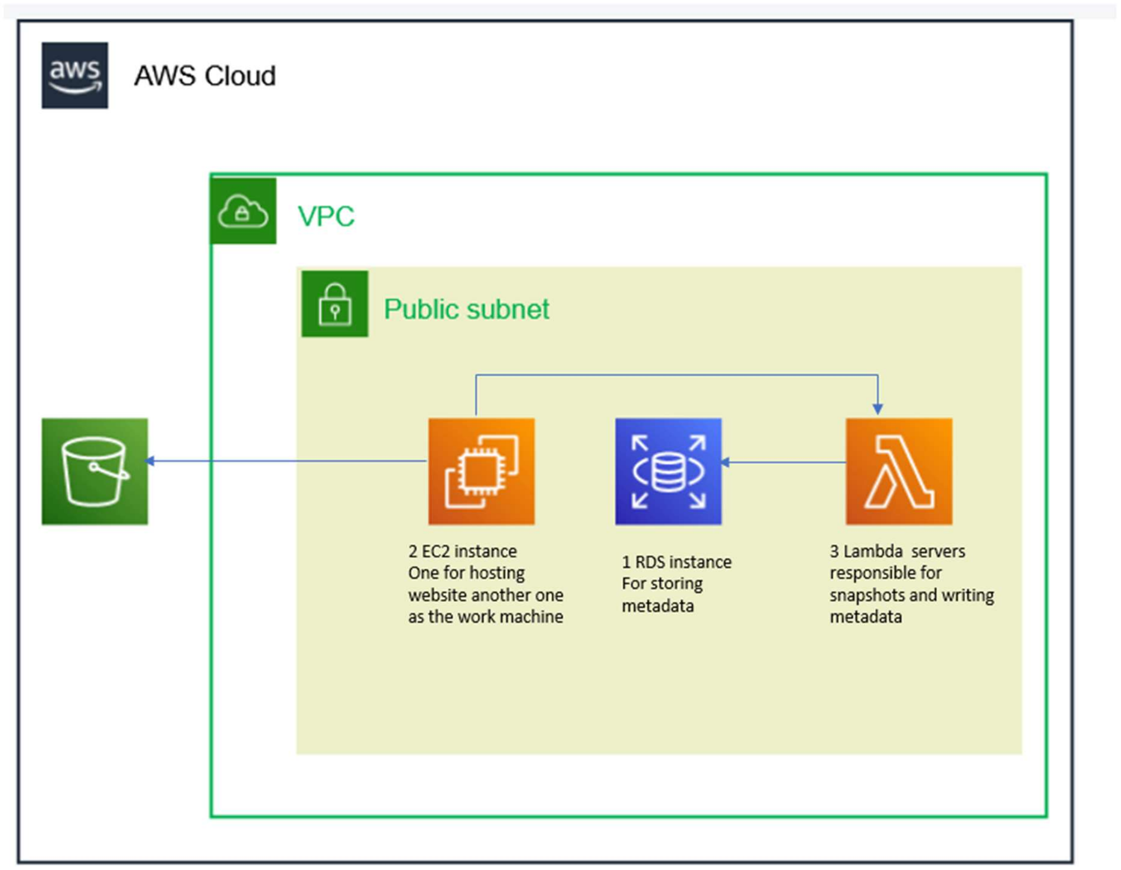
Automated Backup System Official documentation

Introduction: -

Our goal is to automate the backup of Instances in your respective AWS account and to provide you with a portal to access the required metadata for all the backed-up instances. This is solely done through lambda functions which trigger automatically whenever desired to be scheduled. We also let users upload specific files to the s3 bucket and the links to access and instructions to do the same are provided in the same portal.

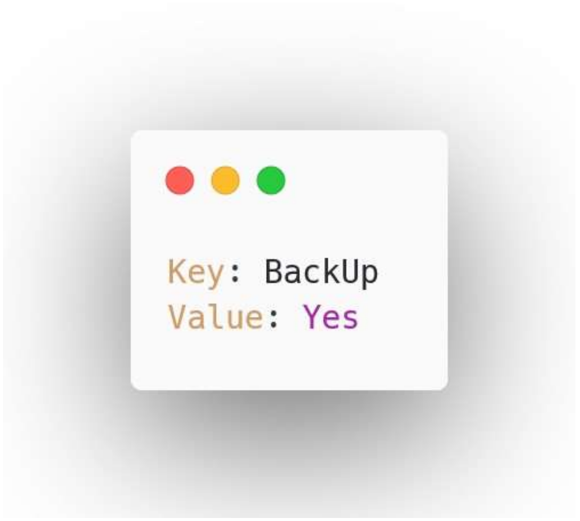
Project Architecture: -



Project Implementation and guidelines: -

Any instance can be automatically backed up by adding the following. This can be done by generally by clicking on the checkbox of instance scroll down and choose tags edit the tags.

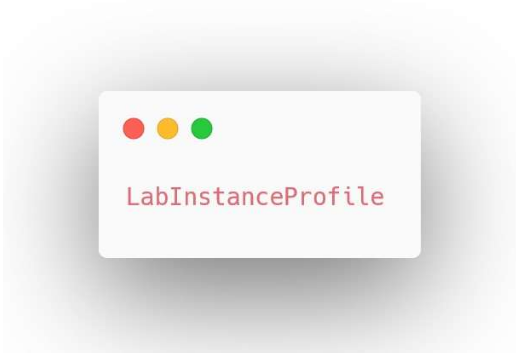
tag: -



```
Key: BackUp
Value: Yes
```

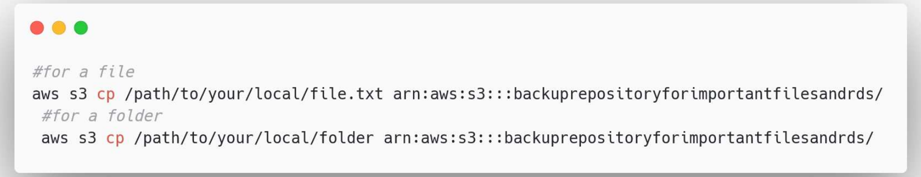
Whenever this tag is added to any instance our lambda functions(demo function and rdsbackupthingy) will scan for this tag over us-east-1 region and if it identifies a resource contains this tag it will automatically trigger a snapshot which can be later found under ec2>snapshots or rds>snapshots .

For Backing up a particular file ensure the instance is attached with the IAM role given below



```
LabInstanceProfile
```

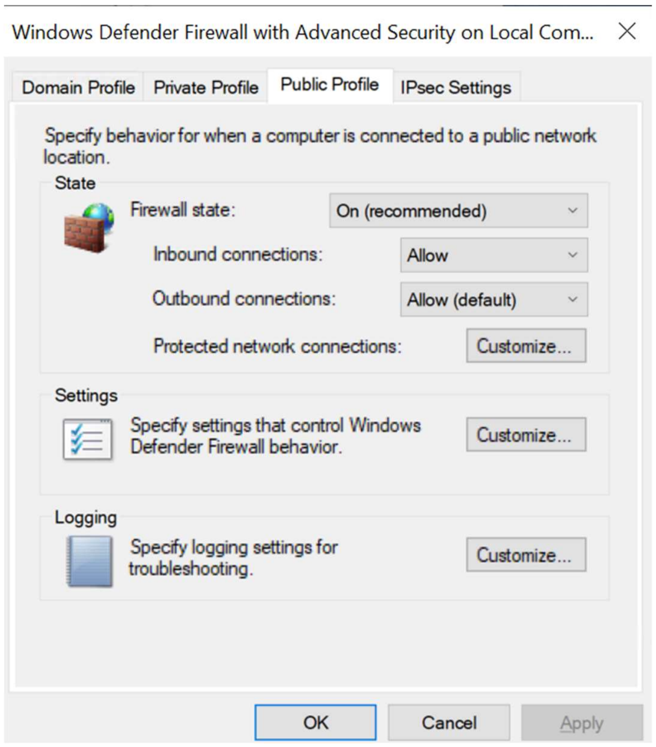
To copy a file from a machine to aws use the following command



```
#for a file
aws s3 cp /path/to/your/local/file.txt arn:aws:s3:::backuprepositoryforimportantfilesandrds/
 #for a folder
 aws s3 cp /path/to/your/local/folder arn:aws:s3:::backuprepositoryforimportantfilesandrds/
```

<u>Website Portal Implementation: -</u>

- To implement the website firstly launch an ec2 Instance of windows server 2022 AMI create a ssh file (PEM file) so that it can be used later on for logging in. (ensure security group allows ssh public access, http public access, https public access)
- Next connect to the instance using rdp . Upload the ssh file to decrypt the password download and run the rdp file.
- After logging in search for windows security > firewall > scroll down to advanced settings > in the control panel click on windows defender firewall properties > select public profile on the menu > select inbound rules click on allow all.
  If done correctly it should look as follows


Windows Defender Firewall with Advanced Security on Local Com...

