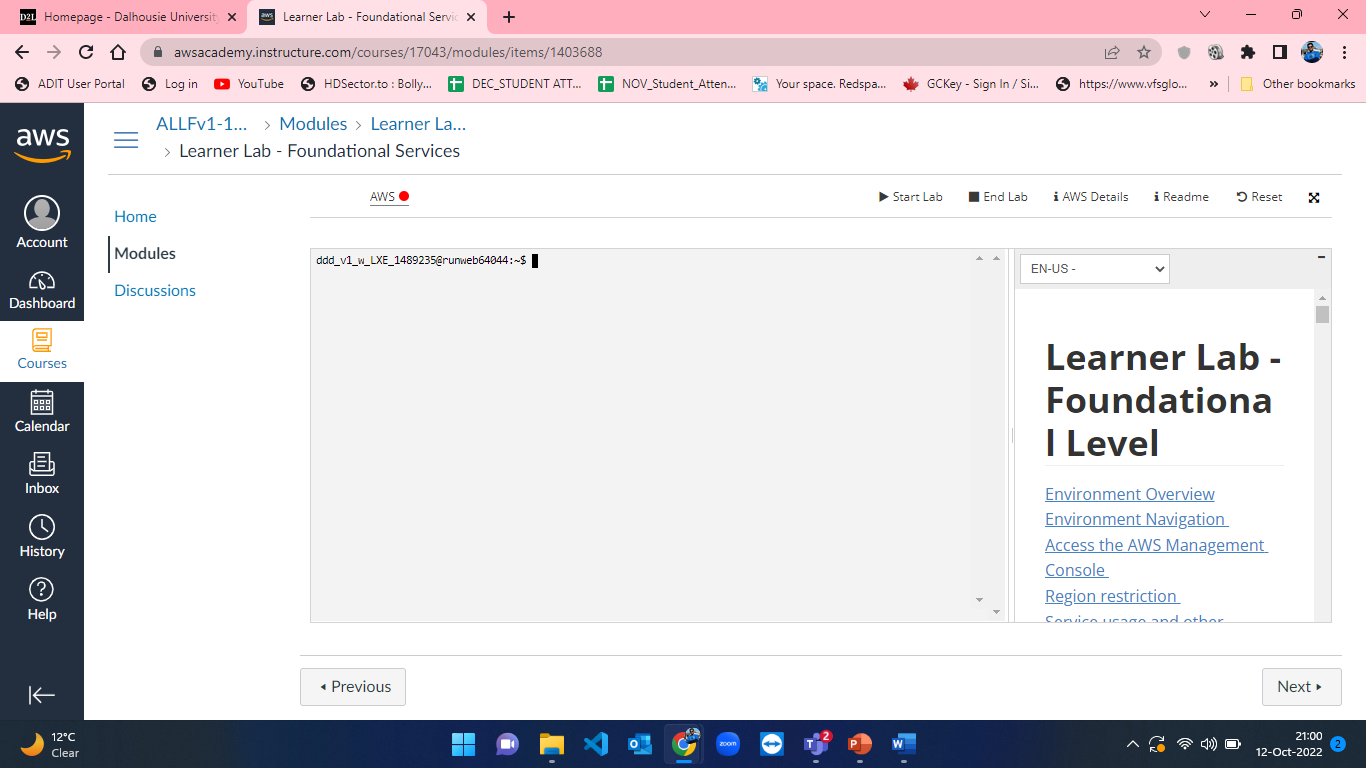


Figure-4 AWS basic home console to start lab.

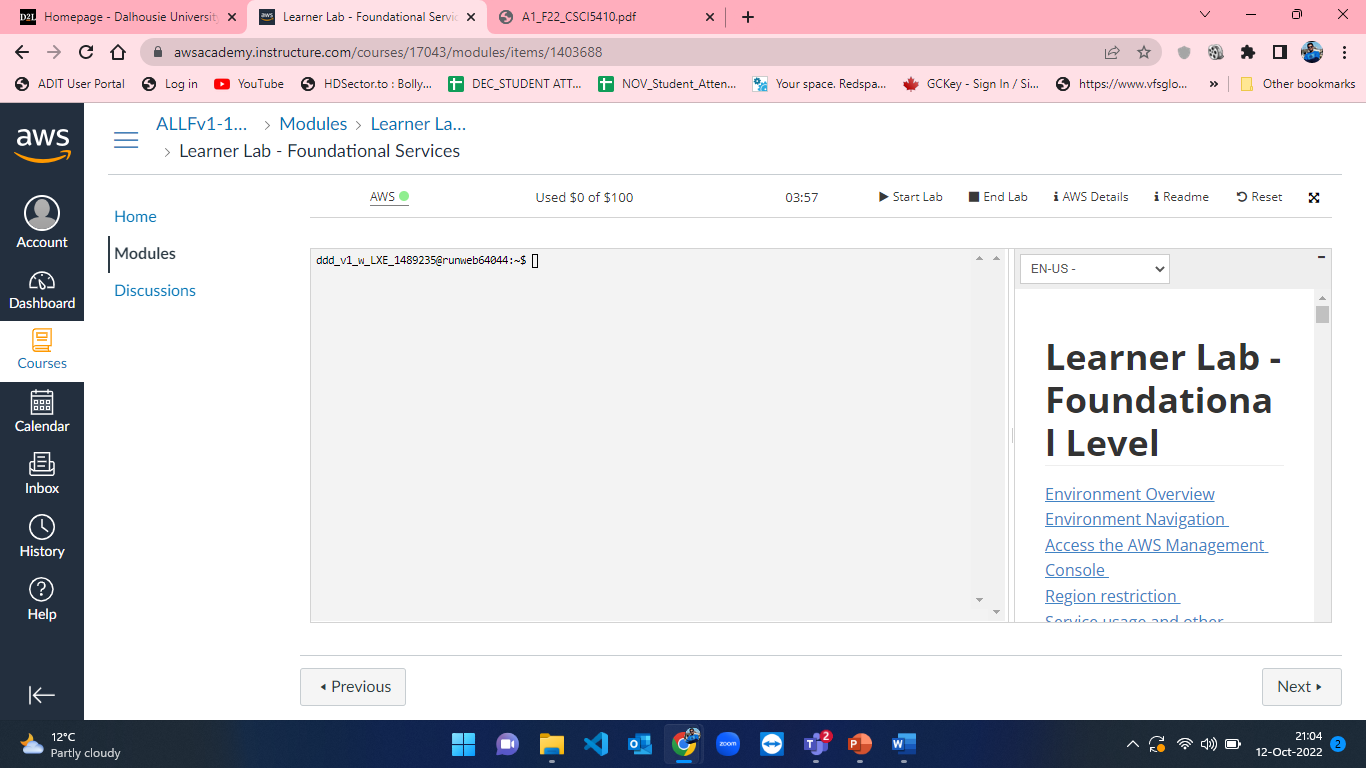


Figure-5 AWS basic home console when lab is started.

After starting lab, clicking on the AWS will open the Console Home in new tab where I selected S3 as we are working on S3 bucket for this assignment. The below attached figure shows the Console home of AWS.

