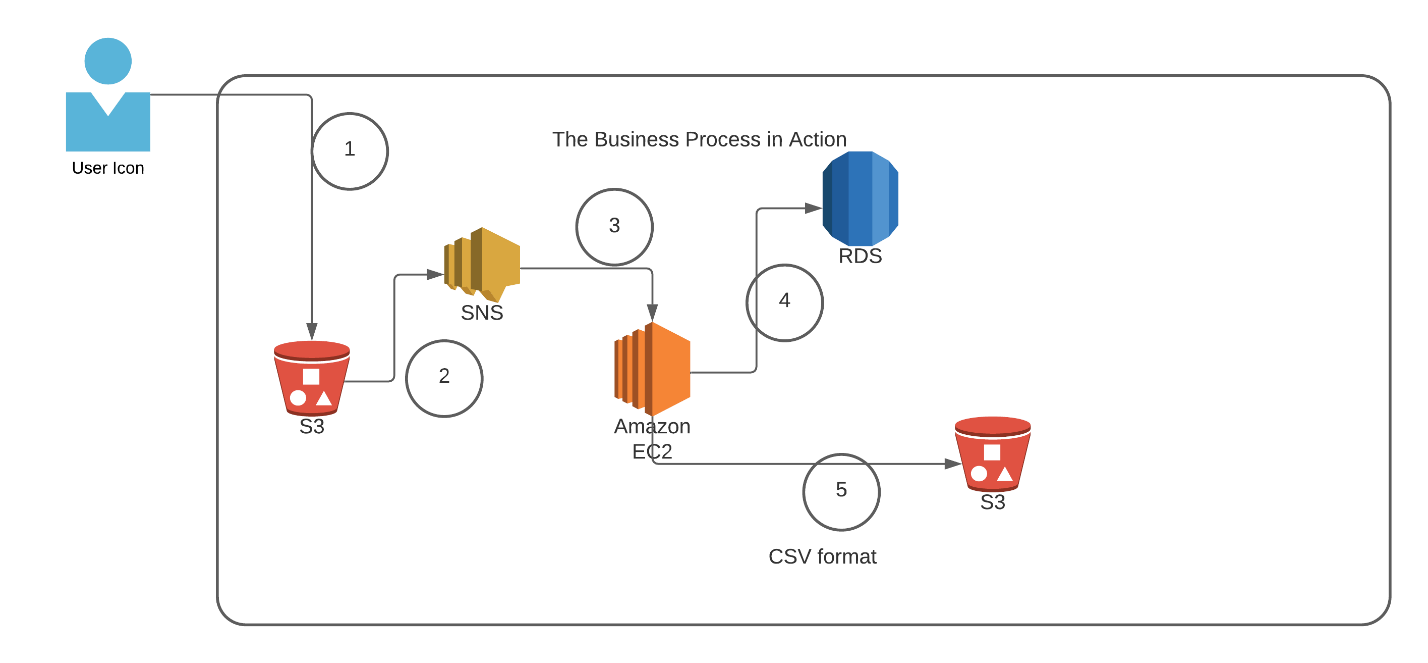
|  |  |
| --- | --- |
| **Declaration** |  |
| Questions in this exercise are intentionally complex and could be convoluted or confusing. This is by design and to simulate real life situations where customers seldom give crystal clear requirements and ask unambiguous questions. | |
| I have read the above statement and agree to these conditions | |
| I AGREE | Vishwajeet Bharti |
| <Enter your name above this line to indicate that you are in agreement> |

|  |  |
| --- | --- |
| **Instructions** |  |
| Every screenshot requested in this workbook is compulsory and carries 1 marks | |
| Your AWS account ID must be clearly visible in every screenshot using the AWS console; missing id or using someone else's id is not permitted. Such cases will be considered as plagiarism and severe penalty will be imposed. | |
| All screenshots must be in the order mentioned under "Expected Screenshots" for every step | |
| DO NOT WAIT UNTIL THE LAST MINUTE. The program office will not extend the project submission deadline under any circumstances. | |
| The file should be renamed in the format BATCH\_FIRSTNAME\_LASTNAME\_PROJECT1. For example: PGPCCMAY18\_VIJAY\_DWIVEDI\_PROJECT1.pdf | |