Specify behavior for when a computer is connected to a public network location.

State

Firewall state: On (recommended)

Inbound connections: Allow

Outbound connections: Allow (default)

Protected network connections: Customize...

Settings

Specify settings that control Windows Defender Firewall behavior. Customize...

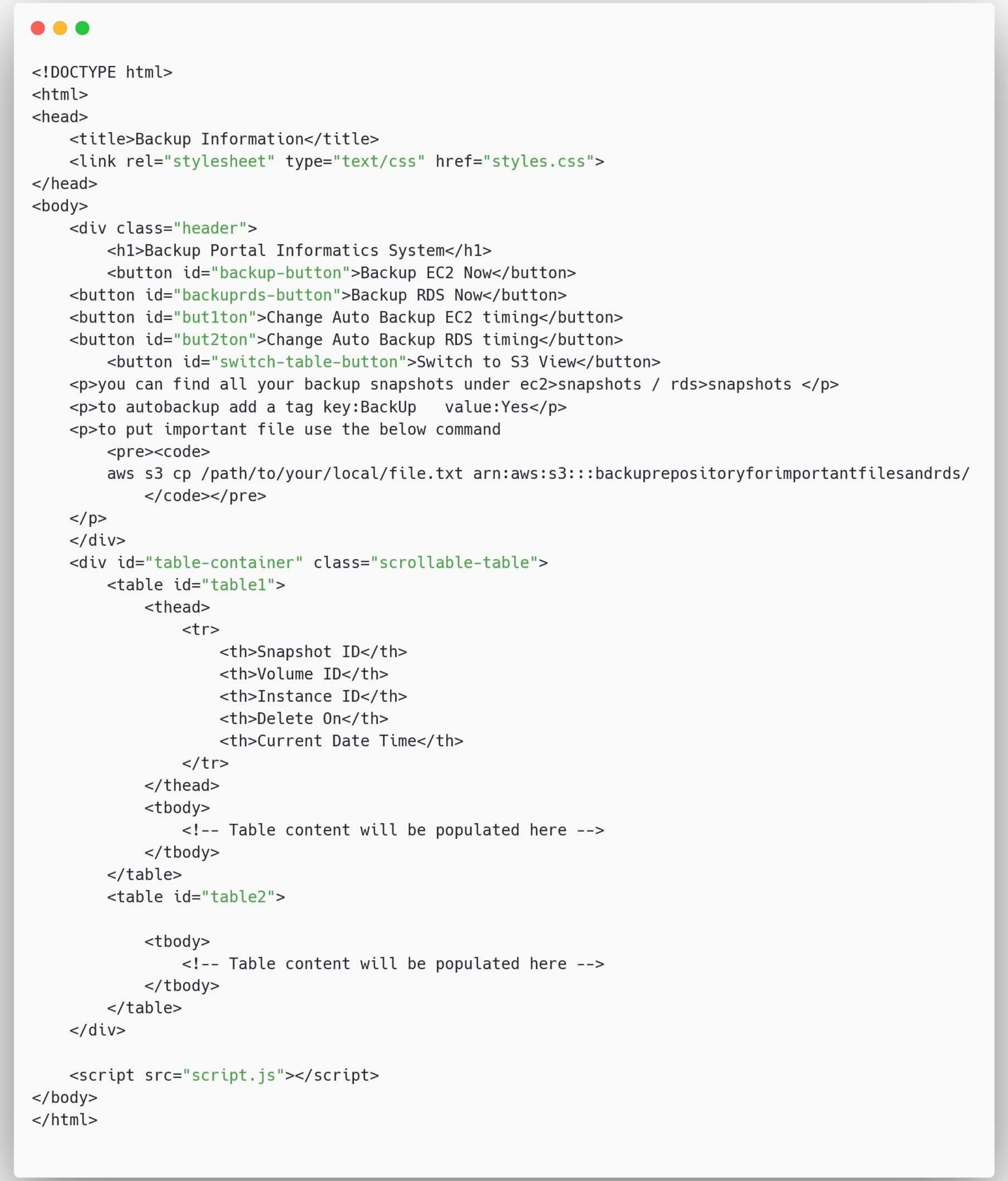
Logging

Specify logging settings for troubleshooting. Customize...

- Next install xampp server from the browser and after installing xampp (just follow regular installation click next until finishing installation).
- After installing xampp navigate to file explorer go to c drive and navigate to xampp folder. click on xampp folder and rename the file to htdocs rename the dashboard file to 1 and create a new file called dashboard upload the code snippets given below to the same
- Now start the xampp server (open xampp next to Apache server click on start button)