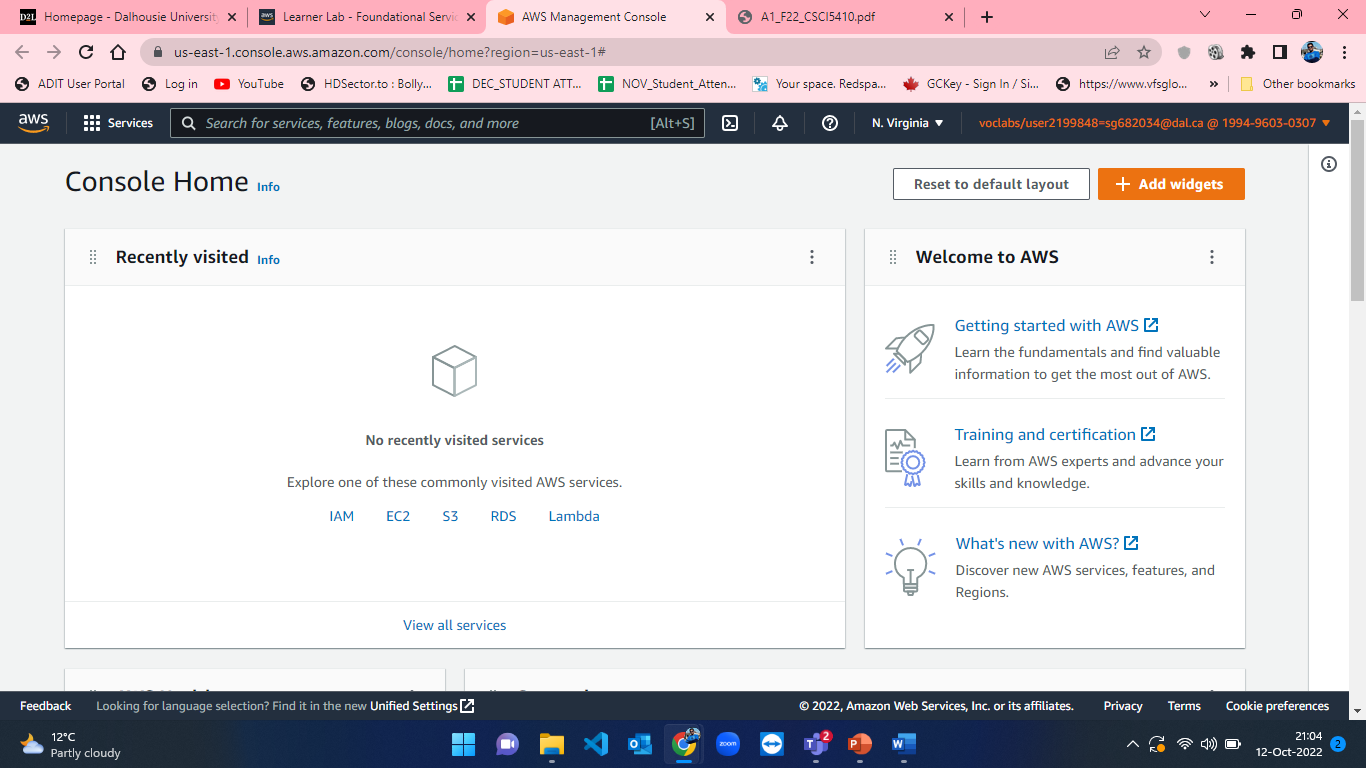


Figure-6 Console home in AWS

On the Console home, clicking on S3 will open S3 tab where we can perform operations related to bucket. Figure-7 and figure-8 shows the S3 home page with common tasks in AWS where we can create a bucket, upload the object, download the object, host a static website and many more.

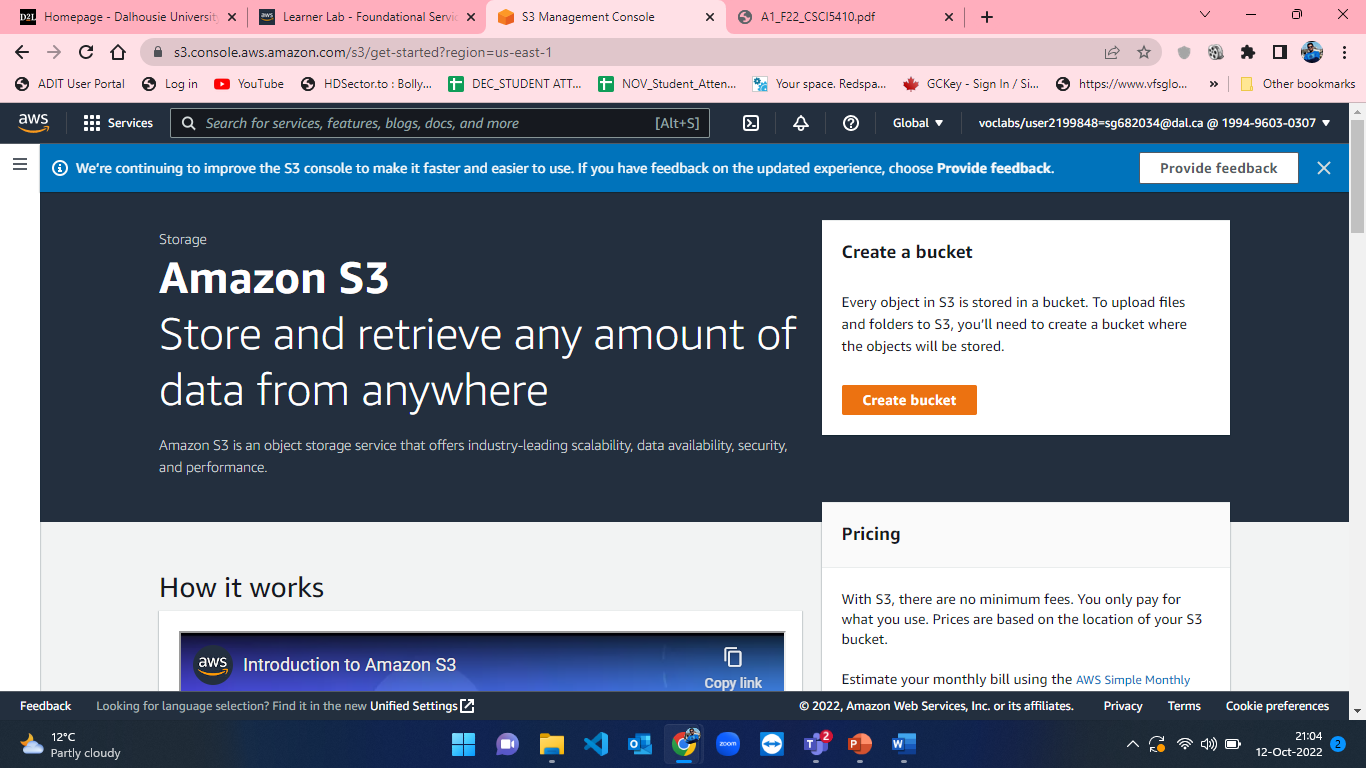
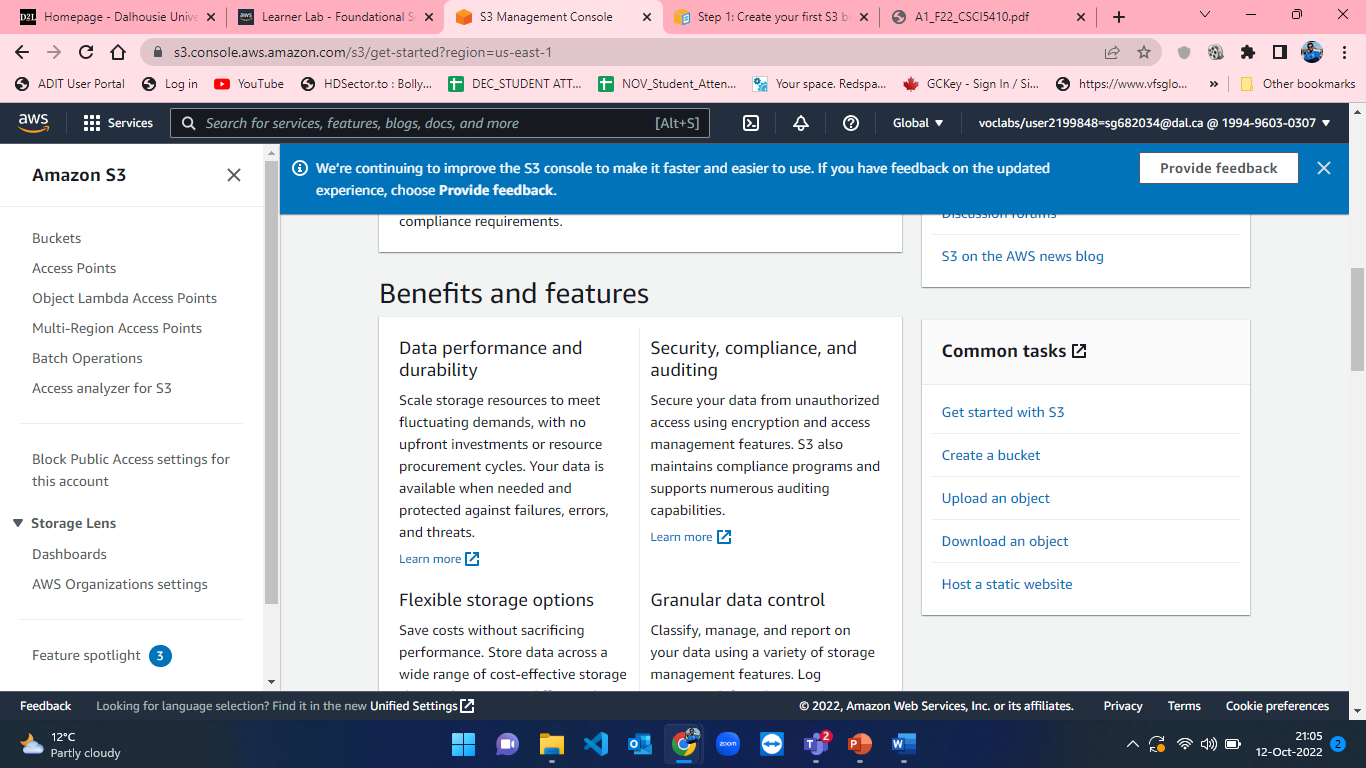


Figure-7 S3 bucket homepage for operations.