|  |  |
| --- | --- |
| **Resource Clean Up** |  |
| Cloud is always pay per use model and all resources/services that we consume are chargeable.  Cleaning up when you’ve completed your lab or project is always necessary. This is true whether you’re doing a lab or implementing a project at your workplace. | |
| After completing the lab, make sure to delete each resource created in reverse chronological order.  Each AWS Academy session lasts for 4 hours by default, although you can extend a session to run longer by pressing the start button to reset your session timer. At the end of each session, any resources you created in the account will be preserved. Some AWS resources, such as EC2 instances, may be automatically shut down, while other resources, such as RDS instances will be left running. | |

|  |  |
| --- | --- |
| **Architecture Implementation** | |
| 1 | The customer uploads the invoice data to S3 bucket in a text format as per their guidelines and policies. This bucket will have a policy to auto delete any content that is more than 1 day old (24 hours). |
| 2 | An event will trigger in the bucket that will place a message in SNS topic |
| 3 | A custom program running in EC2 will subscribe to the SNS topic and get the message placed by S3 event |
| 4 | The program will use S3 API to read from the bucket, parse the content of the file and create a CSV record and save the details in an RDS database |
| 5 | The program will use S3 API to write CSV record to destination S3 bucket as new S3 object. |

**Architecture diagram**

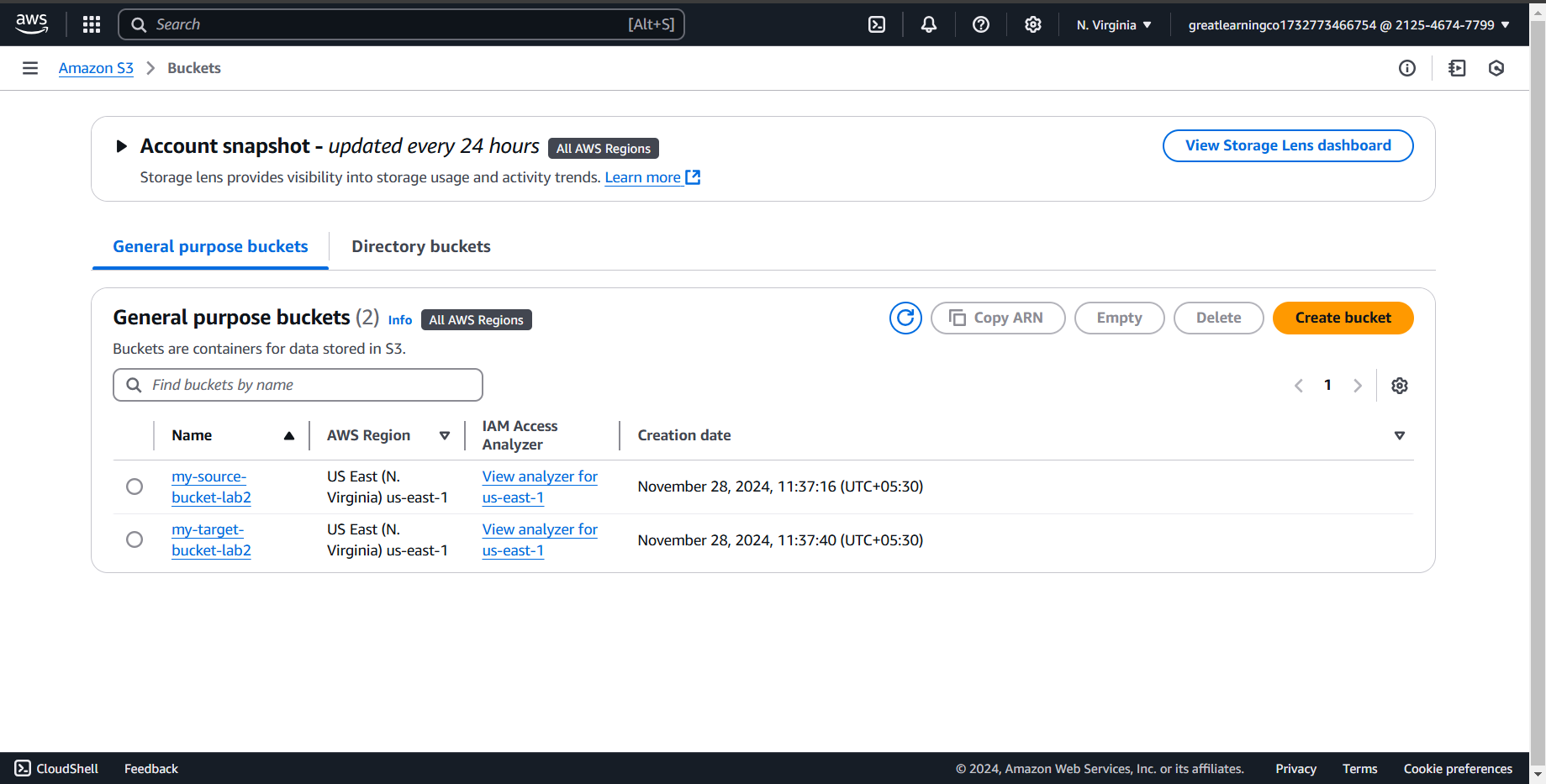


# Step 1: SNS and S3 topic creation

|  |  |
| --- | --- |
| Note | The custom program codebase and sample invoice [h](https://drive.google.com/drive/folders/1h2wjP5fyDrOSNcvq70teMDa0w-K0q5dT?usp=sharing)ave been shared along with this workbook on the LMS. |

|  |  |
| --- | --- |
| Step number | a |
| Step name | Creation of Source and target buckets |
| Instructions | 1. Navigate to S3 using the Services button at the top of the screen 2. Select "Create Bucket" 3. Enter a source bucket name and use the default options for the rest of the fields 4. Click on "Create Bucket' 5. Repeat the above steps to create a target bucket |
| Expected screenshots | 1) Screen showing created S3 source and target buckets |

