DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

Assignment - 1

In
The Class of

CSCI5410: Serverless

By

Deep Patel [B00865413]

Submitted to

Prof. Saurabh Dey

Department of Computer Science

Dalhousie university.

Date: 25th May 2021

Task-1

Paper Review:

The services that provide additional support for deploying and managing the distributed applications like docker, Kubernetes, Mesos and Swarm were originally created to provide a load-balanced and stateless services and running a database clusters due to their enhanced auto recovery and location transparency. Performance of each one of those technologies were evaluated in the paper [1] based on their ability of deploying and managing NoSQL database clusters [1].

The trending technology like NoSQL Database are about to become more and more important because of their supporting to the different kind of storages and their balanced architecture between their consistency, availability and partition tolerance based on which paper [1] is relatable. Even with lots of configuration parameters maintenance becomes a very crucial and unavoidable task [1]. The performance analysis of the containers like Docker Swarm and Kubernetes as database study is also mentioned in the paper based on costs calculated for each container for managing and deploying NoSQL Database cluster. The important applications of the performance on various containers based on deploying and managing NoSQL database clusters [1].

Pre-existing studies were mostly focused on single container comparison with the virtual Machin running on a Linux server [1]. It was Sharma et al who discovered that hybrid models which are nothing but a containerized Redis instances of the virtual machines which can be used as a comparison model with some different virtual machine which can work somewhat batter then the previous studies based on a single comparison. Most CO Systems (Container Orchestrated Systems) were originated from Apache and Google and Verma et al. explained Kubernetes (Ancestors of Borg System) provides efficient and accurate resource allocating process using those [1]. Some good examples of those CO Systems are Docker Swarm, Kubernetes, Mesos and OpenShift (Based on K8s) which are using private OpenStack Clouds. Some of the common features of those CO Systems which are similar to the clusters in some ways are: Highly Customizable Schedular, Node failure detection, Persistent volumes, Virtual Networks and containers and host ports. CO Systems were not that much suited for the deployment and management of the application comparatively with the clusters as they are in conflicts with some algorithms of database systems like Cassandra and MongoDB [1]. The main experiment was done on measurement of the performance overhead of Cluster Orchestrated systems while performing normal operations. The expected outcome of the experiment was supposed to be some features of the CO systems like: Customizable Scheduling, Container eviction on node failure and persistent volumes were higher than the expected amount. The outcome of the other features like Virtual networks and service proxies were not as much significant as above mentioned three features represented [1].

Evaluation of the cluster was done by using YCSB Benchmark for running the performance evaluation experiment [1]. The test results of the operations were according to specific ratio like 90% read operations and 10% update operations and so on which suggests less scalability of the CO based databases and very less performance over head for those which are using Cloud prevised IP endpoints and Host ports [1].

Task-1

Flow Chart:

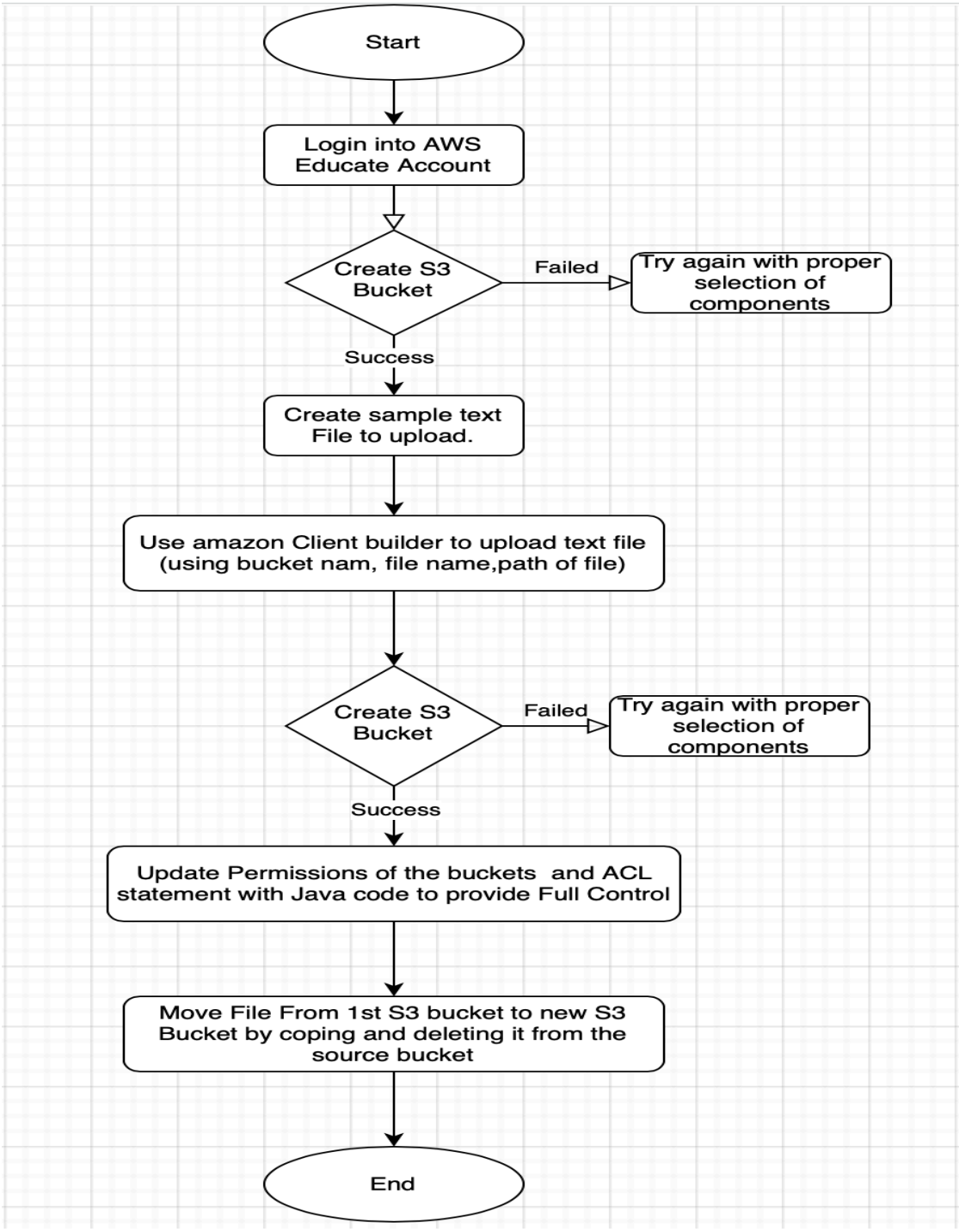

Figure 1: Flow Chart Of Task-2 Process

Observations:

The task-2 covers overall scenarios to do using AWS S3 buckets. Here the bucket creation in AWS Educate account from console as well as using Java code is implemented using various AWS libraries.