Figurer-8 Common task in S3 in AWS

Clicking on Buckets tab from the left panel, will show the number of buckets available in the account. Currently, there are no buckets that is why it shows empty. The below given figure shows buckets tab.

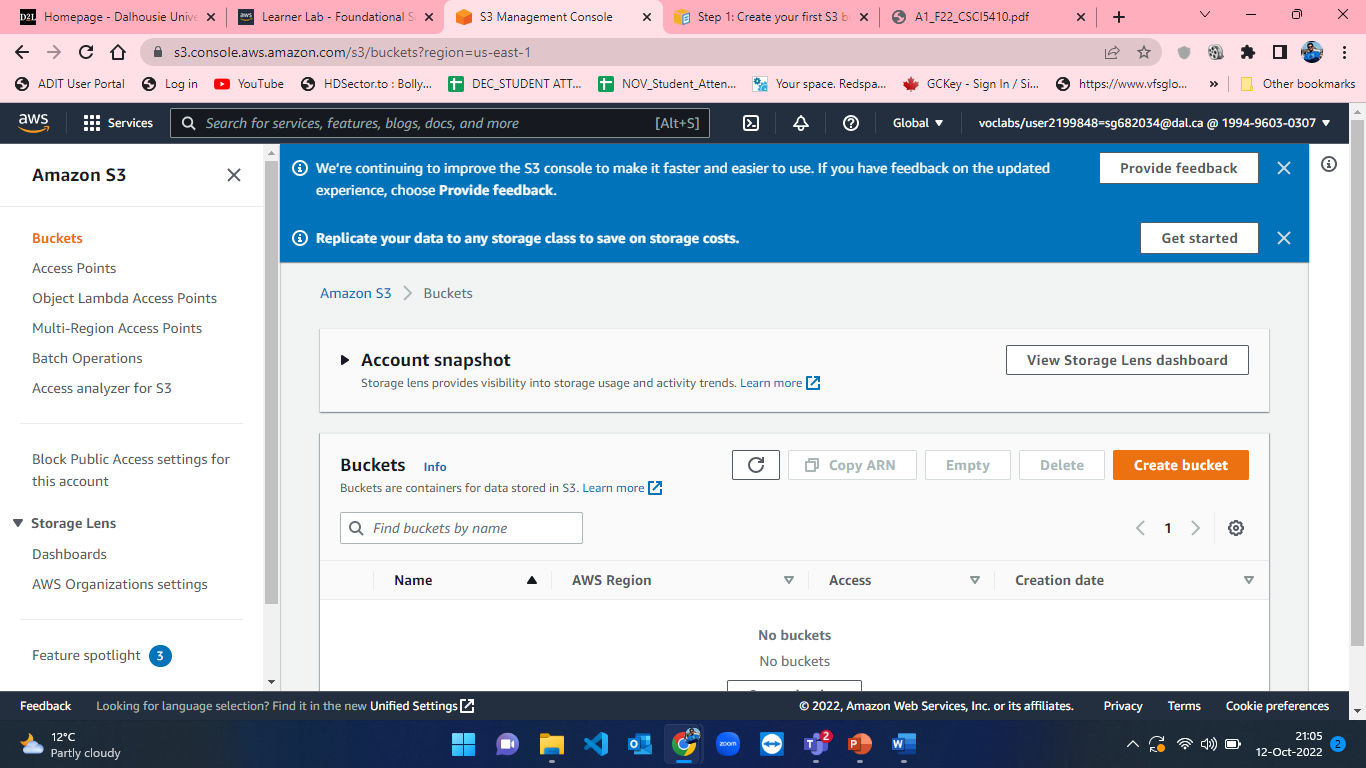


Figure-9 Buckets in AWS

Once exploration completes, first thing I must do is setting up the connection from my java code with AWS this can be done by having aws\_access\_key, aws\_secret\_access\_key, aws\_session\_token. These 3 parameters can be obtained from the main console where we started the lab. There is an option named as AWS details in that under AWS CLI it shows these credentials. Figure-10 presents the parameters for the connection.

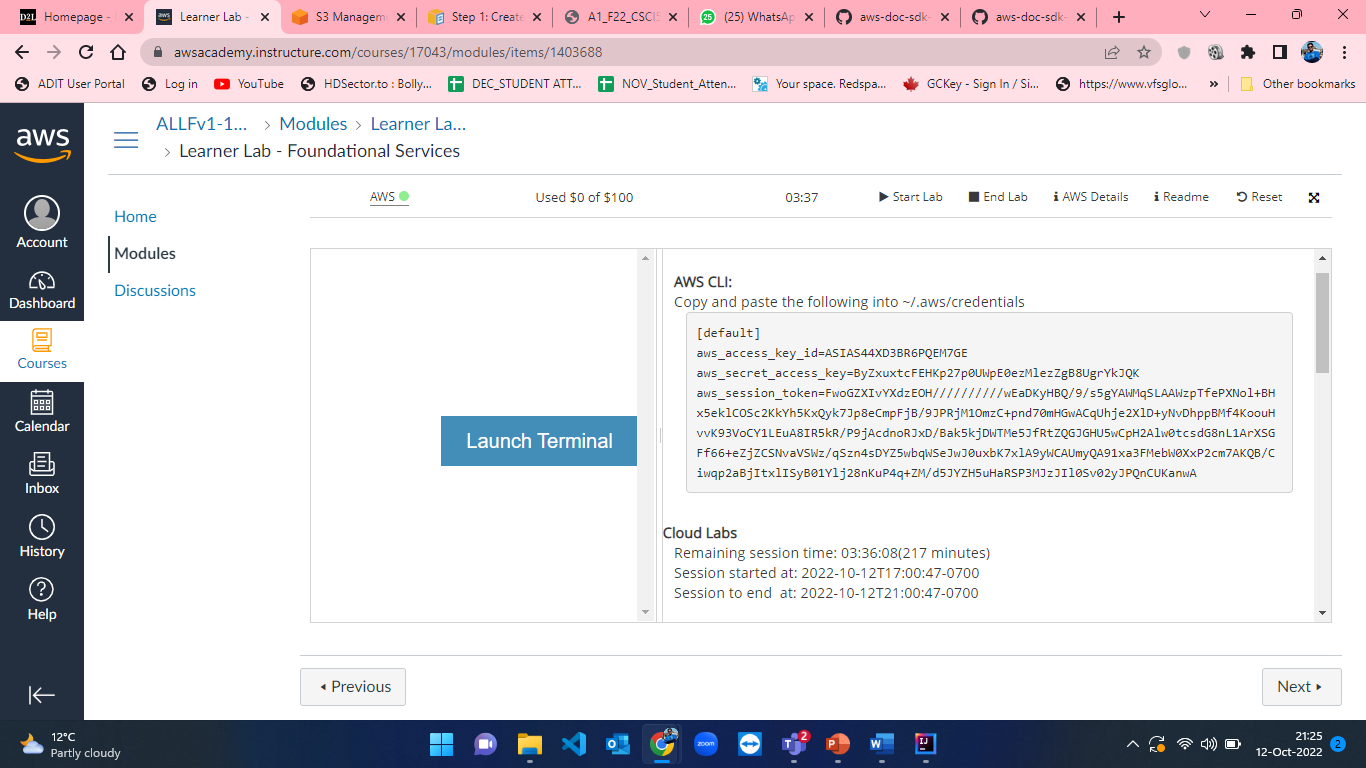


Figure-10 Parameters for the connection in AWS.

After writing the java code and setting up the connection with AWS, I wrote JAVA code to create the bucket. The below attached snapshot represents the created bucket named as freezingfire.