# <Insert screenshot for a(1) here>



|  |  |
| --- | --- |
| Step number | b |
| Step name | Creation of SNS subscription |
| Instructions | 1. Navigate to SNS -> Topics 2. Click on "Create Topic" 3. Enter the following fields   Name : S3toEC2Topic  The other options can be ignored for now   1. Click on Create Topic |
| Expected screenshots | 1) Creation of SNS topic |

# <Insert screenshot for b(1) here>

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Step number | c |
| Step name | Modification of SNS Access Policy |

|  |  |
| --- | --- |
| Instructions | 1) Navigate to SNS -> Topics and select the topic created in the previous step 2) Note down the ARN shown in the topic details 2) Click on Edit and select "Access Policy".  3) Replace the text in the JSON editor with the following  {  "Version": "2012-10-17",  "Id": "example-ID",  "Statement": [  {  "Sid": "example-statement-ID",  "Effect": "Allow",  "Principal": {  "AWS":"\*"  },  "Action": [  "SNS:Publish"  ],  "Resource": "**SNS-topic-ARN**",  "Condition": {  "ArnLike": { "aws:SourceArn": "arn:aws:s3:\*:\*:**bucket-name**" },  "StringEquals": { "aws:SourceAccount": "**bucket-owner-account-id**" }  }  }  ]  }     1. Note: aws:SourceArn: replace the value from arn [Amazon S3](https://us-east-1.console.aws.amazon.com/s3/get-started?region=us-east-1&bucketType=general)>[Buckets](https://us-east-1.console.aws.amazon.com/s3/buckets?region=us-east-1&bucketType=general)>**my-source-bucket-lab2**   **Resource:replace from sns topic arn**  4) Replace the bold text with the SNS topic ARN, source bucket name and your AWS account ID respectively. 5) Click on Save Changes |
| Expected screenshots | 1) JSON Editor screen |

Notes:  
https://www.youtube.com/watch?v=bDBX5rAt3eQ

# <Insert screenshot for c(1) here>

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Step number | d |
| Step name | Configuring SNS notifications for S3 |
| Instructions | 1. Navigate to S3 and select the source bucket created in Step 1 (a) 2. Select Properties and scroll down to Event Notifications and select it 3. Select "Create Event Notification" 4. Fillup the details as follows   Name : S3PutEvent  Select PUT from the list of radio buttons  Destination : Select SNS Topic  SNS : Select S3ToEC2Topic     1. Save Changes |
| Expected screenshots | 1) Event Configuration Screen |