Tasks performed inside the S3 buckets were as mentioned below;

- Create S3 bucket using AWS SDK for java same as bucket creation using console.
- Updating access permissions to the buckets through console and java code
- Blocking or disabling the public access using java code
- Updating Full Control access for bucket owner through java code.
- Copying file from one bucket to another using java code.
- Deleting file from bucket using java code.

Screen Shots of S3 Bucket

1. First Bucket created from AWS Console after login into AWS Educate Account.

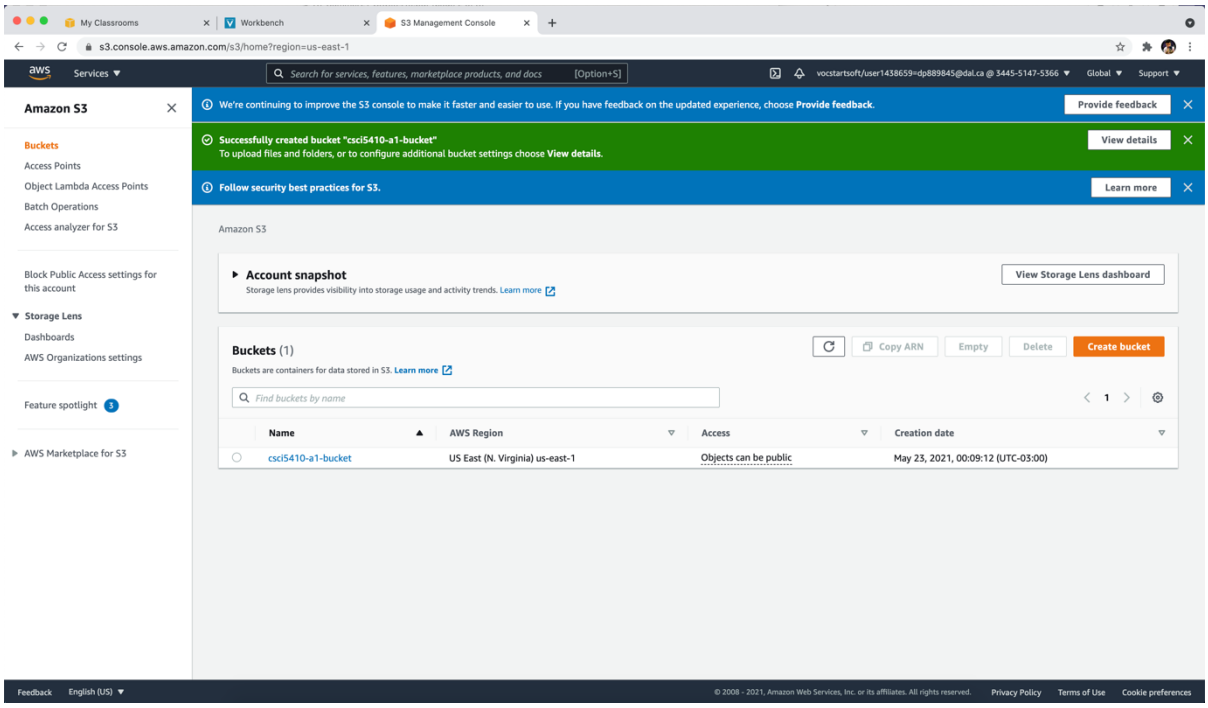

Figure 2:Bucket Created Successfully

2. File added into AWS s3 bucket using java code as mentioned below:

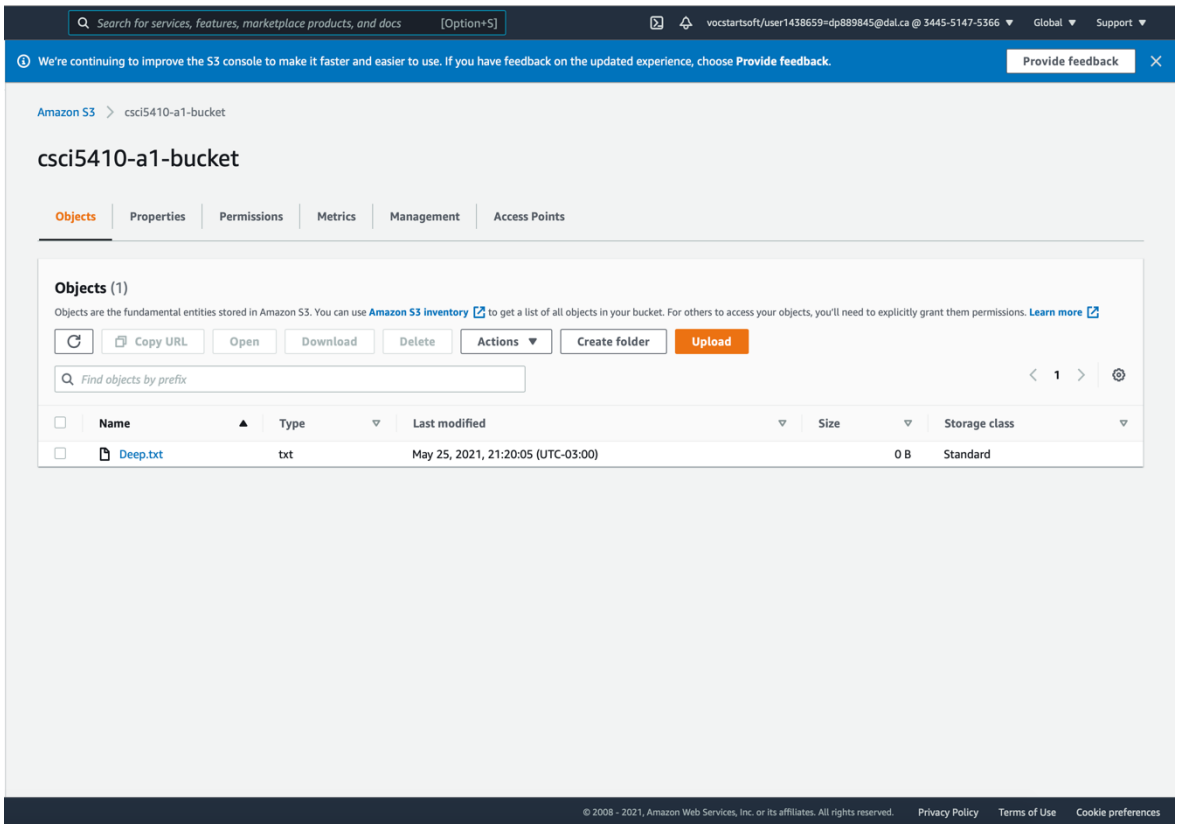

Figure 2: File Uploaded

- Code [2]:

```
import java.io.File;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

public class UploadFile {

	public void uploadFileToS3Bucket(String
filePath) {

	final AmazonS3 s3 =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).
build();