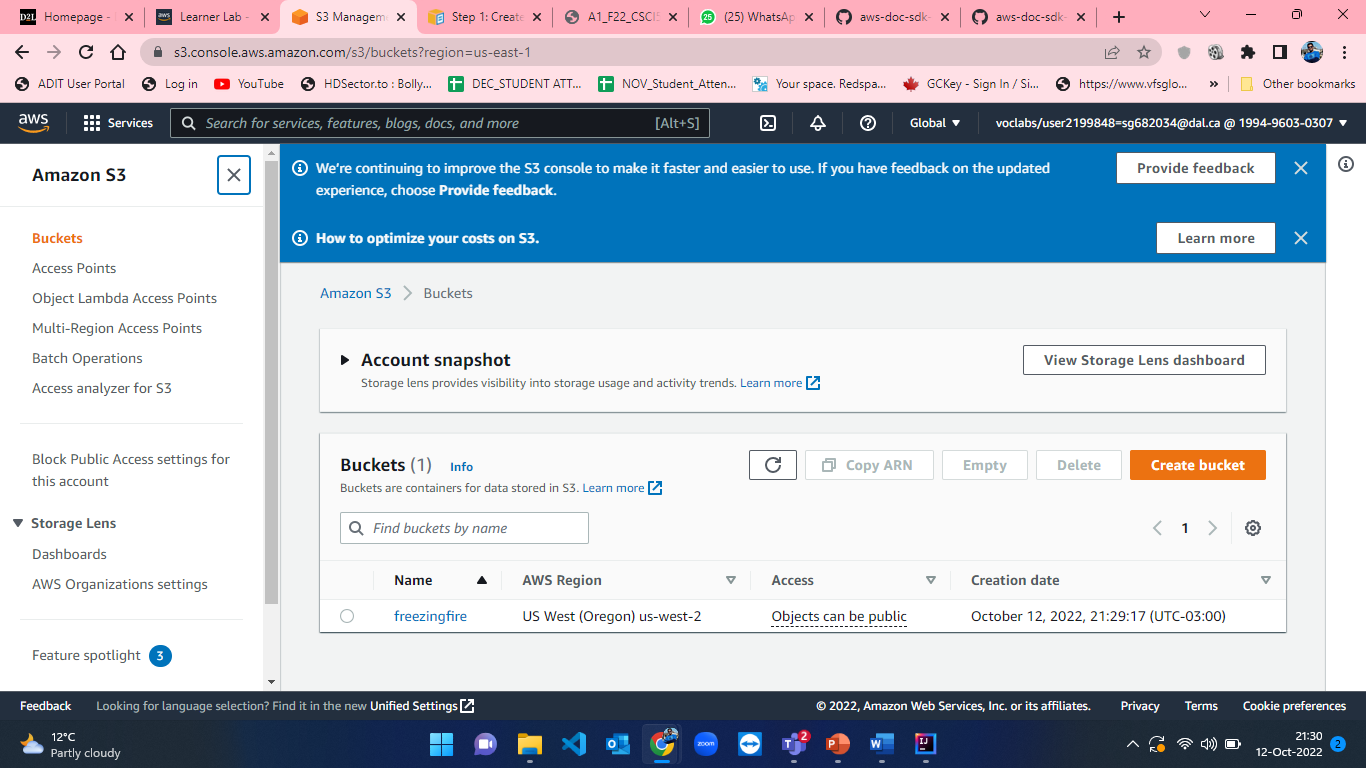


Figure-11 Bucket created in AWS

After creating bucket, I have to upload the index.html file to the newly created bucket. This is done by java code. Figure-11 represents the object uploaded under freezingfire bucket.

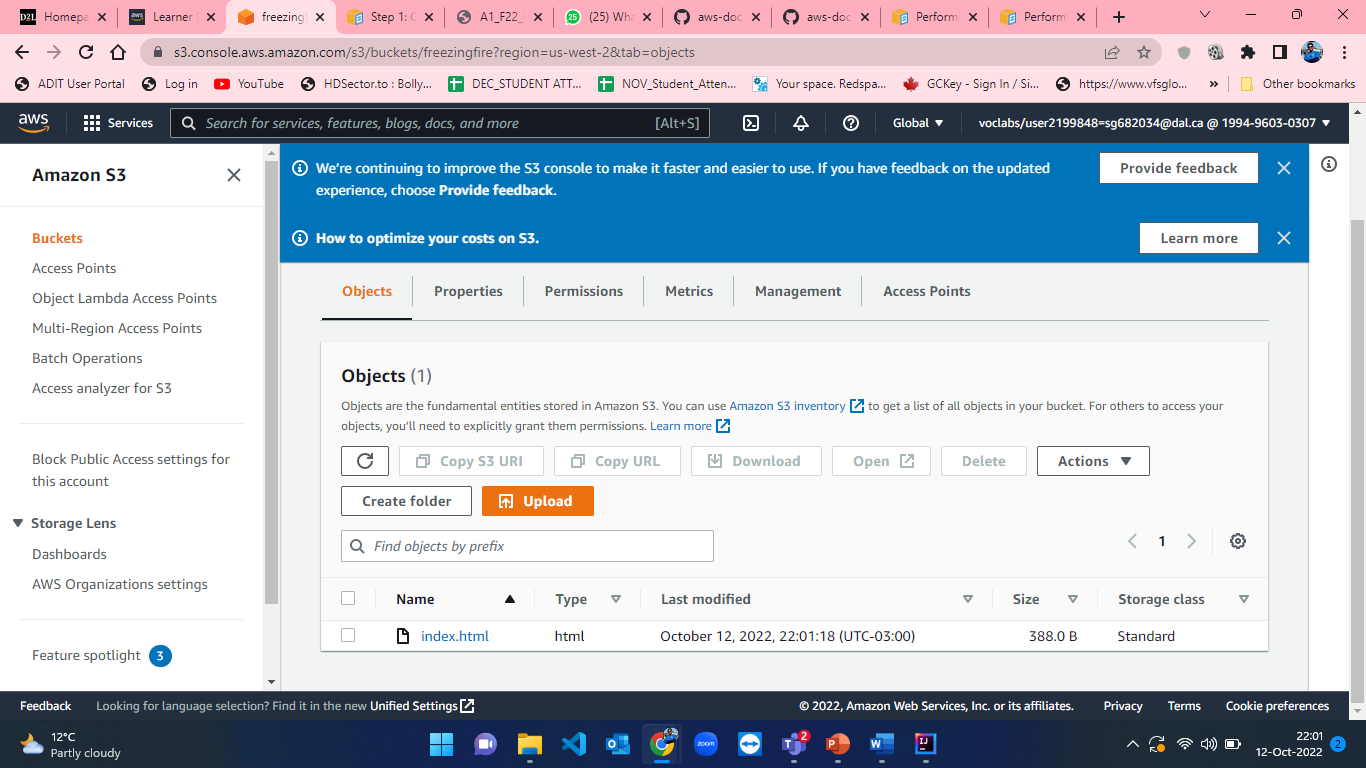


Figure-11 Object uploaded under created in AWS

After uploading object to the bucket, I have to change the bucket policy to give access the object by other accounts. I have used AWS documentation to get the JSON script for making the public access of the object. The given below snapshots shows the changes in bucket policy.