**<Insert screenshot for d(1) here>**

A screenshot of a computer

Description automatically generated

Step 2: Run the custom program in the EC2 instance

|  |
| --- |
| Step number a  Step name Creation of the EC2 instance and RDS instance  Instructions 1) Navigate to EC2 -> Instances   1. Create an EC2 instance with the following parameters   AMI : Amazon Linux 2  VPC : Default  Security group : Ports 22 and 8080 should be opened     1. Navigate to RDS 2. Create an RDS instance with the following parameters:     Engine type : MySql  Template : Dev/Test  Set the username and password as required  DB Instance class : Burstable  Instance type : t3.micro  Public Access : Yes  VPC Security group : Create New ()    Under Additional Configuration, add an initial database name. Take note of this name as it will be required later.    Uncheck “Enable Enhanced Monitoring”    Ensure that the security group created by the RDS deployment has port 3306 open for all incoming connections from all sources.  Expected 1) List of instances after creation of EC2 screenshots instance  2) List of RDS instances |

# <Insert screenshot for a(1) here>

# A screenshot of a computer Description automatically generated

# <Insert screenshot for a(2) here>

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Step number | b |
| Step name | Assignment of IAM role for EC2 instance |
| Instructions | 1. Navigate back to EC2- > Instances 2. Select the EC2 instance created in the previous step and select Actions-> Security -> Modify IAM role 3. Select the role LabInstanceProfile from the dropdown and click on Save |
| Expected screenshots | 1) Modify IAM role screen |

# <Insert screenshot for b(1) here>

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Step number | c |  |
| Step name | Configuration and Uploading of custom program |  |
| Instructi ons | 1. Download the file **docproc-new.zip** on your machine 2. Unzip the downloaded file 3. Enter the unzipped folder and open the file [views.py](http://views.py/) in the API folder using a text editor 4. In line number 19-24, modify the target bucket name to the one created in Step 2 (a) and modify the hostname, username, password and database variables to the values set while creating the RDS database and save the file 5. Copy the folder docproc-new to the home folder of the EC2 instance created in Step 3(a) using scp. Use the command given below   *scp -i <pem> -r ./docproc-new ec2user@<ip>:/home/ec2-user* |  |
| Expecte d screens hots | 1) Modifying of the [views.py](http://views.py/) file to point to the target bucket | 2)Copying the folder to the  EC2 instance |

Notes:  
chmod 400 sns2.pem

Transfer the file to ec2 user

scp -i "key281124.pem" ~/docproc-new.zip ec2-user@ec2-54-227-23-17.compute-1.amazonaws.com:/home/ec2-user/