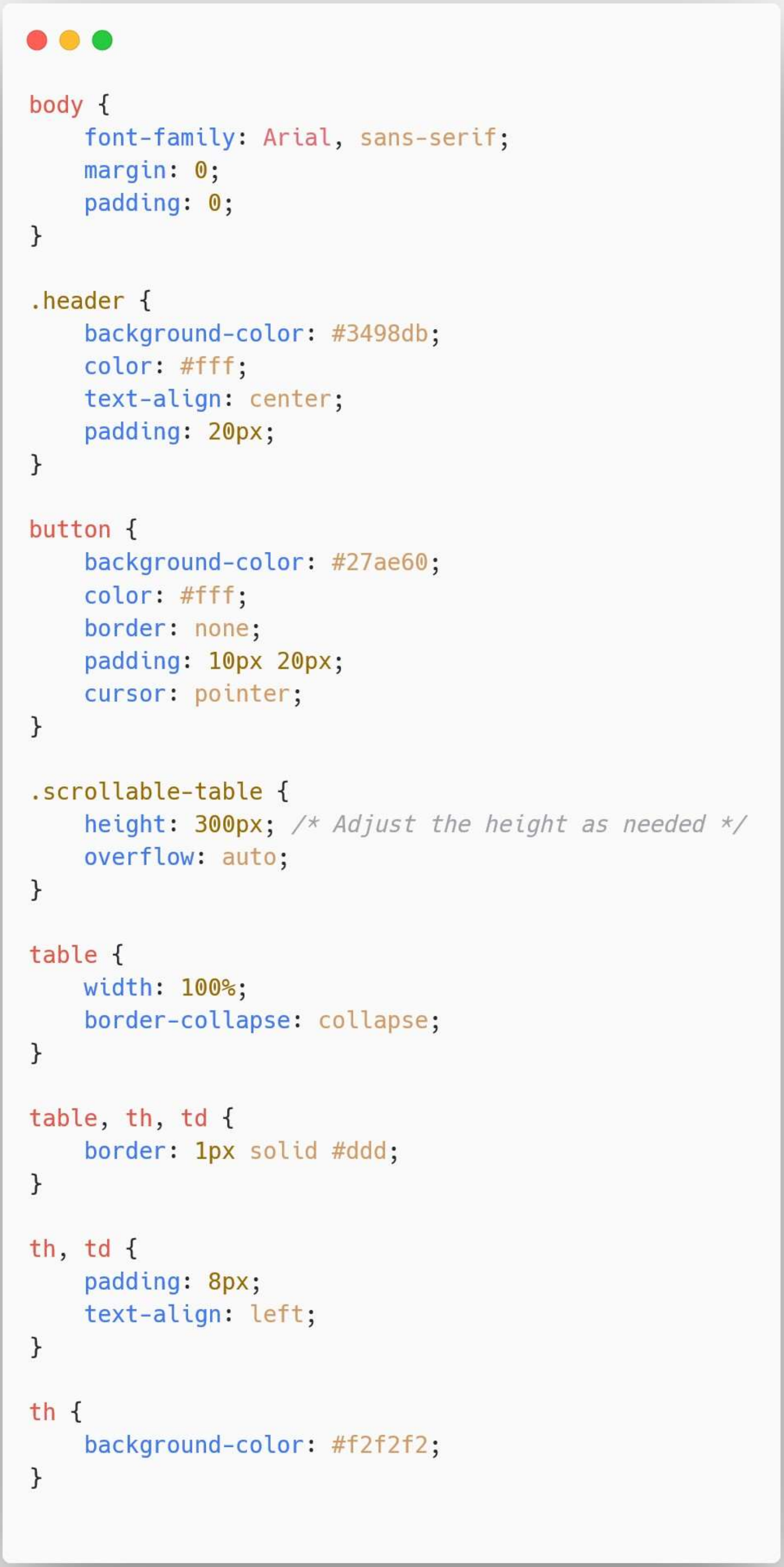
Code files are given below: -

index.html



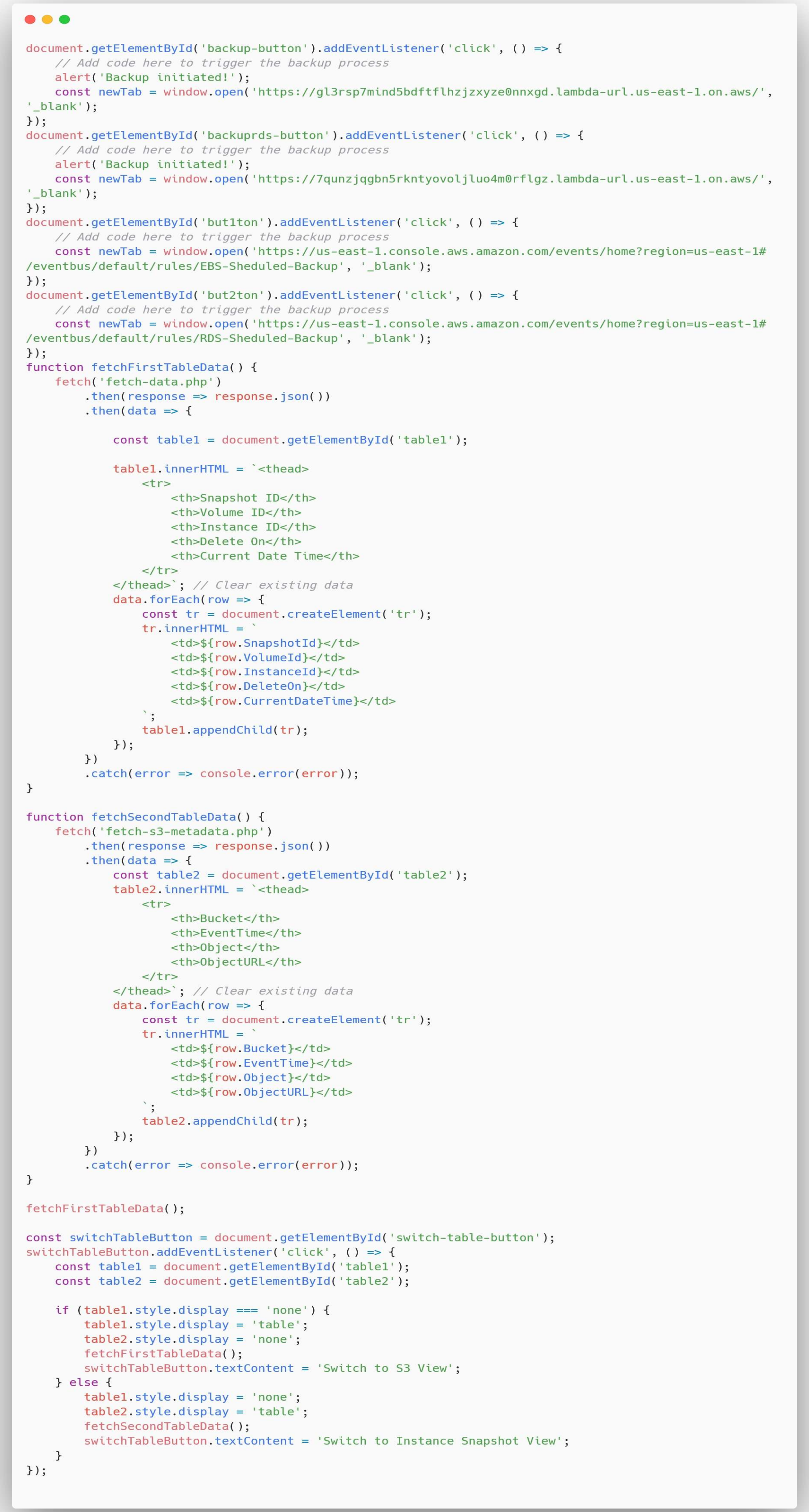
```html
<!DOCTYPE html>
<html>
<head>
    <title>Backup Information</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <div class="header">
        <h1>Backup Portal Informatics System</h1>
        <button id="backup-button">Backup EC2 Now</button>
    <button id="backuprds-button">Backup RDS Now</button>
    <button id="but1ton">Change Auto Backup EC2 timing</button>
    <button id="but2ton">Change Auto Backup RDS timing</button>
        <button id="switch-table-button">Switch to S3 View</button>
    <p>you can find all your backup snapshots under ec2>snapshots / rds>snapshots </p>
    <p>to autobackup add a tag key:BackUp   value:Yes</p>
    <p>to put important file use the below command
        <pre><code>
        aws s3 cp /path/to/your/local/file.txt arn:aws:s3:::backuprepositoryforimportantfilesandrds/
            </code></pre>
    </p>
    </div>
    <div id="table-container" class="scrollable-table">
        <table id="table1">
            <thead>
                <tr>
                    <th>Snapshot ID</th>
                    <th>Volume ID</th>
                    <th>Instance ID</th>
                    <th>Delete On</th>
                    <th>Current Date Time</th>
                </tr>
            </thead>
            <tbody>
                <!-- Table content will be populated here -->
            </tbody>
        </table>
        <table id="table2">

            <tbody>
                <!-- Table content will be populated here -->
            </tbody>
        </table>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

styles.css



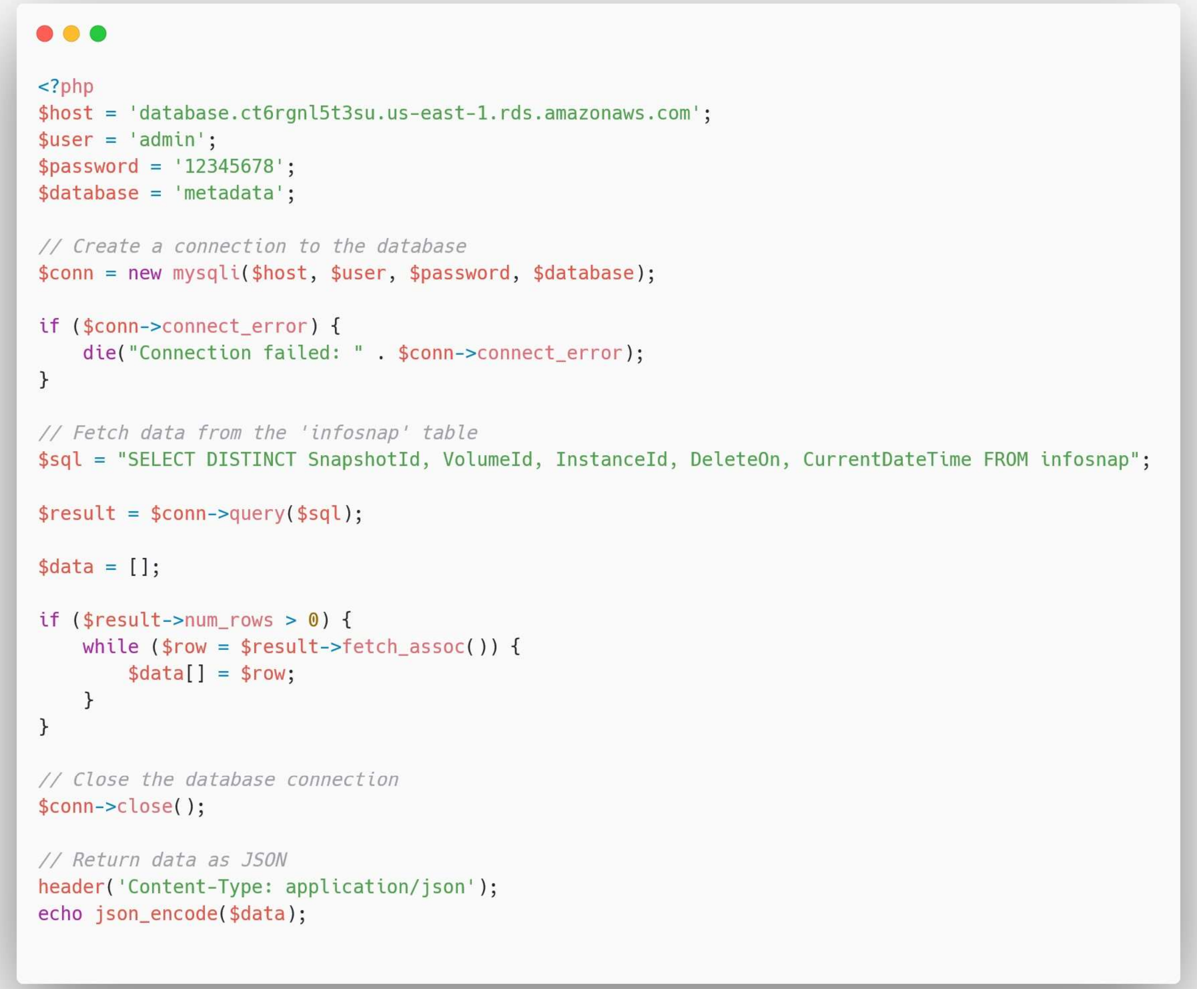
```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