	String bucketName = "csci5410-a1-bucket";
	String fileName = "Deep.txt";

	s3.putObject(bucketName,fileName,new File(filePath));
	System.out.println("File Successfully Uploded!!");
}
}
```

3. File Uploaded Successfully:

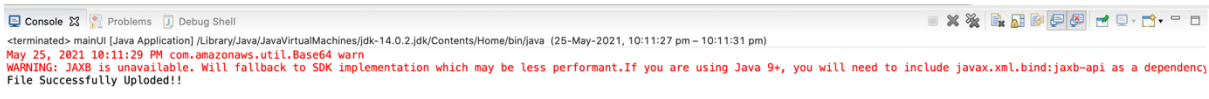

Figure 3: Successful File updating

4. Bucket Creation From java code

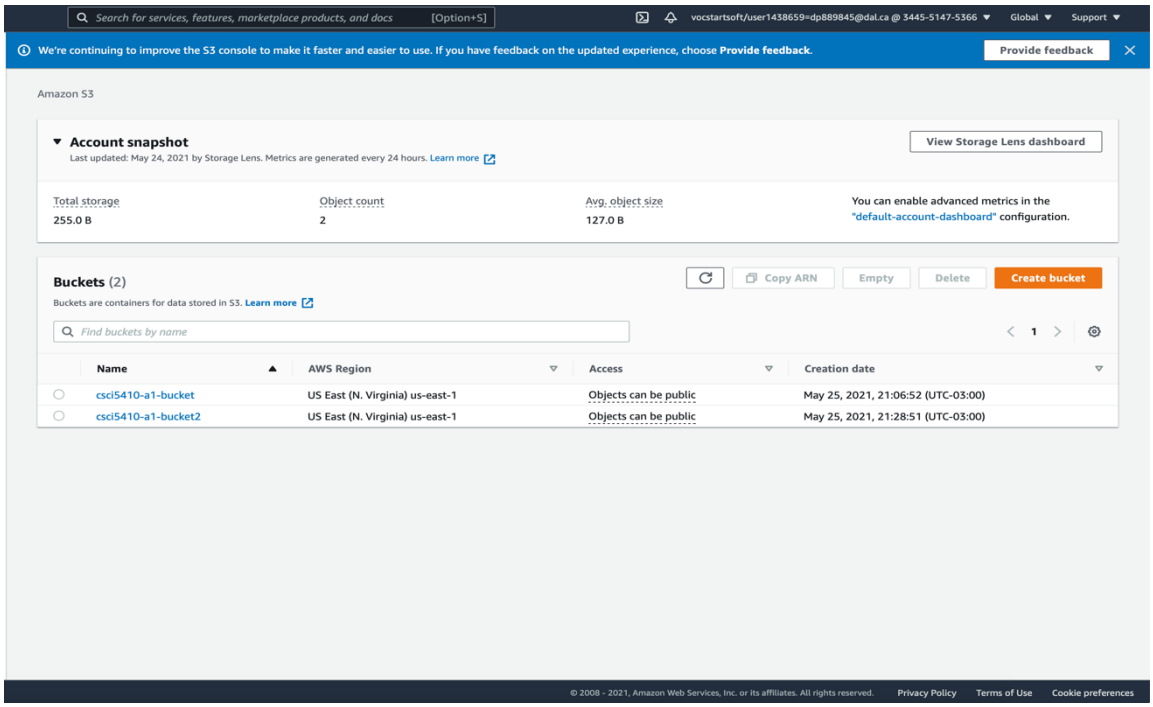

Figure 4: Another Bucket Created from java code

- Code [3]:

```
final static String BUCKET_NAME = "csci5410-a1-bucket2";
final AmazonS3 s3Bucket =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).
build();

public void create_S3_Bucket() {

      if (s3Bucket.doesBucketExistV2(BUCKET_NAME)) {
          System.out.format("Bucket %s already exists.\n",
BUCKET_NAME);
      }
      else {
          try {
            s3Bucket.createBucket(BUCKET_NAME);

            System.out.println("Bucket Created Successfully");
          } catch (AmazonS3Exception e) {
              System.err.println(e.getErrorMessage());
          }
      }
}
```