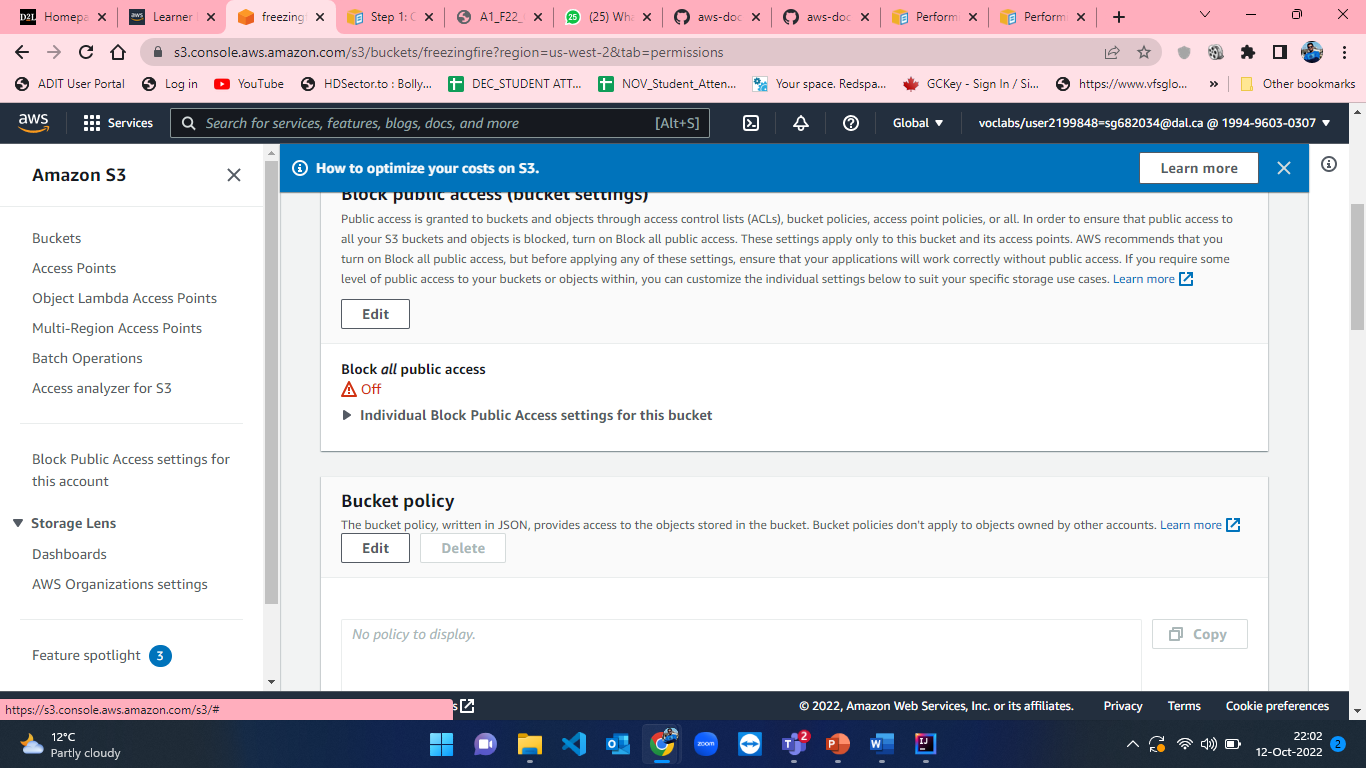


Figure-12 Bucket Policy in default mode.

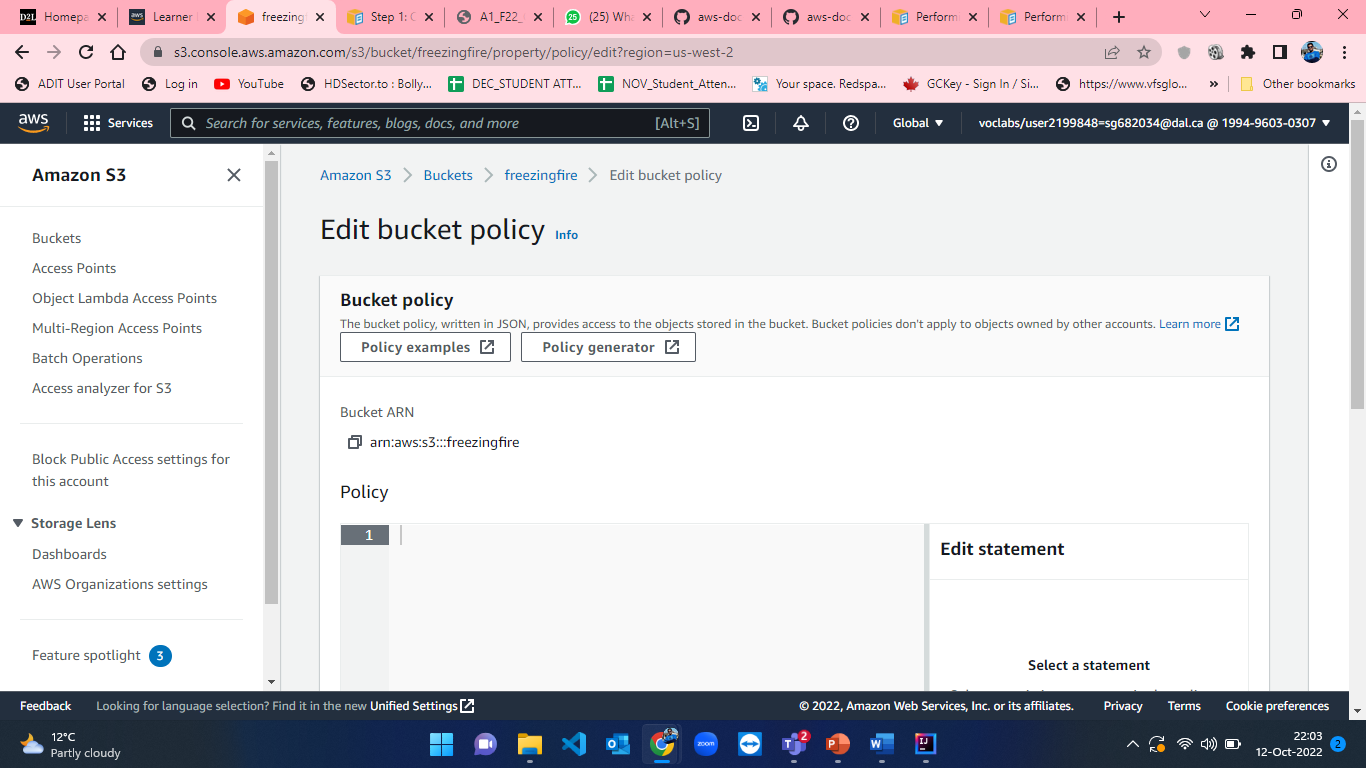


Figure-13 Editing blank bucket policy.

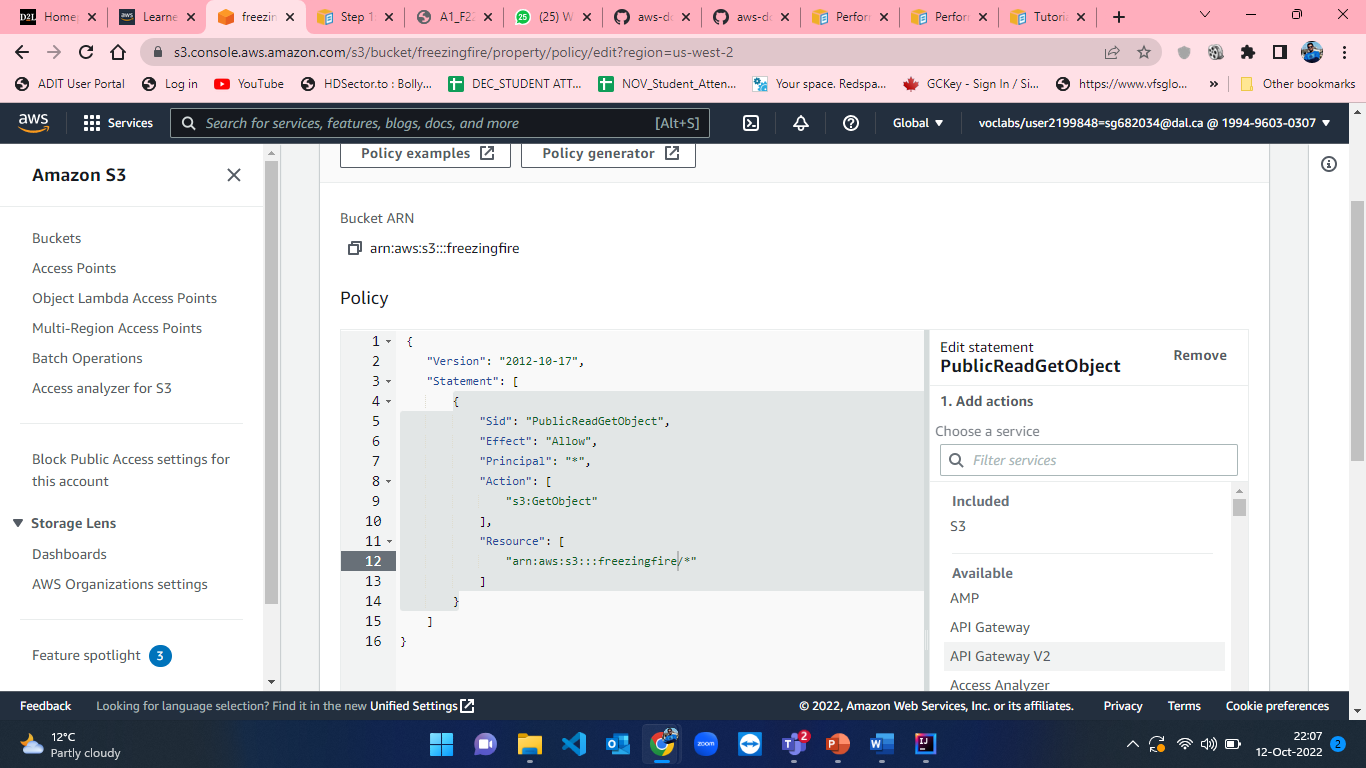


Figure-14 Adding JSON script for public access in AWS.