.header {
    background-color: #3498db;
    color: #fff;
    text-align: center;
    padding: 20px;
}

button {
    background-color: #27ae60;
    color: #fff;
    border: none;
    padding: 10px 20px;
    cursor: pointer;
}

.scrollable-table {
    height: 300px; /* Adjust the height as needed */
    overflow: auto;
}

table {
    width: 100%;
    border-collapse: collapse;
}

table, th, td {
    border: 1px solid #ddd;
}

th, td {
    padding: 8px;
    text-align: left;
}

th {
    background-color: #f2f2f2;
}
```

script.js*(some links might vary will be discussed later in the docs)



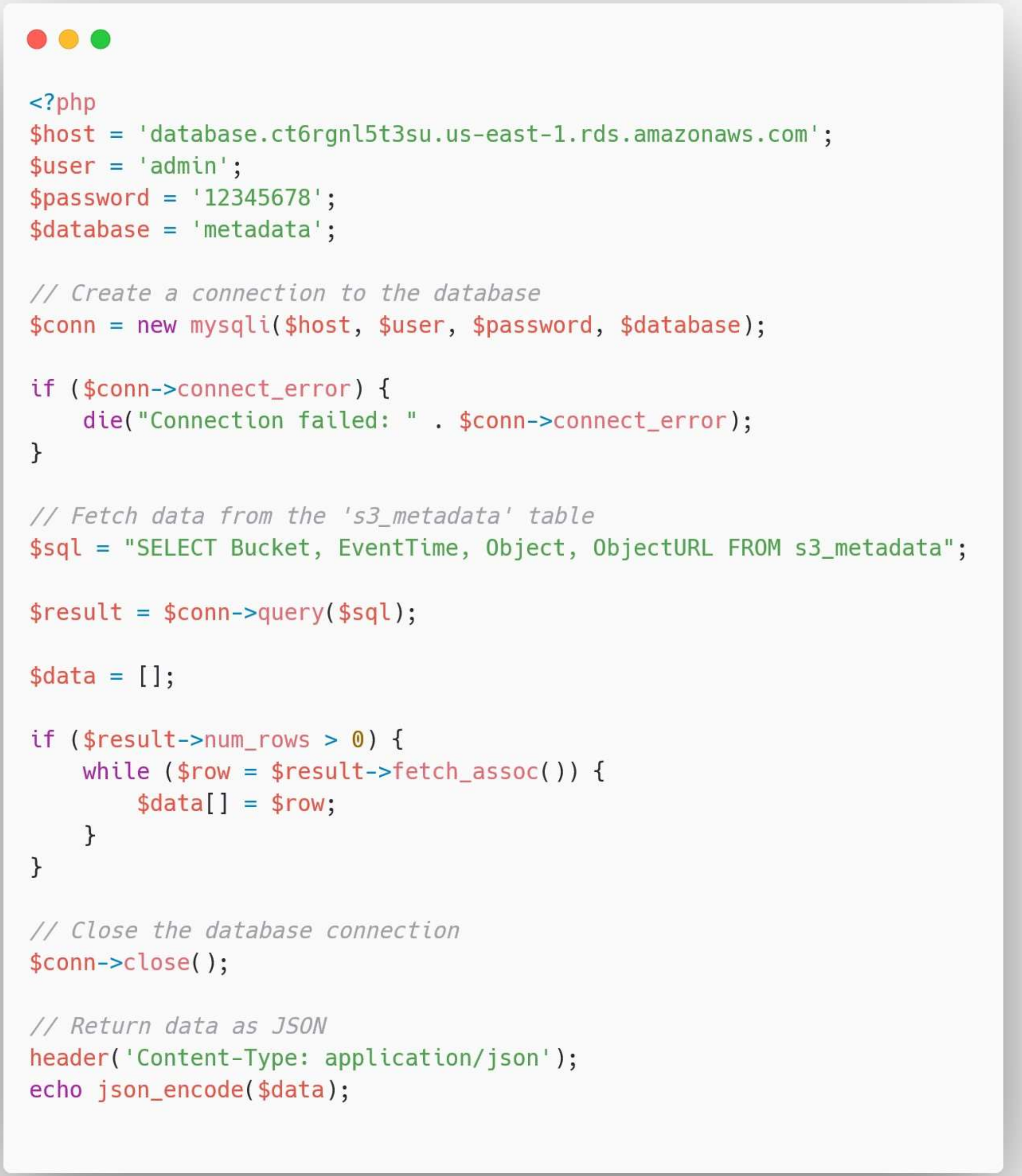
```javascript
document.getElementById('backup-button').addEventListener('click', () => {
    // Add code here to trigger the backup process
    alert('Backup initiated!');
    const newTab = window.open('https://gl3rsp7mind5bdftflhzjzxyze0nnxgd.lambda-url.us-east-1.on.aws/',
'_blank');
});
document.getElementById('backuprds-button').addEventListener('click', () => {
    // Add code here to trigger the backup process
    alert('Backup initiated!');
    const newTab = window.open('https://7qunzjqgbn5rkntyovoljluo4m0rflgz.lambda-url.us-east-1.on.aws/',
'_blank');
});
document.getElementById('but1ton').addEventListener('click', () => {
    // Add code here to trigger the backup process
    const newTab = window.open('https://us-east-1.console.aws.amazon.com/events/home?region=us-east-1#
/eventbus/default/rules/EBS-Sheduled-Backup', '_blank');
});
document.getElementById('but2ton').addEventListener('click', () => {
    // Add code here to trigger the backup process
    const newTab = window.open('https://us-east-1.console.aws.amazon.com/events/home?region=us-east-1#
/eventbus/default/rules/RDS-Sheduled-Backup', '_blank');
});
function fetchFirstTableData() {
    fetch('fetch-data.php')
        .then(response => response.json())
        .then(data => {

            const table1 = document.getElementById('table1');

            table1.innerHTML = `<thead>
                <tr>
                    <th>Snapshot ID</th>
                    <th>Volume ID</th>
                    <th>Instance ID</th>
                    <th>Delete On</th>
                    <th>Current Date Time</th>
                </tr>
            </thead>`; // Clear existing data
            data.forEach(row => {
                const tr = document.createElement('tr');
                tr.innerHTML = `
                    <td>${row.SnapshotId}</td>
                    <td>${row.VolumeId}</td>
                    <td>${row.InstanceId}</td>
                    <td>${row.DeleteOn}</td>
                    <td>${row.CurrentDateTime}</td>
                `;
                table1.appendChild(tr);
            });
        })
        .catch(error => console.error(error));
}

function fetchSecondTableData() {
    fetch('fetch-s3-metadata.php')
        .then(response => response.json())
        .then(data => {
            const table2 = document.getElementById('table2');
            table2.innerHTML = `<thead>
                <tr>
                    <th>Bucket</th>
                    <th>EventTime</th>
                    <th>Object</th>
                    <th>ObjectURL</th>
                </tr>
            </thead>`; // Clear existing data
            data.forEach(row => {
                const tr = document.createElement('tr');
                tr.innerHTML = `
                    <td>${row.Bucket}</td>
                    <td>${row.EventTime}</td>
                    <td>${row.Object}</td>
                    <td>${row.ObjectURL}</td>
                `;
                table2.appendChild(tr);
            });
        })
        .catch(error => console.error(error));
}

fetchFirstTableData();

const switchTableButton = document.getElementById('switch-table-button');
switchTableButton.addEventListener('click', () => {
    const table1 = document.getElementById('table1');
    const table2 = document.getElementById('table2');

    if (table1.style.display === 'none') {
        table1.style.display = 'table';
        table2.style.display = 'none';
        fetchFirstTableData();
        switchTableButton.textContent = 'Switch to S3 View';
    } else {
        table1.style.display = 'none';
        table2.style.display = 'table';
        fetchSecondTableData();
        switchTableButton.textContent = 'Switch to Instance Snapshot View';
    }
});
```

fetch-data.php*(check in db_host endpoints and password username)