5. Bucket Created Successfully:

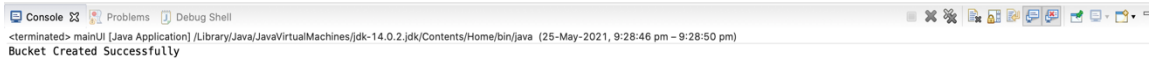

Figure 5: Java output for successfully creation of bucket

6. Block All access for that bucket

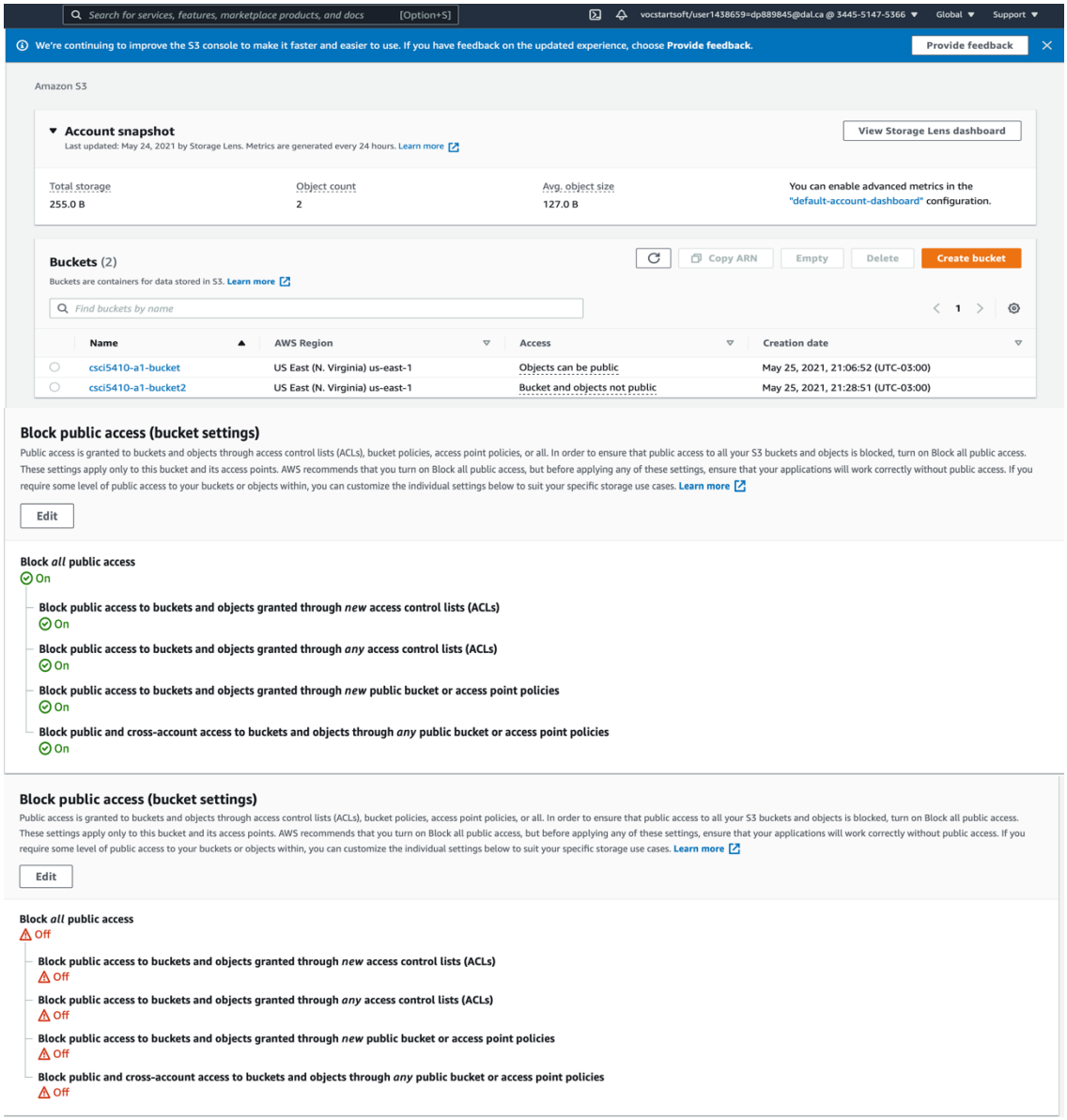

Figure 6: Public Access of the buckets are blocked

- Code [4]:

```
final static String BUCKET_NAME = "csci5410-a1-bucket2";
final AmazonS3 s3Bucket =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).
build();
public void updatePermissions() {

s3Bucket.setPublicAccessBlock(new SetPublicAccessBlockRequest()
                        .withBucketName(this.BUCKET_NAME)
                        .withPublicAccessBlockConfiguration(new
                        PublicAccessBlockConfiguration()
                        .withBlockPublicAcls(true)
                        .withIgnorePublicAcls(true)
                        .withBlockPublicPolicy(true)
                        .withRestrictPublicBuckets(true)));
        System.out.format("Permissions updated!");
}
```