--remove directory

rm -r mydirectory

--remove file

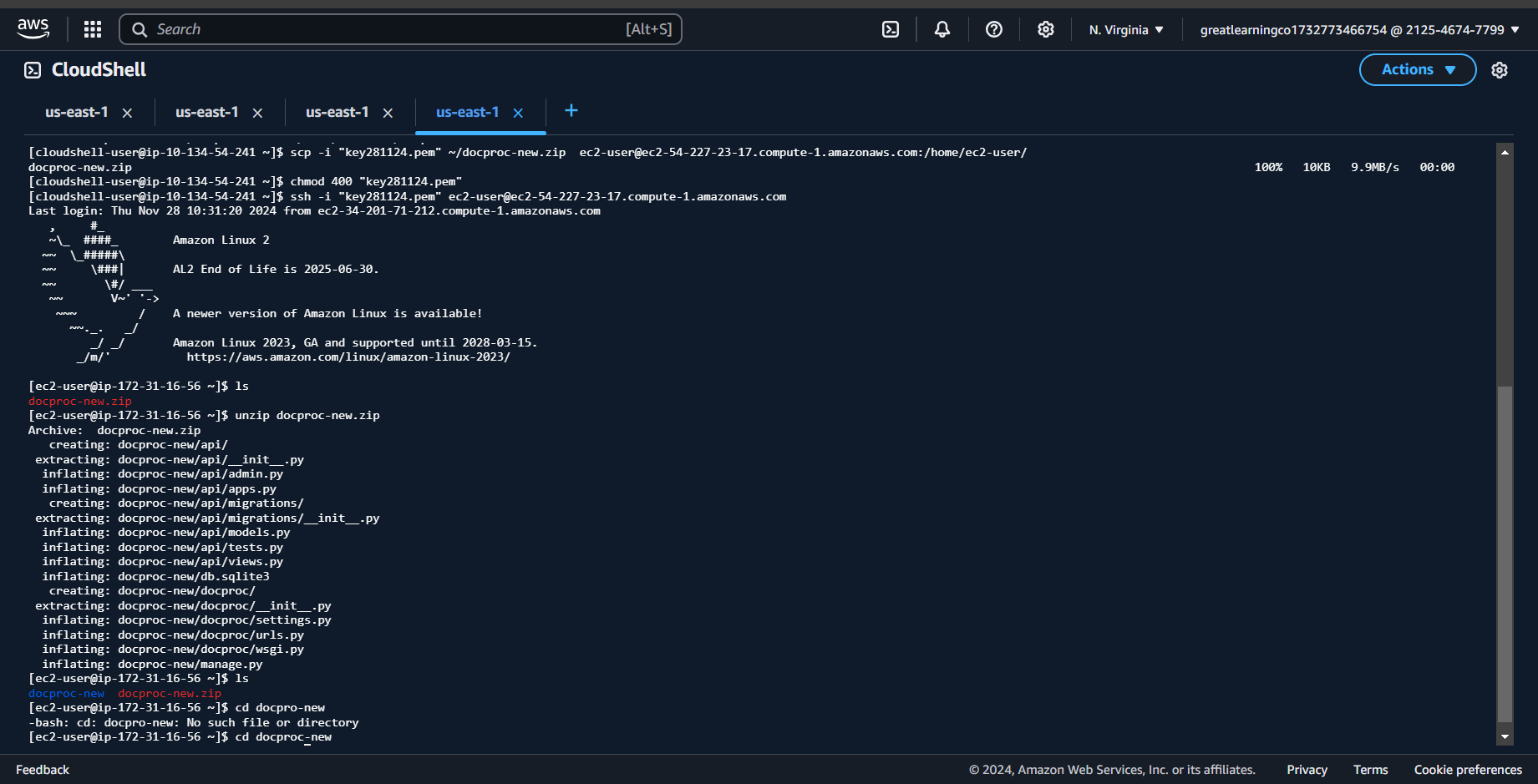
rm file.txt

# <Insert screenshot for c(1) here>

A screenshot of a computer

Description automatically generated

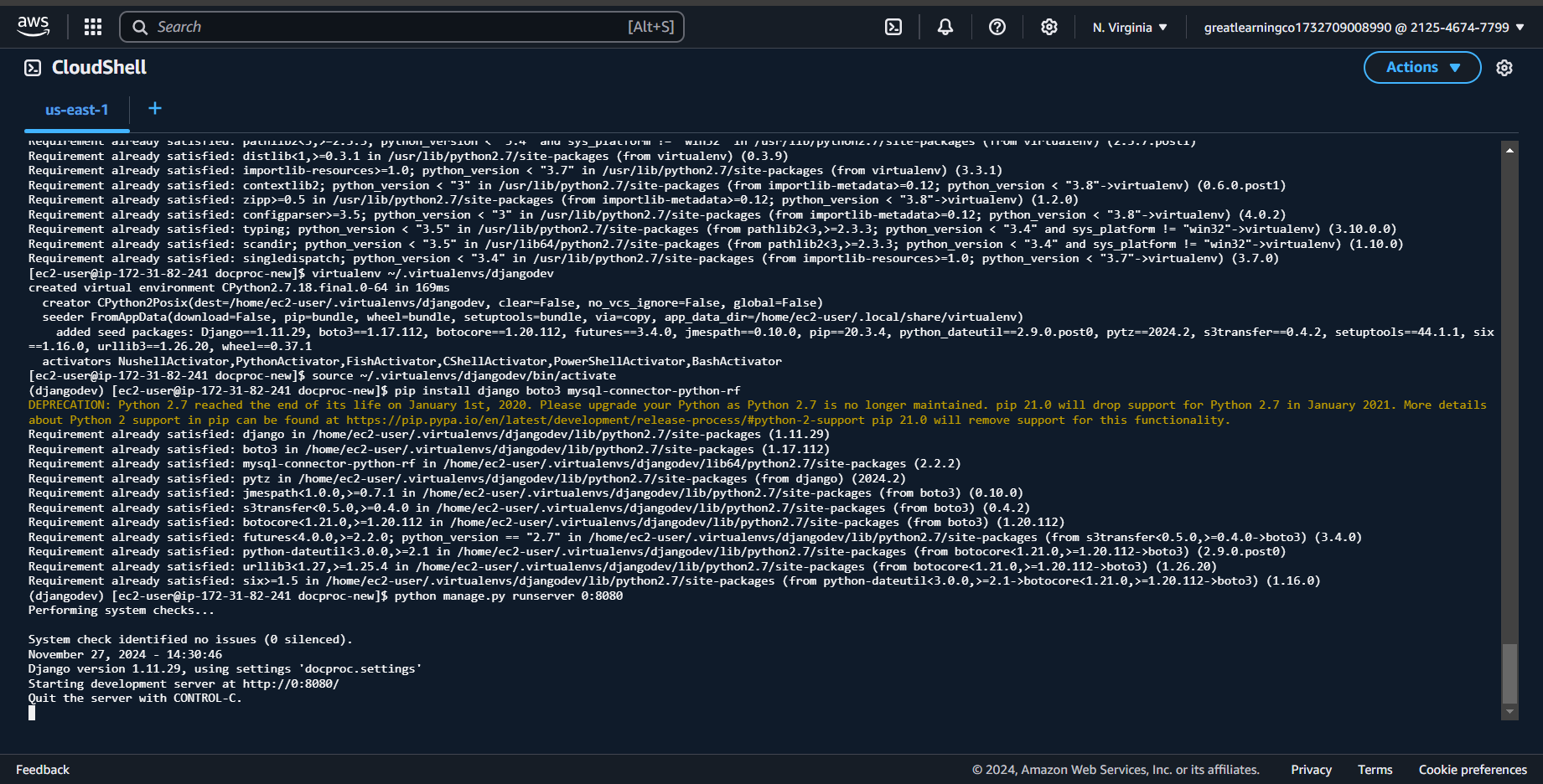
**<Insert screenshot for c(2) here>**



Step 3: Creation and Verification of SNS subscription and Generation of CSV file

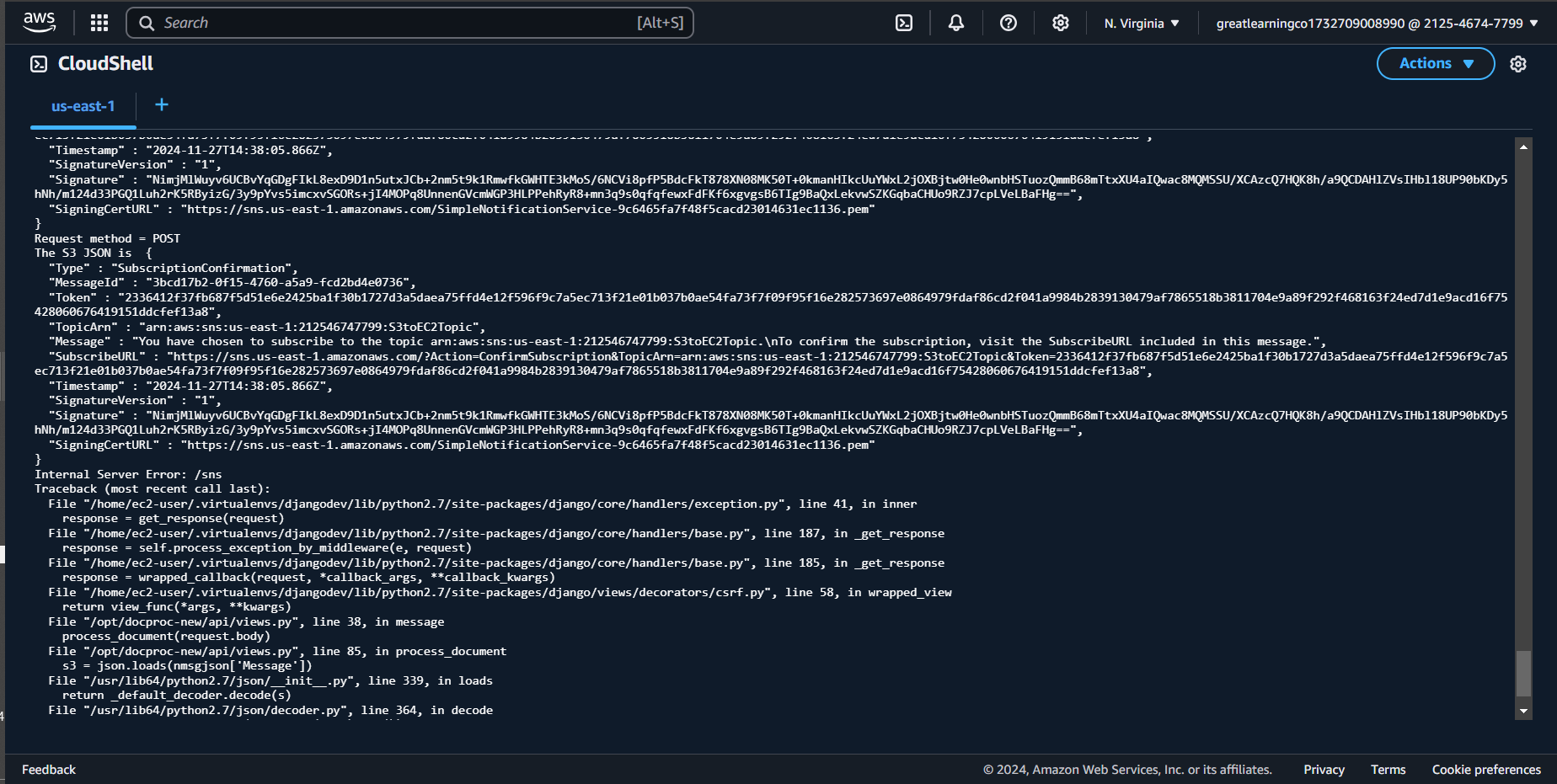
|  |  |
| --- | --- |
| Step number | a |
| Step name | Starting the  EC2 custom program |
| Instructions | 1. Log into the EC2 instance using SSH 2. Run the followng commands after successful SSH to start the server sudo cp -r docproc-new /opt sudo chown