```php
<?php
$host = 'database.ct6rgnl5t3su.us-east-1.rds.amazonaws.com';
$user = 'admin';
$password = '12345678';
$database = 'metadata';

// Create a connection to the database
$conn = new mysqli($host, $user, $password, $database);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Fetch data from the 'infosnap' table
$sql = "SELECT DISTINCT SnapshotId, VolumeId, InstanceId, DeleteOn, CurrentDateTime FROM infosnap";

$result = $conn->query($sql);

$data = [];

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $data[] = $row;
    }
}

// Close the database connection
$conn->close();

// Return data as JSON
header('Content-Type: application/json');
echo json_encode($data);
```

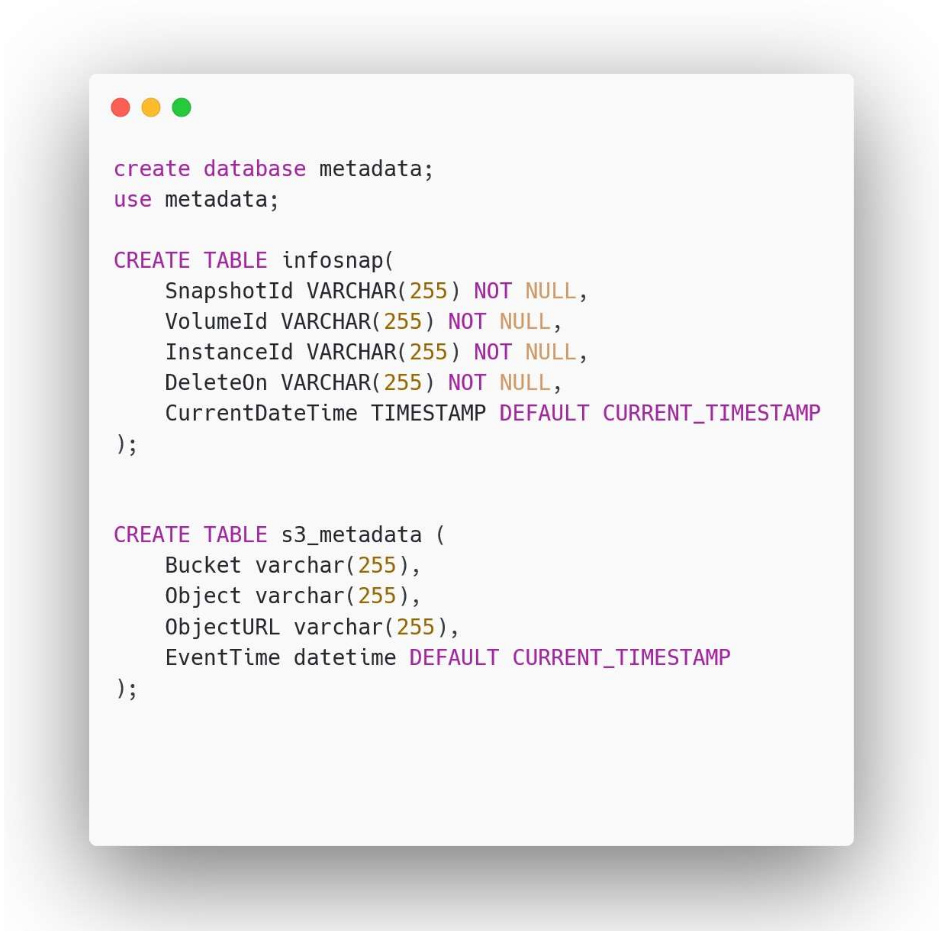
fetch-s3-metadata.php*(check in db_host endpoints and password username)


```php
<?php
$host = 'database.ct6rgnl5t3su.us-east-1.rds.amazonaws.com';
$user = 'admin';
$password = '12345678';
$database = 'metadata';

// Create a connection to the database
$conn = new mysqli($host, $user, $password, $database);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Fetch data from the 's3_metadata' table
$sql = "SELECT Bucket, EventTime, Object, ObjectURL FROM s3_metadata";

$result = $conn->query($sql);

$data = [];

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $data[] = $row;
    }
}

// Close the database connection
$conn->close();

// Return data as JSON
header('Content-Type: application/json');
echo json_encode($data);
```

Database Implementation :-

- Navigate to RDS and select on create database
- Choose Engine as MySQL follow standard create
- Click on free tier
- Choose username and password
- Scroll down to security groups. enable public access so that database can be accessed from anywhere . pickup a default security group and modify that security group later on ensure that the database can be connected from anyipv4 address .
- Click on create the database .

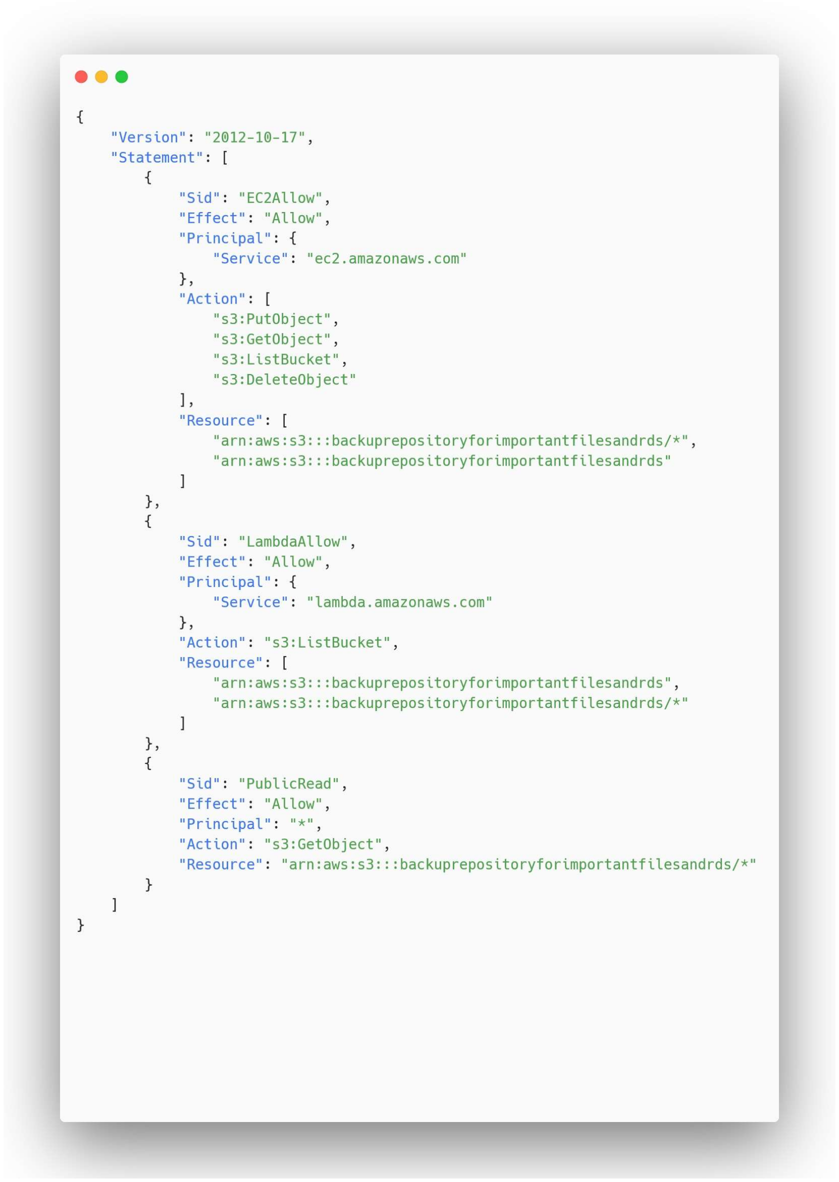
After creating the database connect to the database using mysql workbench or any other platform to do the same and follow the following commands below



```sql
create database metadata;
use metadata;