7. ACL Update

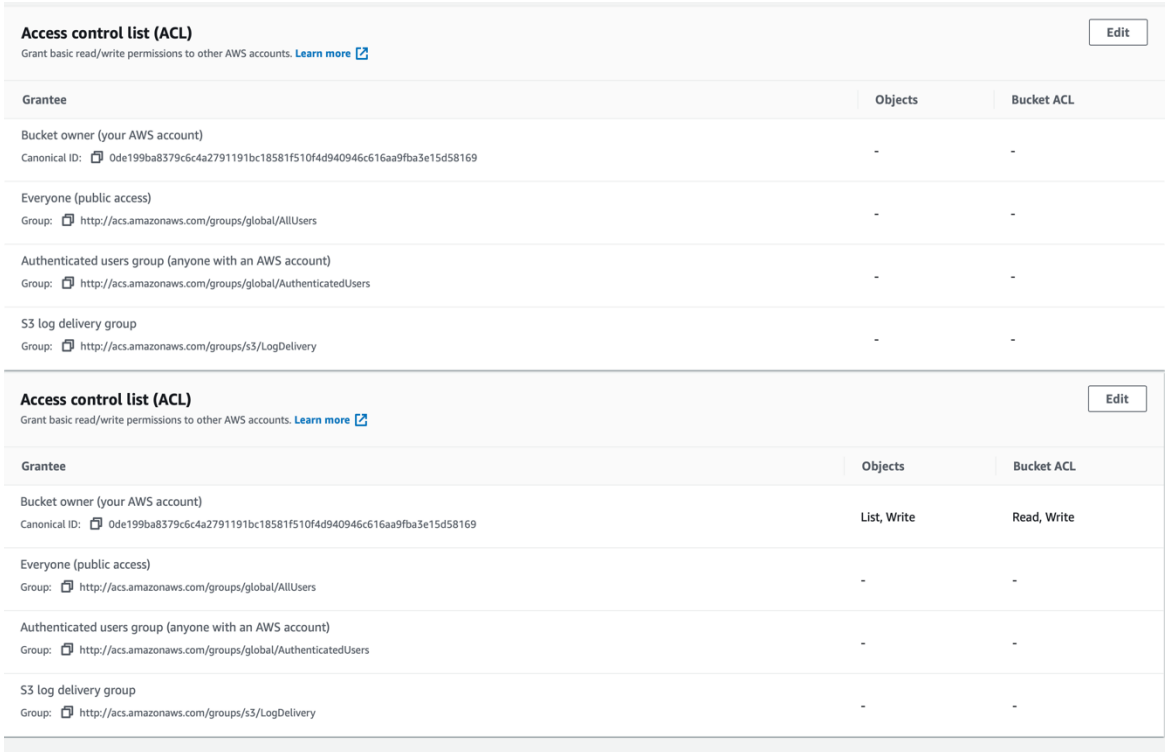

Figure 7: ACL Updated using java for full control

- Code [5][6]:

```
final static String BUCKET_NAME = "csci5410-a1-bucket2";
final AmazonS3 s3Bucket =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).
build();

public void setFullControlToUserUsingACL() {

      final AccessControlList acl =
s3Bucket.getBucketAcl(BUCKET_NAME);

        acl.grantAllPermissions(new Grant(new
CanonicalGrantee(acl.getOwner().getId()),
Permission.FullControl));
        Grant grant1 = new Grant(new
CanonicalGrantee(s3Bucket.getS3AccountOwner().getId()),
Permission.FullControl);

        AccessControlList newBucketAcl =
s3Bucket.getBucketAcl(BUCKET_NAME);
        newBucketAcl.grantAllPermissions(grant1);
        s3Bucket.setBucketAcl(BUCKET_NAME, newBucketAcl);
        System.out.format("ACL updated!");
}
```

8. Moving the bucket

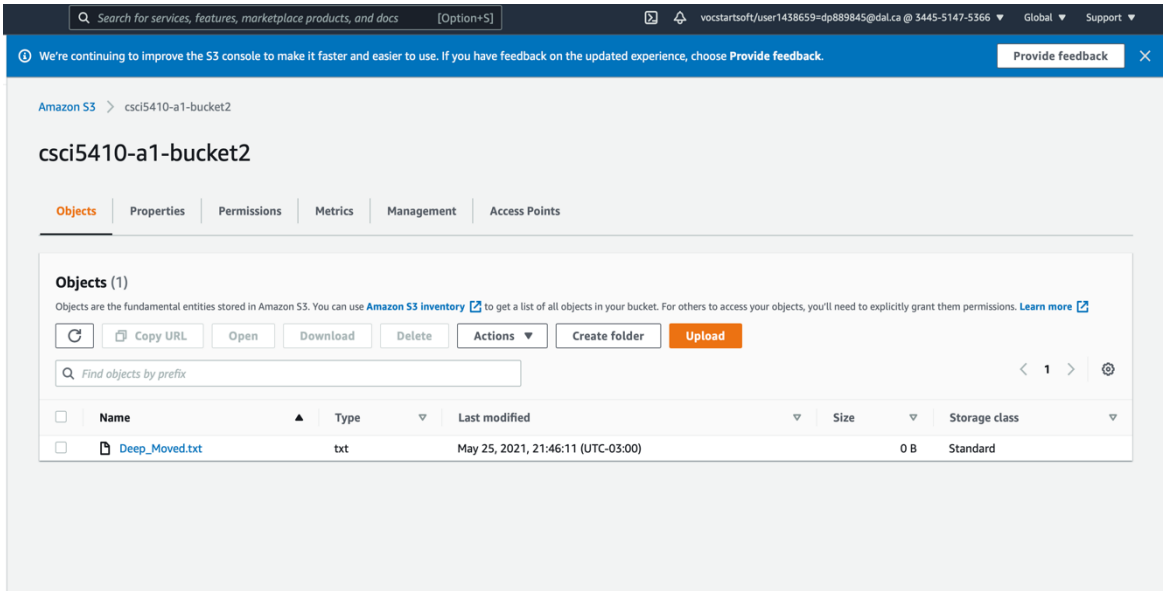

Figure 8: File Moved into second Bucket and removed from first bucket

- Code [7][8]:

```
private final static String BUCKET_1 = "csci5410-a1-bucket";
private final static String BUCKET_1_FILE = "Deep.txt";

private final static String BUCKET_2 = "csci5410-a1-bucket2";
private final static String BUCKET_2_FILE = "Deep_Moved.txt";

public void moveFilesBetweenBuckets() {
      try {
            final AmazonS3 s3 =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).
build();

            //Copy file from one bucket to another
            s3.copyObject(BUCKET_1, BUCKET_1_FILE, BUCKET_2,
            BUCKET_2_FILE);
            System.out.println("Copy Process Complete!");


            //Delete file after completion of copy process
          s3.deleteObject(BUCKET_1, BUCKET_1_FILE);
            System.out.println("Deletion Process Complete!");

      } catch (Exception e) {
            e.printStackTrace();
      }
      System.out.println("Moving Complete!");
}
```