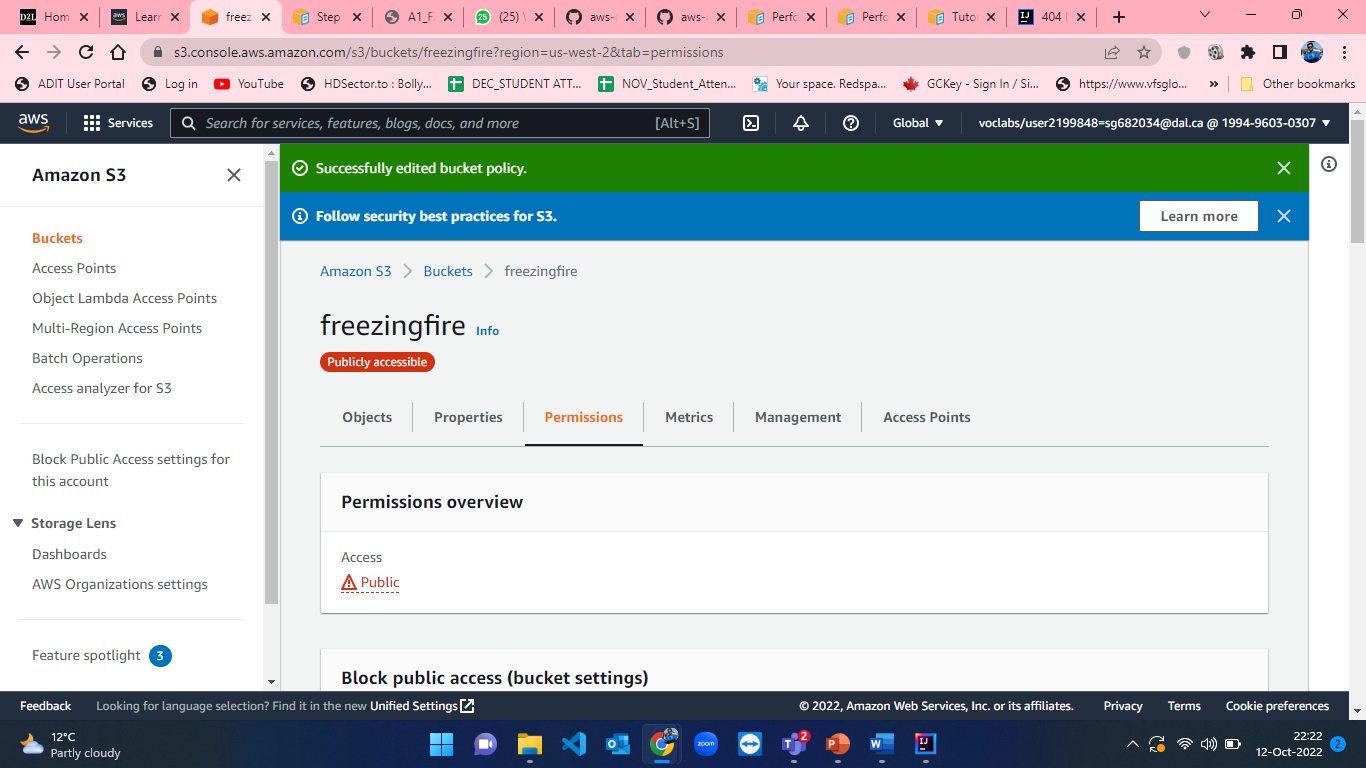


Figure-15 Successfully saving the bucket policy.

For static web hosting, we have to enable the static web hosting. This operations is shown in the below attached figures.

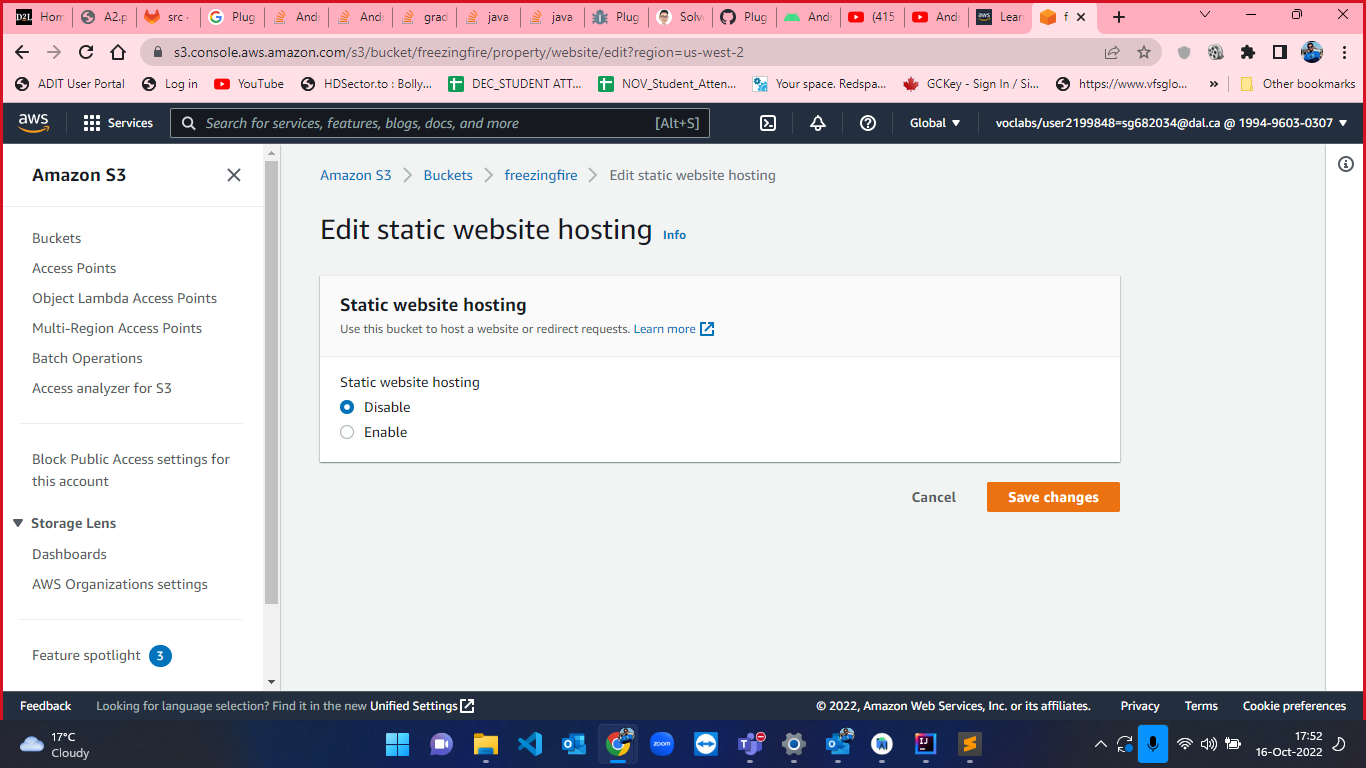


Figure-16 Default disable option in static web hosting.