CREATE TABLE infosnap(
    SnapshotId VARCHAR(255) NOT NULL,
    VolumeId VARCHAR(255) NOT NULL,
    InstanceId VARCHAR(255) NOT NULL,
    DeleteOn VARCHAR(255) NOT NULL,
    CurrentDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE s3_metadata (
    Bucket varchar(255),
    Object varchar(255),
    ObjectURL varchar(255),
    EventTime datetime DEFAULT CURRENT_TIMESTAMP
);
```

S3 Bucket Implementation :-

- Go to aws services choose s3
- Choose create bucket and name the bucket backuprepositoryforimportantfilesandrds and uncheck block public access accept the acknowledgement .
- Click on create bucket now click on the bucket and navigate to permissions
- Click on bucket policies and select edit enter the json file below and click save



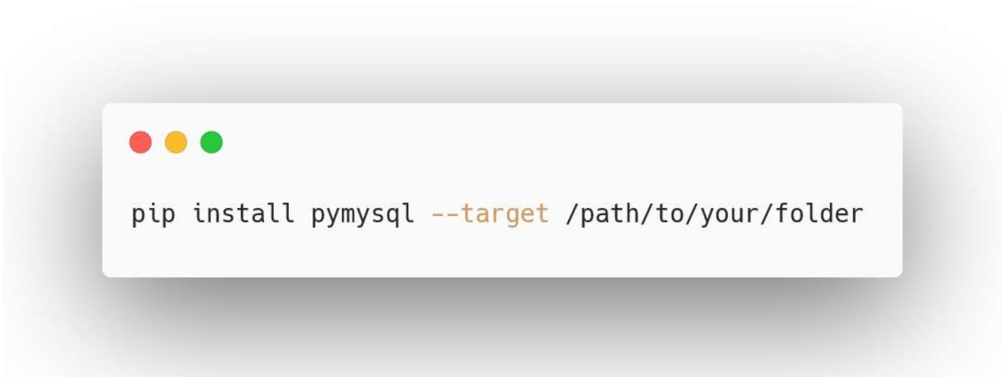
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EC2Allow",
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::backuprepositoryforimportantfilesandrds/*",
                "arn:aws:s3:::backuprepositoryforimportantfilesandrds"
            ]
        },
        {
            "Sid": "LambdaAllow",
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "s3:ListBucket",
            "Resource": [
                "arn:aws:s3:::backuprepositoryforimportantfilesandrds",
                "arn:aws:s3:::backuprepositoryforimportantfilesandrds/*"
            ]
        },
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::backuprepositoryforimportantfilesandrds/*"
        }
    ]
}
```

- Click on save

Lambda Implementation :-

You will have to create 3 lambda functions in total for the above system to function . note that every function requires you to have MySQL package installed so that lambda functions can access the same.

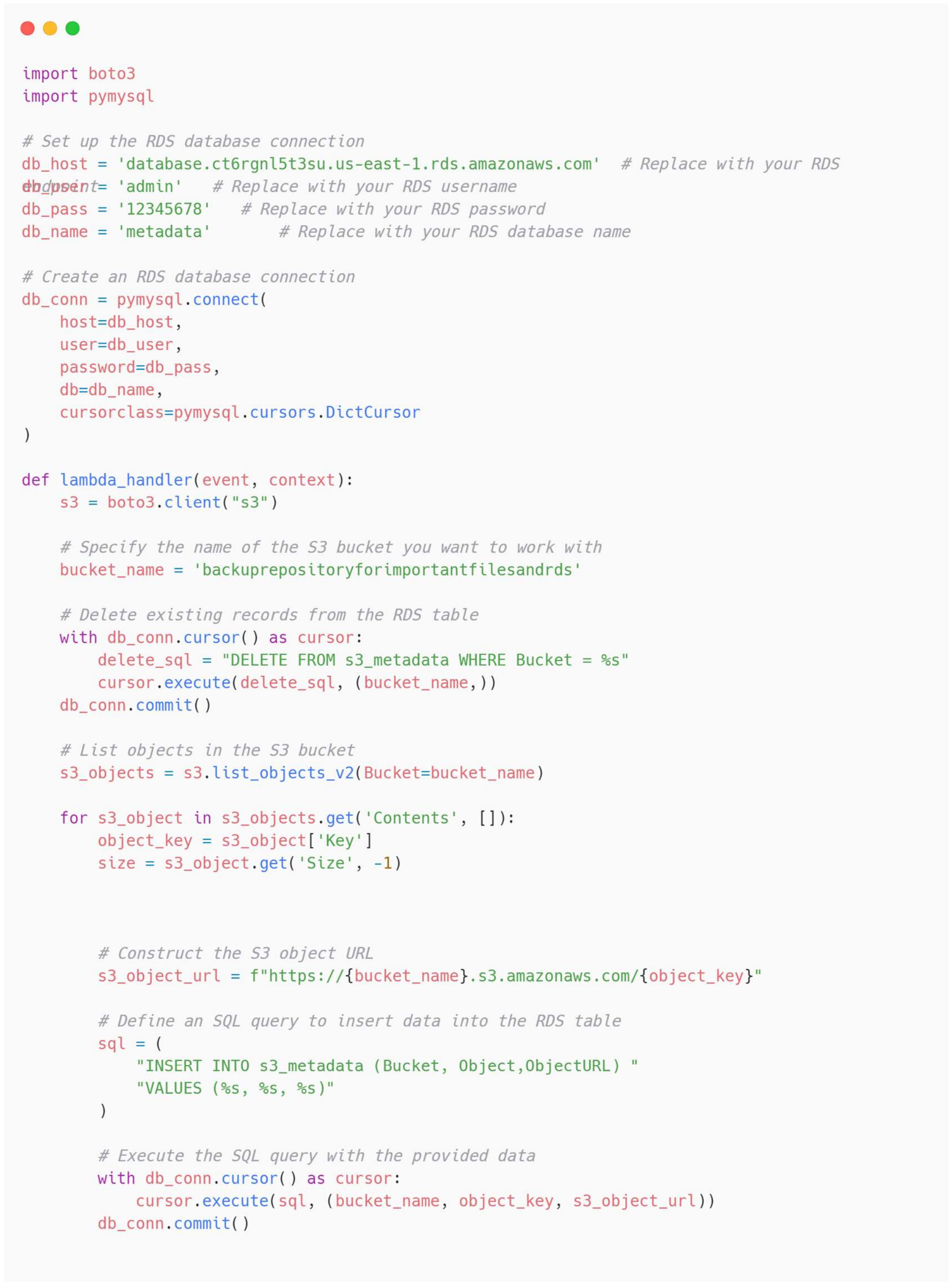
- Now create a folder in your local desktop and copy the path of the folder now use the command below to install the same: -



```
pip install pymysql --target /path/to/your/folder
```

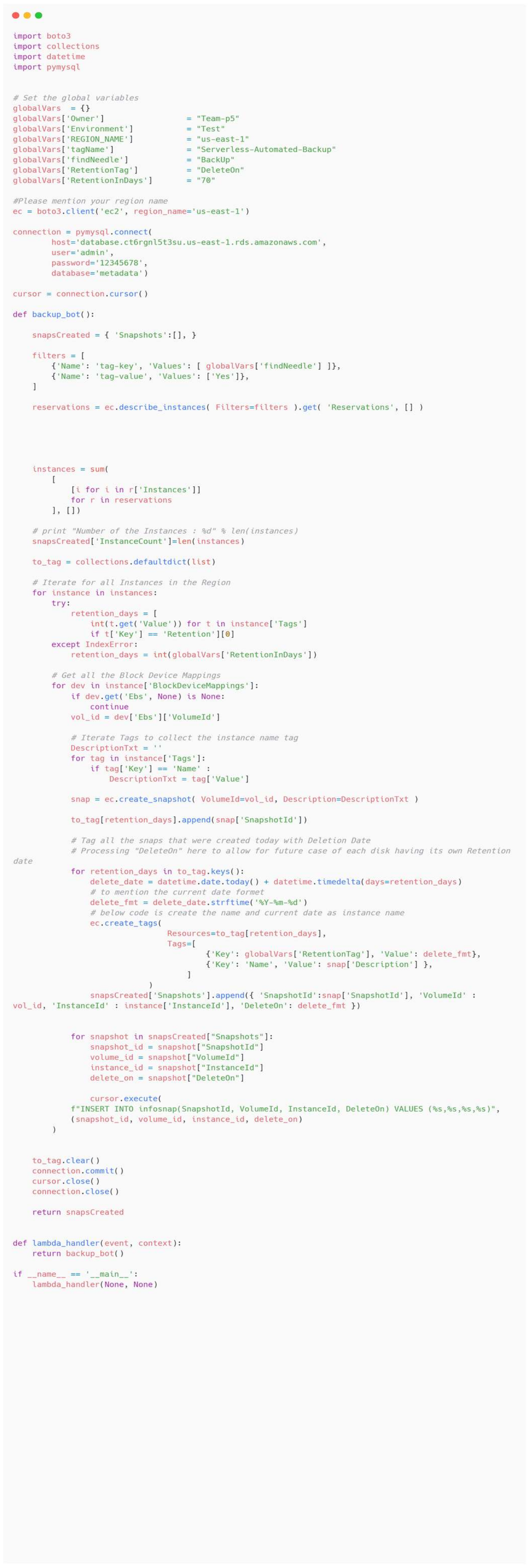
- After creating now go to the file and create lambda_function.py file in that folder.
- The codes for each function are given below but these are the common steps which needs to be followed after pasting the codes in lambda_function.py export them as zip folder
- Now create lambda function while creating lambda function choose advanced settings and select ‘labrole’ which will allow the services to talk to each other .
- Now go to that lambda function and go to the code editor below you will find the button upload from . click that and choose .zip file to import the code to lambda function .