Task-3

S3 Bucket Screen Shots

1. Lookup Table on S3 Bucket

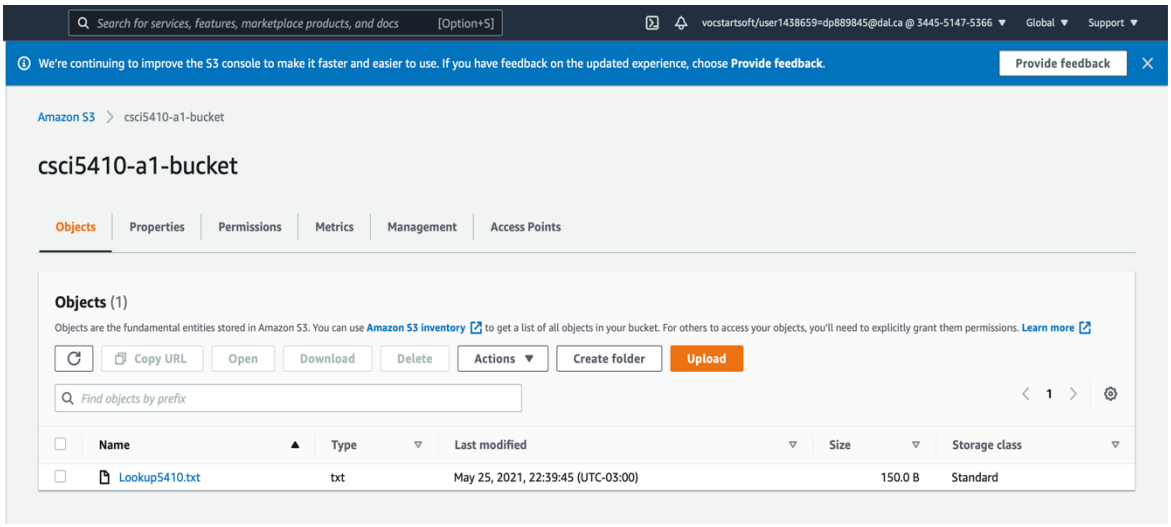

Figure 9: Lookup.txt on S3 Bucket

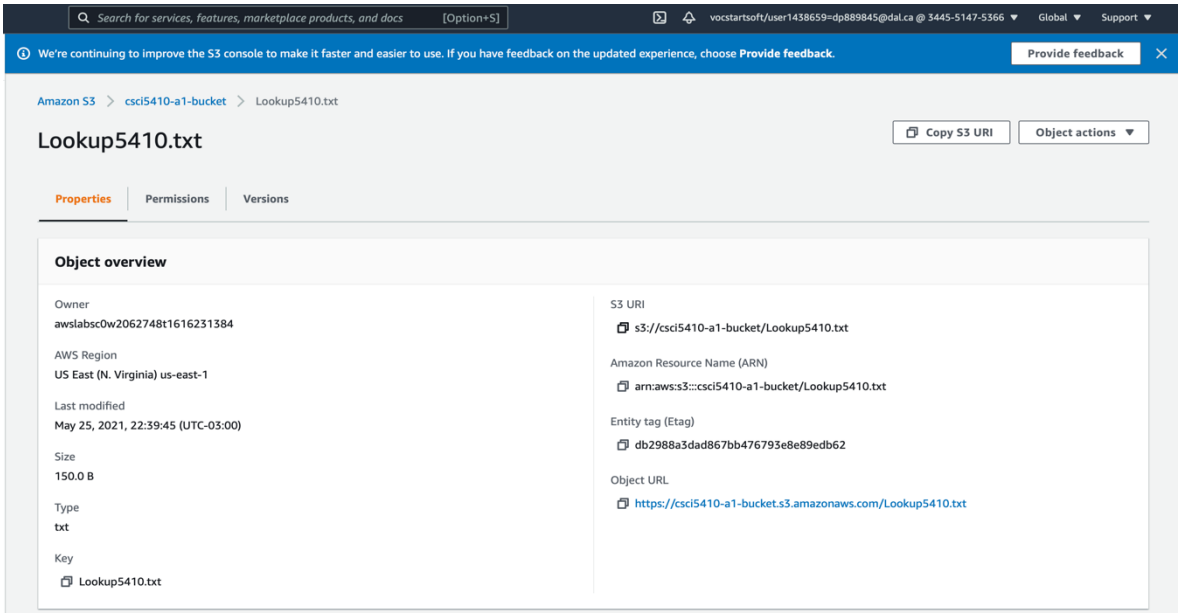

Figure 10: Lookuptable.txt Information on S3 Bucket

Screen Shots of RDBMS

1. Connection Configuration for RDS Database:

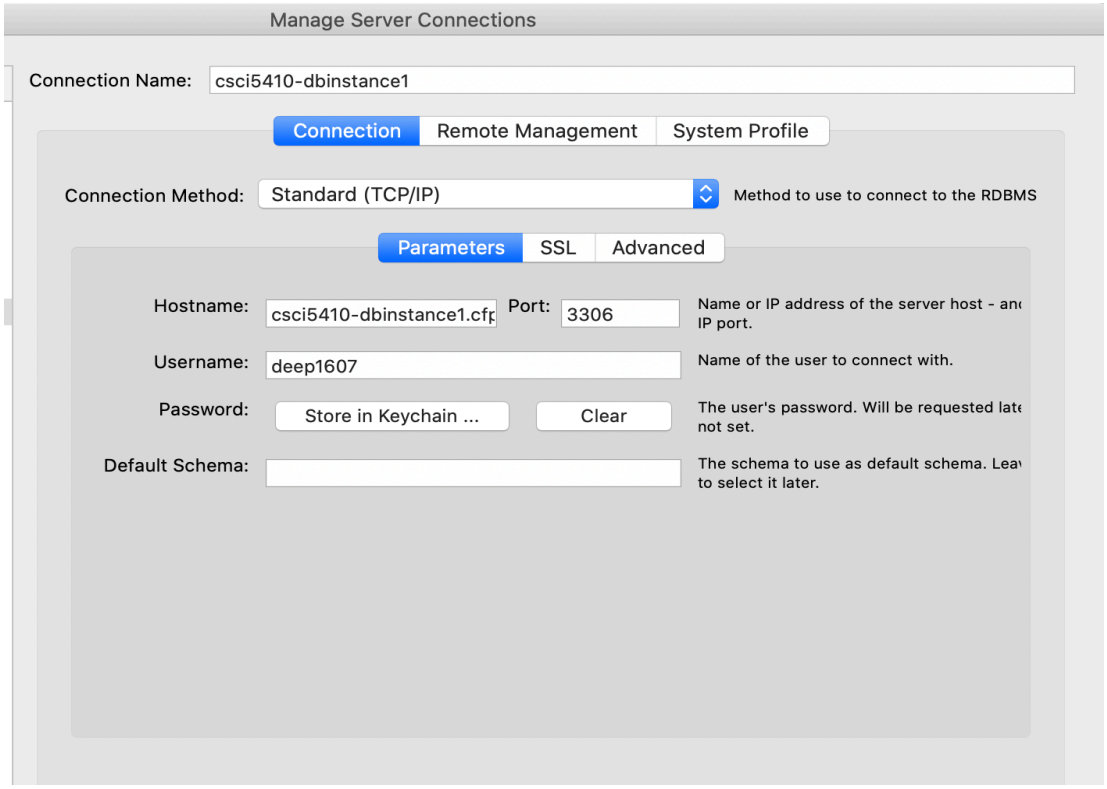

Figure 11: RDS Connection Configuration

2. Database containing table users:

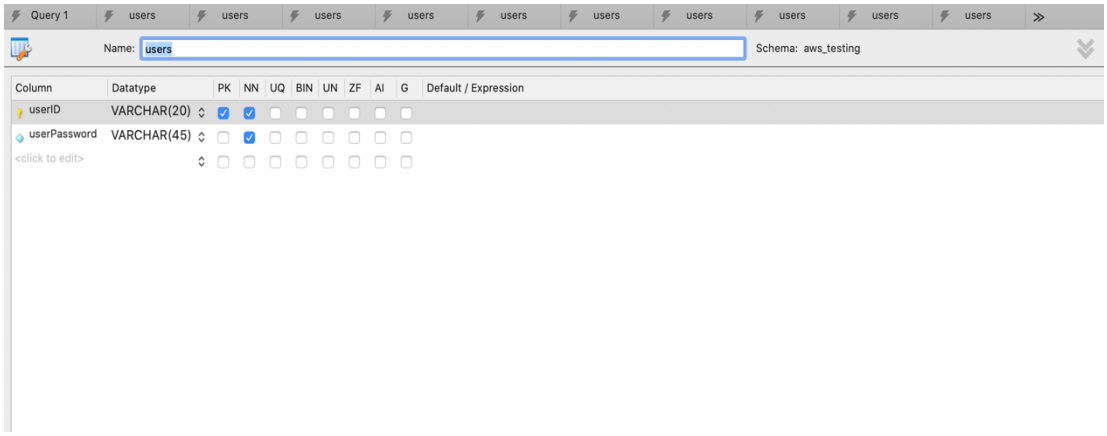

Figure 12:Users table with fields userID and userPassword

3. Encrypted Data Inside Users Table:

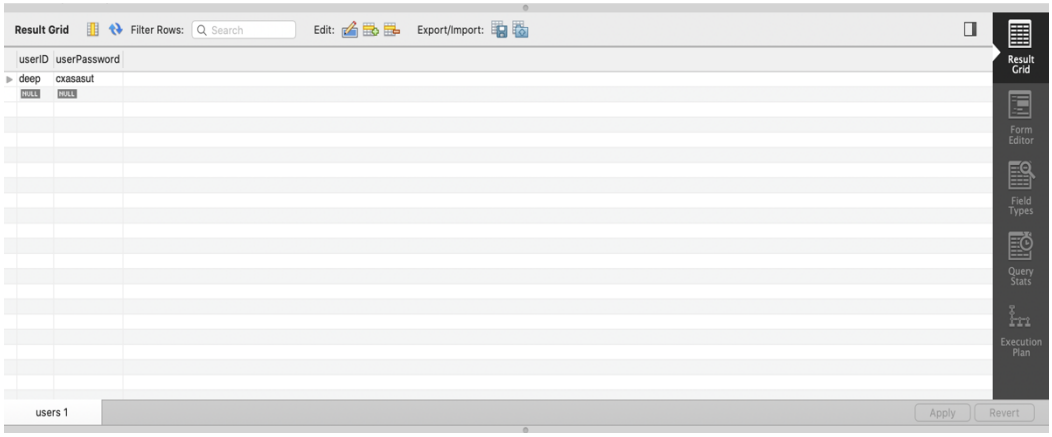

Figure 13: Encrypted Data inserted into DB

4. Decrypted Password from Java output:

```
Connection Successful
Enter userId to get password:
deep
Decrypted Password is: deep
```

Figure 14: Password Decrypted

Code

1. **Code To upload Lookup File [2]:**

```
import java.io.File;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

public class UploadFile {

public void uploadFileToS3Bucket(String filePath) {


      final AmazonS3 s3 =
AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).build(
);

      String bucketName = "csci5410-a1-bucket";
      String fileName = "Lookup5410.txt";

      s3.putObject(bucketName,fileName,new File(filePath));

      System.out.println("Updated Successfully!!");
}
}
```

2. **Code to Add User:**

```
import java.sql.Connection;
import java.sql.PreparedStatement;

public class AddUser {

private String userName = "deep";
private String password = "Deep";

public void addUserIntoDatabase() {

      DBConnection objConnect = new DBConnection();
      Connection conn = objConnect.connectToInstance();

      String userName = "deep";
      String password = "Deep";

      EncryptData objEncrypt = new EncryptData();
      objEncrypt.createLookupTable();

      try {
            // the mysql insert statement
            String sql1 = " insert into users (userID, userPassword)"
+ " values (?, ?)";

            // create the mysql insert preparedstatement
            PreparedStatement preparedStmt =
conn.prepareStatement(sql1);
            preparedStmt.setString (1, userName);
```

```
            preparedStmt.setString (2,
objEncrypt.encryptPassword(password));

            // execute the preparedstatement
            boolean isInserted = preparedStmt.execute();


            if(isInserted) {
                  System.out.println("Data Inserted!!");
            }
      }catch(Exception e) {
            e.printStackTrace();
      }
      finally {
            try {
                  conn.close();
            }catch(Exception e) {
                  e.printStackTrace();
            }
      }
}
}
```