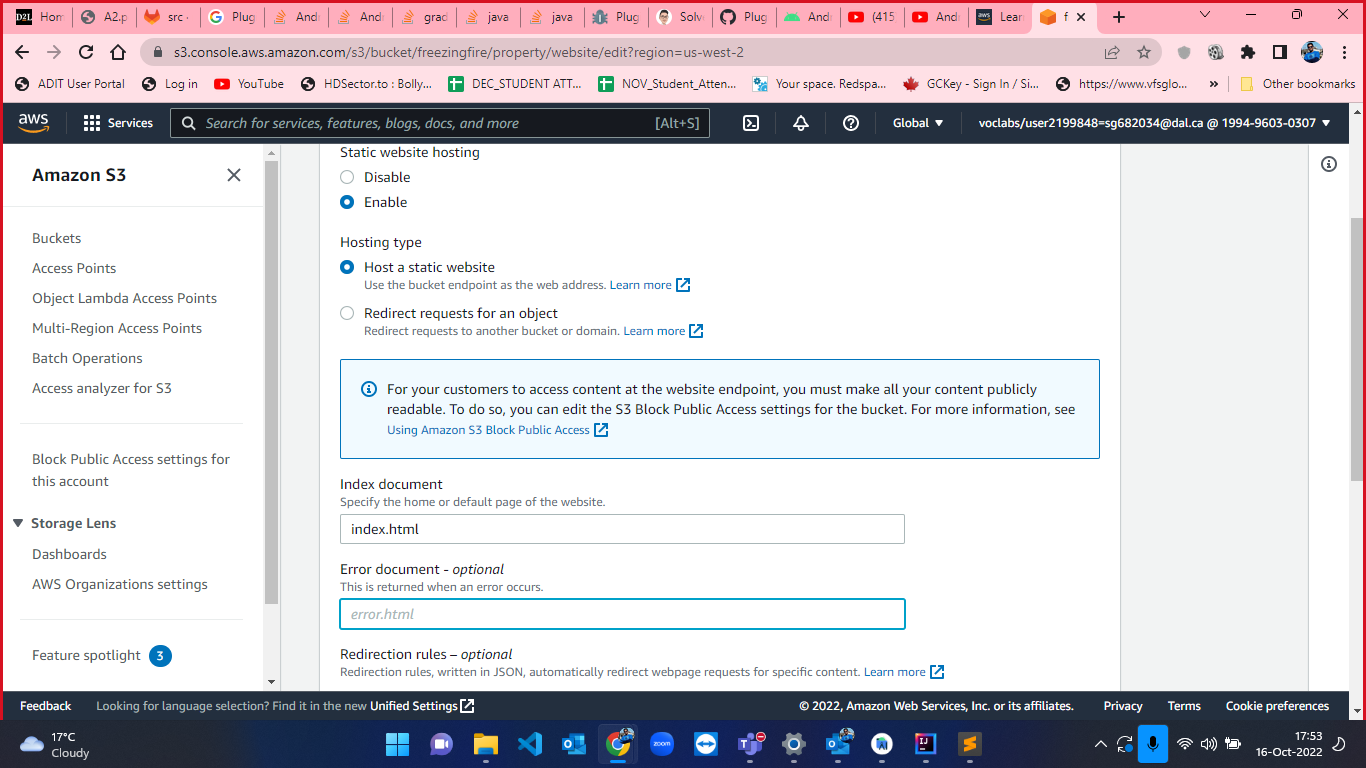


Figure-17 Performing operations to enable the static hosting in AWS

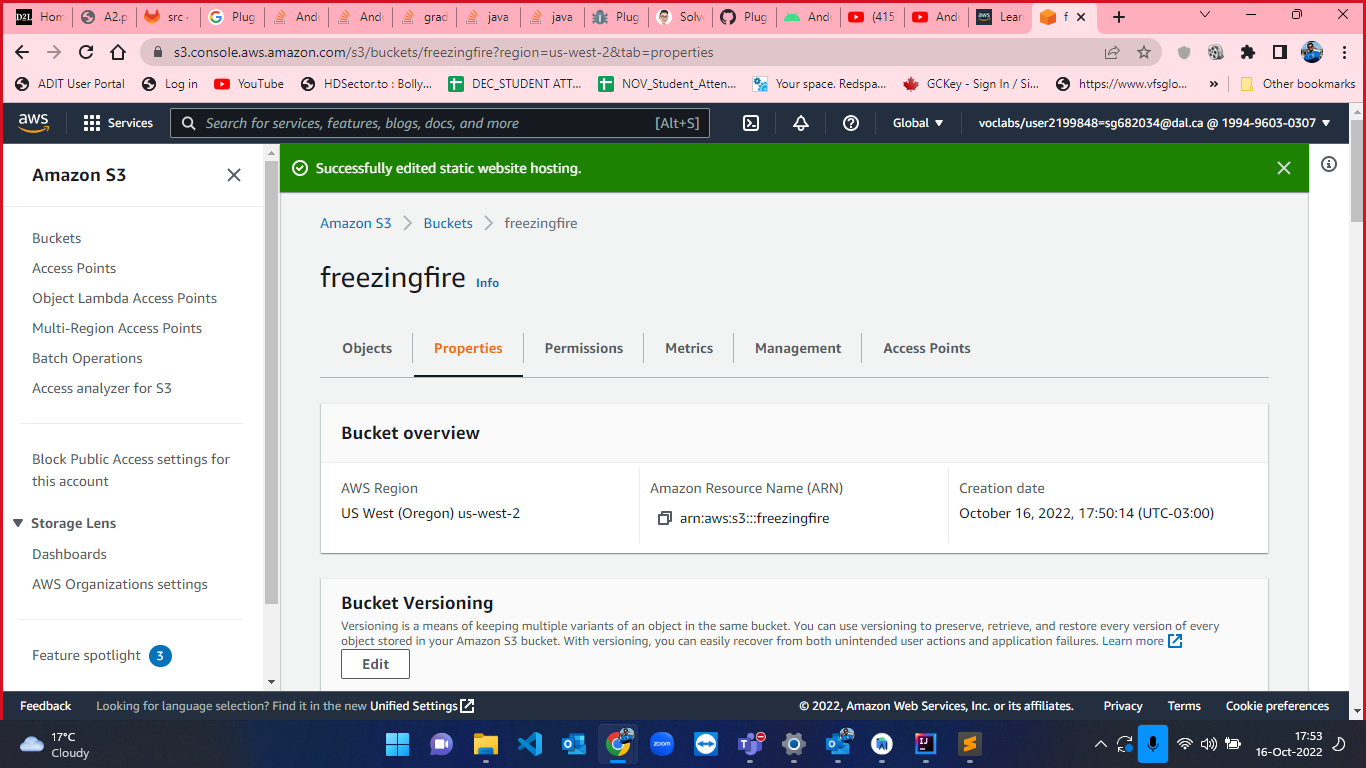


Figure-18 Successfully edited static web hosting in AWS.

After setting up the static web hosting and the bucket policy we can see the webpage index.html page publicly. This can be seen by the attached figure-19.

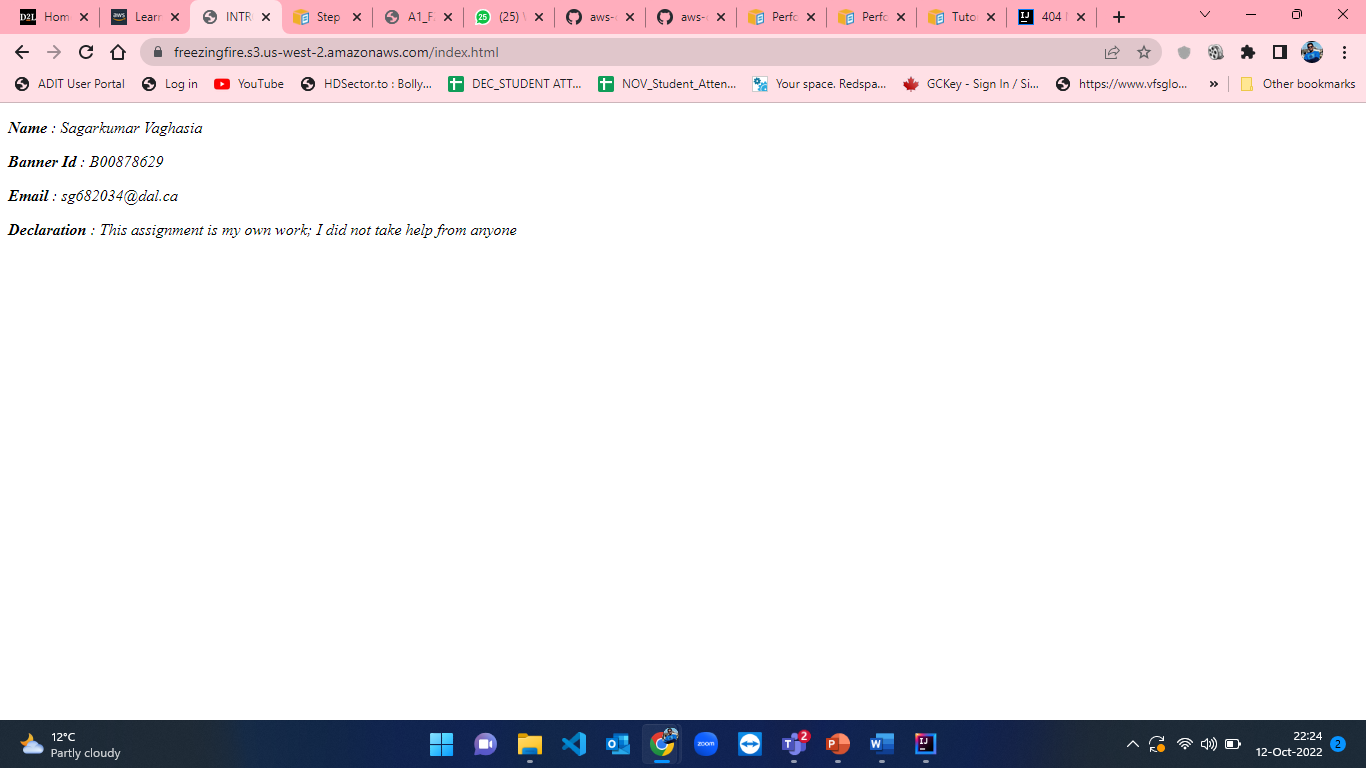


Figure-19 Hostel index.html in AWS.