s3updater code:-



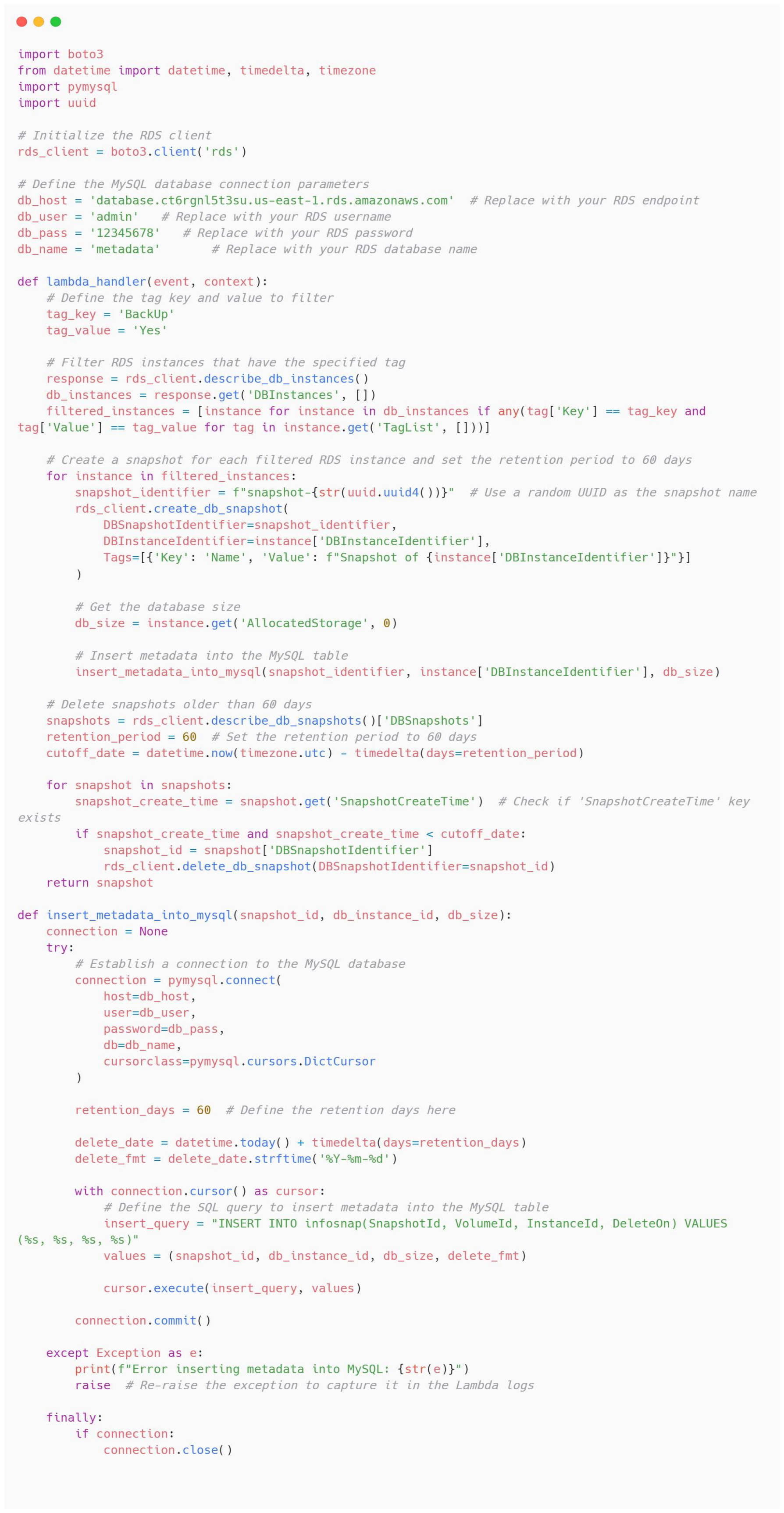
```python
import boto3
import pymysql

# Set up the RDS database connection
db_host = 'database.ct6rgnl5t3su.us-east-1.rds.amazonaws.com'  # Replace with your RDS endpoint
db_user = 'admin'   # Replace with your RDS username
db_pass = '12345678'   # Replace with your RDS password
db_name = 'metadata'       # Replace with your RDS database name

# Create an RDS database connection
db_conn = pymysql.connect(
    host=db_host,
    user=db_user,
    password=db_pass,
    db=db_name,
    cursorclass=pymysql.cursors.DictCursor
)

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    # Specify the name of the S3 bucket you want to work with
    bucket_name = 'backuprepositoryforimportantfilesandrds'

    # Delete existing records from the RDS table
    with db_conn.cursor() as cursor:
        delete_sql = "DELETE FROM s3_metadata WHERE Bucket = %s"
        cursor.execute(delete_sql, (bucket_name,))
    db_conn.commit()

    # List objects in the S3 bucket
    s3_objects = s3.list_objects_v2(Bucket=bucket_name)

    for s3_object in s3_objects.get('Contents', []):
        object_key = s3_object['Key']
        size = s3_object.get('Size', -1)



        # Construct the S3 object URL
        s3_object_url = f"https://{bucket_name}.s3.amazonaws.com/{object_key}"

        # Define an SQL query to insert data into the RDS table
        sql = (
            "INSERT INTO s3_metadata (Bucket, Object,ObjectURL) "
            "VALUES (%s, %s, %s)"
        )

        # Execute the SQL query with the provided data
        with db_conn.cursor() as cursor:
            cursor.execute(sql, (bucket_name, object_key, s3_object_url))
        db_conn.commit()
```

democode code :-



```python
import boto3
import collections
import datetime
import pymysql


# Set the global variables
globalVars  = {}
globalVars['Owner']                 = "Team-p5"
globalVars['Environment']           = "Test"
globalVars['REGION_NAME']           = "us-east-1"
globalVars['tagName']               = "Serverless-Automated-Backup"
globalVars['findNeedle']            = "BackUp"
globalVars['RetentionTag']          = "DeleteOn"
globalVars['RetentionInDays']       = "70"

#Please mention your region name
ec = boto3.client('ec2', region_name='us-east-1')

connection = pymysql.connect(
        host='database.ct6rgnl5t3su.us-east-1.rds.amazonaws.com',
        user='admin',
        password='12345678',
        database='metadata')

cursor = connection.cursor()

def backup_bot():

    snapsCreated = { 'Snapshots':[], }

    filters = [
        {'Name': 'tag-key', 'Values': [ globalVars['findNeedle'] ]},
        {'Name': 'tag-value', 'Values': ['Yes']},
    ]

    reservations = ec.describe_instances( Filters=filters ).get( 'Reservations', [] )




    instances = sum(
        [
            [i for i in r['Instances']]
            for r in reservations
        ], [])

    # print "Number of the Instances : %d" % len(instances)
    snapsCreated['InstanceCount']=len(instances)

    to_tag = collections.defaultdict(list)

    # Iterate for all Instances in the Region
    for instance in instances:
        try:
            retention_days = [
                int(t.get('Value')) for t in instance['Tags']
                if t['Key'] == 'Retention'][0]
        except IndexError:
            retention_days = int(globalVars['RetentionInDays'])

        # Get all the Block Device Mappings
        for dev in instance['BlockDeviceMappings']:
            if dev.get('Ebs', None) is None:
                continue
            vol_id = dev['Ebs']['VolumeId']

            # Iterate Tags to collect the instance name tag
            DescriptionTxt = ''
            for tag in instance['Tags']:
                if tag['Key'] == 'Name' :
                    DescriptionTxt = tag['Value']

            snap = ec.create_snapshot( VolumeId=vol_id, Description=DescriptionTxt )

            to_tag[retention_days].append(snap['SnapshotId'])

            # Tag all the snaps that were created today with Deletion Date
            # Processing "DeleteOn" here to allow for future case of each disk having its own Retention
date
            for retention_days in to_tag.keys():
                delete_date = datetime.date.today() + datetime.timedelta(days=retention_days)
                # to mention the current date formet
                delete_fmt = delete_date.strftime('%Y-%m-%d')
                # below code is create the name and current date as instance name
                ec.create_tags(
                                Resources=to_tag[retention_days],
                                Tags=[
                                        {'Key': globalVars['RetentionTag'], 'Value': delete_fmt},
                                        {'Key': 'Name', 'Value': snap['Description'] },
                                    ]
                            )
                snapsCreated['Snapshots'].append({ 'SnapshotId':snap['SnapshotId'], 'VolumeId' :
vol_id, 'InstanceId' : instance['InstanceId'], 'DeleteOn': delete_fmt })


            for snapshot in snapsCreated["Snapshots"]:
                snapshot_id = snapshot["SnapshotId"]
                volume_id = snapshot["VolumeId"]
                instance_id = snapshot["InstanceId"]
                delete_on = snapshot["DeleteOn"]

                cursor.execute(
            f"INSERT INTO infosnap(SnapshotId, VolumeId, InstanceId, DeleteOn) VALUES (%s,%s,%s,%s)",
            (snapshot_id, volume_id, instance_id, delete_on)
        )


    to_tag.clear()
    connection.commit()
    cursor.close()
    connection.close()

    return snapsCreated


def lambda_handler(event, context):
    return backup_bot()

if __name__ == '__main__':
    lambda_handler(None, None)
```