3. **Code for Database Connection:**

```
import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnection {
private static String DRIVER_INFO = "com.mysql.cj.jdbc.Driver";
private static String DATABASE_URL = "jdbc:mysql://csci5410-
dbinstance1.cfprz1ccsmta.ap-south-
1.rds.amazonaws.com:3306/aws_testing?user=deep1607&password=Dee16798p
*";
private static String DATABASE_USERNAME = "deep1607";
private static String DATABASE_PASSWORD = "Dee16798p*";

public Connection connectToInstance() {

      Connection conn = null;
      try {
            Class.forName(DRIVER_INFO);
            // conn =
            //
DriverManager.getConnection("jdbc:mysql://db.cs.dal.ca:3306/csci3901?
serverTimezone=UTC",
            // "dppatel","B00865413");


            conn = DriverManager.getConnection(DATABASE_URL);

            System.out.println("Connection Successful");

      } catch (Exception e) {
            System.out.println("ERROR: " + e.getMessage());
      }
      return conn;
}
}
```

4. **Encryption Code by generating lookup map:**

```
public static final Map<String,String> MAP_LOOKUP_TABLE = new
HashMap<String,String>();

public void createLookupTable() {

	BufferedReader reader;
	try {

		AmazonS3 s3Client = new AmazonS3Client(new
ProfileCredentialsProvider());
		S3Object object = s3Client.getObject(new
GetObjectRequest("csci5410-a1-bucket", "Lookup5410.txt"));
		InputStream objectData = object.getObjectContent();



		reader = new BufferedReader(new
InputStreamReader(objectData));
		String line = reader.readLine();

		// Process the objectData stream.

		int counter = 1;
		while (line != null) {
			if(counter != 1) {
				String[] arrLookupTable = line.split("    ");

				MAP_LOOKUP_TABLE.put(arrLookupTable[0],
arrLookupTable[1]);
			}

			// read next line
			line = reader.readLine();
			counter++;
		}

		objectData.close();

		reader.close();

	}catch(Exception e) {
		e.printStackTrace();
	}
}

public String encryptPassword(String password) {
	String encryptedPassword = "";

	password = password.toLowerCase();

	for(int i = 0; i < password.length(); i++ ) {

		String tempReplaceValue =
MAP_LOOKUP_TABLE.get(String.valueOf(password.charAt(i)));
		encryptedPassword += tempReplaceValue;
	}
	return encryptedPassword;
}
```

5. **Decryption Code By generating reverse lookup map:**

```
public void createDecryptLookupTable() {

      BufferedReader reader;
      try {

            AmazonS3 s3Client = new AmazonS3Client(new
ProfileCredentialsProvider());
            S3Object object = s3Client.getObject(new
GetObjectRequest("csci5410-a1-"
                        + "", "Lookup5410.txt"));
            InputStream objectData = object.getObjectContent();

            reader = new BufferedReader(new
InputStreamReader(objectData));
            String line = reader.readLine();

            // Process the objectData stream.

            int counter = 1;
            while (line != null) {
                  if(counter != 1) {
                        String[] arrLookupTable = line.split("    ");


MAP_DECRYPT_LOOKUP_TABLE.put(arrLookupTable[1], arrLookupTable[0]);
                  }

                  // read next line
                  line = reader.readLine();
                  counter++;
            }

            objectData.close();

            reader.close();

      }catch(Exception e) {
            e.printStackTrace();
      }
}

public String decryptPassword(String password) {

    String[] arrSplittedPassword = password.split("(?<=\\G..)");

    String decryptedPassword = "";

    for(int i = 0; i< arrSplittedPassword.length; i++) {
      decryptedPassword +=
MAP_DECRYPT_LOOKUP_TABLE.get(arrSplittedPassword[i]);
    }

    return decryptedPassword;
}
```

6. **Find password from UserID Code in Java:**

```
public void getPasswordFromID() {
	DBConnection objConnect = new DBConnection();
	Connection conn = objConnect.connectToInstance();
	Scanner sc = new Scanner(System.in);

	System.out.println("Enter userId to get password: ");
	String idToFind = sc.nextLine();
	String foundPassword = "";

	DecryptData objDecryptData = new DecryptData();
	objDecryptData.createDecryptLookupTable();


	try {
		Statement stmt1 = conn.createStatement();
		ResultSet rs = stmt1.executeQuery("SELECT * FROM
aws_testing.users");


		while (rs.next())
		{
		  String tempIdToMatch = rs.getString("userID");

		  if(idToFind.equals(tempIdToMatch)) {
		      foundPassword = rs.getString("userPassword");
		  }
		}
	}catch(Exception e) {
		e.printStackTrace();
	}

	String decryptedPassword =
objDecryptData.decryptPassword(foundPassword);

	System.out.println(decryptedPassword);
}
```

References

[1]. "Evaluation of container orchestration systems for deploying and managing NoSQL database clusters" by Eddy Truyen, Dimitri Van Landuyt, Bert Lagaisse, Wouter Joosen, Matt Bruzek

[2]. https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-s3-objects.html#upload-object

[3]. https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-s3-buckets.html#create-bucket

[4]. https://docs.aws.amazon.com/AmazonS3/latest/userguide/configuring-block-public-access-bucket.html

[5]. https://docs.aws.amazon.com/AmazonS3/latest/userguide/managing-acls.html

[6]. https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/example_code/s3/src/main/java/com/example/s3/SetAcl.java

[7]. https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/java/example_code/s3/src/main/java/aws/example/s3/CopyObject.java

[8]. https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/java/example_code/s3/src/main/java/aws/example/s3/GetObject.java