**JAVA Code:**

**CreateBucket.java**

import com.amazonaws.auth.AWSStaticCredentialsProvider;  
import com.amazonaws.auth.BasicSessionCredentials;  
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;  
  
  
public class CreateBucket {  
  
 //https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html  
 public static Bucket getBucket(String bucket\_name) {  
 final AmazonS3 s3 = AmazonS3ClientBuilder.*standard*()  
 .withRegion(Regions.*DEFAULT\_REGION*)  
 .withCredentials(new AWSStaticCredentialsProvider(new BasicSessionCredentials(  
 "ASIAS44XD3BR76XC7G4Q",  
 "LsNDVqZeLo07YX4oNdzMFdPvniluKStSO5AcPjwF",  
 "FwoGZXIvYXdzED4aDEZJW3l1fZTHZWy2BSLAARz5s62574lTWroOjeeHpywYRmoGRcRp6PLPWLSIxG8MKX+df+yaY+SapQKNtzg8a8ZeIqyLL8jyGRC4c34ZaKlx7EopLbZ17s1om5hoPc+kYg1G+n03MS759AlQDpp3vDIzhZ5HmW6nsCTOOuAKzYhdUj7a4RZ4ea6uqJtYwhkI4MRQQabDoMqwTi4Nl4kKBpzXlUHEb/5IDBz7wcsLDQ/LUV5pT4iUjF5ZwVJmHZc01ouDBbAZnny6l10d+bT3uijr1bGaBjItUjWLg1l1eYP2Fwv7f7byrzTwW3fpoFKYJeS4BHHbQChOCK3yXIzuqQIEBaP5"  
 ))).build();  
 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;  
 }  
  
 //https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html  
 public static Bucket createBucket(String bucket\_name) {  
 final AmazonS3 s3 = AmazonS3ClientBuilder.*standard*()  
 .withRegion(Regions.*DEFAULT\_REGION*)  
 .withCredentials(new AWSStaticCredentialsProvider(new BasicSessionCredentials(  
 "ASIAS44XD3BR76XC7G4Q",  
 "LsNDVqZeLo07YX4oNdzMFdPvniluKStSO5AcPjwF",  
 "FwoGZXIvYXdzED4aDEZJW3l1fZTHZWy2BSLAARz5s62574lTWroOjeeHpywYRmoGRcRp6PLPWLSIxG8MKX+df+yaY+SapQKNtzg8a8ZeIqyLL8jyGRC4c34ZaKlx7EopLbZ17s1om5hoPc+kYg1G+n03MS759AlQDpp3vDIzhZ5HmW6nsCTOOuAKzYhdUj7a4RZ4ea6uqJtYwhkI4MRQQabDoMqwTi4Nl4kKBpzXlUHEb/5IDBz7wcsLDQ/LUV5pT4iUjF5ZwVJmHZc01ouDBbAZnny6l10d+bT3uijr1bGaBjItUjWLg1l1eYP2Fwv7f7byrzTwW3fpoFKYJeS4BHHbQChOCK3yXIzuqQIEBaP5"  
 ))).build();  
 Bucket b = null;  
 if (s3.doesBucketExistV2(bucket\_name)) {  
 System.*out*.format("Bucket %s already exists.\n", bucket\_name);  
 b = *getBucket*(bucket\_name);  
 } else {  
 try {  
 b = s3.createBucket(bucket\_name);  
 } catch (AmazonS3Exception e) {  
 System.*err*.println(e.getErrorMessage());  
 }  
 }  
 return b;  
 }  
  
 public static void main(String[] args) {  
  
 UploadIndex uploadIndex = new UploadIndex();  
 uploadIndex.uploadfile();  
  
 final String USAGE = "\n" +  
 "CreateBucket - create an S3 bucket\n\n" +  
 "Usage: CreateBucket <bucketname>\n\n" +  
 "Where:\n" +  
 " bucketname - the name of the bucket to create.\n\n" +  
 "The bucket name must be unique, or an error will result.\n";  
  
 String bucket\_name = "freezingfire";  
  
 System.*out*.format("\nCreating S3 bucket: %s\n", bucket\_name);  
 Bucket b = *createBucket*(bucket\_name);  
 if (b == null) {  
 System.*out*.println("Error creating bucket!\n");  
 } else {  
 System.*out*.println("Done!\n");  
 }  
 }  
}

**UploadIndex.java**

import com.amazonaws.AmazonServiceException;  
import com.amazonaws.auth.AWSStaticCredentialsProvider;  
import com.amazonaws.auth.BasicSessionCredentials;  
import com.amazonaws.regions.Regions;  
import com.amazonaws.services.s3.AmazonS3;  
import com.amazonaws.services.s3.AmazonS3ClientBuilder;  
  
  
  
import java.io.File;  
  
  
public class UploadIndex {  
// https://www.codejava.net/aws/upload-file-to-s3-java-console  
 public void uploadfile(){  
 CreateBucket createBucket = new CreateBucket();  
  
 final AmazonS3 s3 = AmazonS3ClientBuilder.*standard*()  
 .withRegion(Regions.*DEFAULT\_REGION*)  
 .withCredentials(new AWSStaticCredentialsProvider(new BasicSessionCredentials(  
 "ASIAS44XD3BR76XC7G4Q",  
 "LsNDVqZeLo07YX4oNdzMFdPvniluKStSO5AcPjwF",  
 "FwoGZXIvYXdzED4aDEZJW3l1fZTHZWy2BSLAARz5s62574lTWroOjeeHpywYRmoGRcRp6PLPWLSIxG8MKX+df+yaY+SapQKNtzg8a8ZeIqyLL8jyGRC4c34ZaKlx7EopLbZ17s1om5hoPc+kYg1G+n03MS759AlQDpp3vDIzhZ5HmW6nsCTOOuAKzYhdUj7a4RZ4ea6uqJtYwhkI4MRQQabDoMqwTi4Nl4kKBpzXlUHEb/5IDBz7wcsLDQ/LUV5pT4iUjF5ZwVJmHZc01ouDBbAZnny6l10d+bT3uijr1bGaBjItUjWLg1l1eYP2Fwv7f7byrzTwW3fpoFKYJeS4BHHbQChOCK3yXIzuqQIEBaP5"  
 ))).build();  
  
 try{  
 System.*out*.println("File is being uploaded");  
 s3.putObject("freezingfire", "index.html", new File("D:\\Sagar\\SEM-2 (SEP)\\Serverless Data Processing\\Assignment 1\\CodePart2\\src\\main\\java\\index.html"));  
  
 } catch(AmazonServiceException e){  
 System.*err*.println(e.getErrorMessage());  
 System.*exit*(1);  
 }  
  
 }  
  
  
}

**index.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title> INTRODUCTION </title>  
 <p><i><b>Name </b> : Sagarkumar Vaghasia </i></p>  
 <p><i><b> Banner Id </b> : B00878629 </i></p>  
 <p><i><b>Email</b> : sg682034@dal.ca </i></p>  
 <p><i><b>Declaration</b> : This assignment is my own work; I did not take help from anyone </i></p>  
</head>  
<body>

**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>CodePart2</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 </properties>  
  
 https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html  
 <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>org.bouncycastle</groupId>  
 <artifactId>bcprov-jdk16</artifactId>  
 <version>1.45</version>  
 </dependency>  
 <dependency>  
 <groupId>com.amazonaws</groupId>  
 <artifactId>aws-java-sdk-kms</artifactId>  
 </dependency>  
 </dependencies>  
  
</project>

**References :**

[1]Amazon.com. [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html. [Accessed: 12-Oct-2022].

[2]N. H. Minh, “Upload file to S3 using AWS Java SDK - Java console program,” Codejava.net. [Online]. Available: https://www.codejava.net/aws/upload-file-to-s3-java-console. [Accessed: 12-Oct-2022].

[3]Amazon.com. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html. [Accessed: 12-Oct-2022].

[4]Amazon.com. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteEndpoints.html. [Accessed: 12-Oct-2022].

[5]Amazon.com. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/EnableWebsiteHosting.html. [Accessed: 12-Oct-2022].

[6]Amazon.com. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteAccessPermissionsReqd.html. [Accessed: 12-Oct-2022].