rdsbackupthingy code :-



```
import boto3
from datetime import datetime, timedelta, timezone
import pymysql
import uuid

# Initialize the RDS client
rds_client = boto3.client('rds')

# Define the MySQL database connection parameters
db_host = 'database.ct6rgnl5t3su.us-east-1.rds.amazonaws.com'  # Replace with your RDS endpoint
db_user = 'admin'   # Replace with your RDS username
db_pass = '12345678'   # Replace with your RDS password
db_name = 'metadata'       # Replace with your RDS database name

def lambda_handler(event, context):
    # Define the tag key and value to filter
    tag_key = 'BackUp'
    tag_value = 'Yes'

    # Filter RDS instances that have the specified tag
    response = rds_client.describe_db_instances()
    db_instances = response.get('DBInstances', [])
    filtered_instances = [instance for instance in db_instances if any(tag['Key'] == tag_key and
tag['Value'] == tag_value for tag in instance.get('TagList', []))]

    # Create a snapshot for each filtered RDS instance and set the retention period to 60 days
    for instance in filtered_instances:
        snapshot_identifier = f"snapshot-{str(uuid.uuid4())}"  # Use a random UUID as the snapshot name
        rds_client.create_db_snapshot(
            DBSnapshotIdentifier=snapshot_identifier,
            DBInstanceIdentifier=instance['DBInstanceIdentifier'],
            Tags=[{'Key': 'Name', 'Value': f"Snapshot of {instance['DBInstanceIdentifier']}"}]
        )

        # Get the database size
        db_size = instance.get('AllocatedStorage', 0)

        # Insert metadata into the MySQL table
        insert_metadata_into_mysql(snapshot_identifier, instance['DBInstanceIdentifier'], db_size)

    # Delete snapshots older than 60 days
    snapshots = rds_client.describe_db_snapshots()['DBSnapshots']
    retention_period = 60  # Set the retention period to 60 days
    cutoff_date = datetime.now(timezone.utc) - timedelta(days=retention_period)

    for snapshot in snapshots:
        snapshot_create_time = snapshot.get('SnapshotCreateTime')  # Check if 'SnapshotCreateTime' key
exists
        if snapshot_create_time and snapshot_create_time < cutoff_date:
            snapshot_id = snapshot['DBSnapshotIdentifier']
            rds_client.delete_db_snapshot(DBSnapshotIdentifier=snapshot_id)
    return snapshot

def insert_metadata_into_mysql(snapshot_id, db_instance_id, db_size):
    connection = None
    try:
        # Establish a connection to the MySQL database
        connection = pymysql.connect(
            host=db_host,
            user=db_user,
            password=db_pass,
            db=db_name,
            cursorclass=pymysql.cursors.DictCursor
        )

        retention_days = 60  # Define the retention days here

        delete_date = datetime.today() + timedelta(days=retention_days)
        delete_fmt = delete_date.strftime('%Y-%m-%d')

        with connection.cursor() as cursor:
            # Define the SQL query to insert metadata into the MySQL table
            insert_query = "INSERT INTO infosnap(SnapshotId, VolumeId, InstanceId, DeleteOn) VALUES
(%s, %s, %s, %s)"
            values = (snapshot_id, db_instance_id, db_size, delete_fmt)

            cursor.execute(insert_query, values)

        connection.commit()

    except Exception as e:
        print(f"Error inserting metadata into MySQL: {str(e)}")
        raise  # Re-raise the exception to capture it in the Lambda logs

    finally:
        if connection:
            connection.close()
```

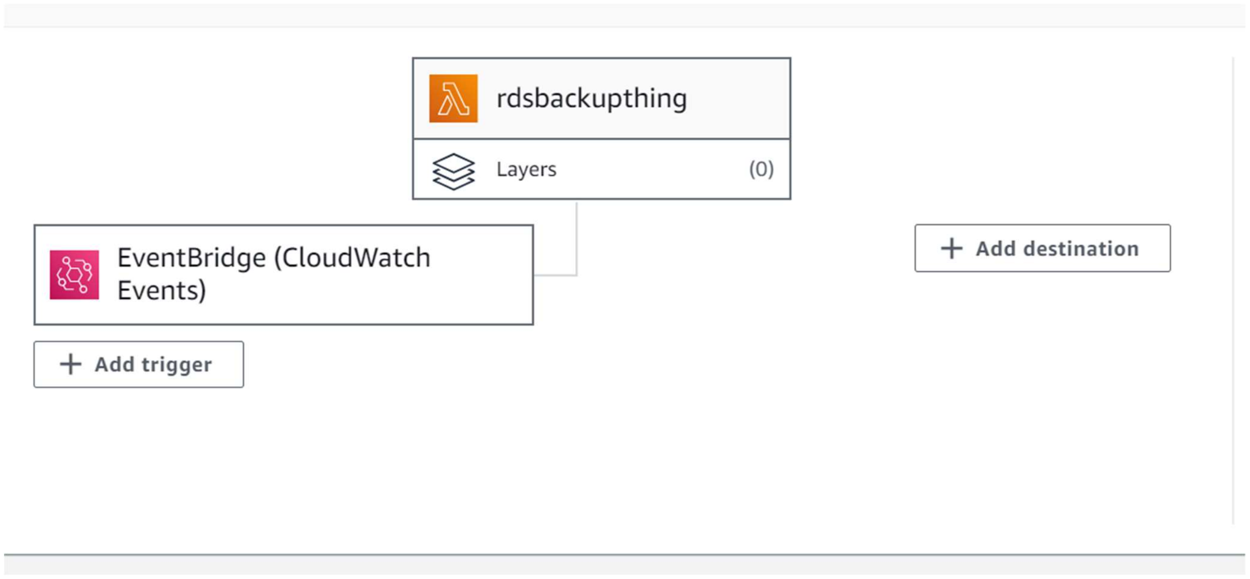
Setting up triggers for the code democode and rdsbackupthingy : -

- First go to functions of the code click on democode
- Select add triggers now search cloudwatch eventbridge
- Select create new event and give the name for the event
- Set rate to the required event and click on save

The event will be created successfully and will automatically help to trigger the events accordingly . you can see these events under cloudwatch eventbridge rules .

Remember earlier javascript code we discussed some links will differ for some buttons there are some eventbridge links change them for the respective functions (they are just links copypasted from browser search)

After configuring triggers they would look like this as follows :-



Now in the functions > configuration > function url > click on create url > select the one which can be accessed without authentication and click create . now update the same in javascript function wherever url is present with lambda url .

Setting up triggers for the s3updater : -

- First go to functions of the code click on demothingy
- Select add triggers now search for S3
- Select create new event and give a name and navigate to create

--X--