ec2-user:ec2-user -R /opt cd /opt/docproc-new sudo yum update   sudo yum install python-pip -y  python -m pip install --upgrade pip setuptools sudo pip install virtualenv virtualenv ~/.virtualenvs/djangodev source ~/.virtualenvs/djangodev/bin/activate pip install django pip install boto3  pip install mysql-connector-python-rf python manage.py runserver 0:8080    **Keep this terminal window open throughout the rest of the exercise** |
| Expected screenshots | 1) Server in  waiting state |

# <Insert screenshot for a(1) here>



|  |  |
| --- | --- |
| Step number | b |
| Step name | Creation of  SNS  subscription |
| Instructions | 1. Navigate to SNS in the AWS Console and select the topic S3ToEC2Topic 2. Click on Create Subscription 3. Enter the following details   Protocol : HTTP  Endpoint : http://<host>:8080/sns where <host> in the public IP of the EC2 instance Click on Create Subscription   1. In the EC2 terminal window, look for the field "SubscribeURL" and copy the entire link given   **Note: If a message is seen "ValueError: No JSON object could be decoded", it can be safely ignored**   1. Paste that link into a browser window to verify the SNS subscription (Ignore any messages received in the web browser) |
| Expected screenshots | 1)  Subscription URL in EC2  terminal  Window |

# <Insert screenshot for b(1) here>



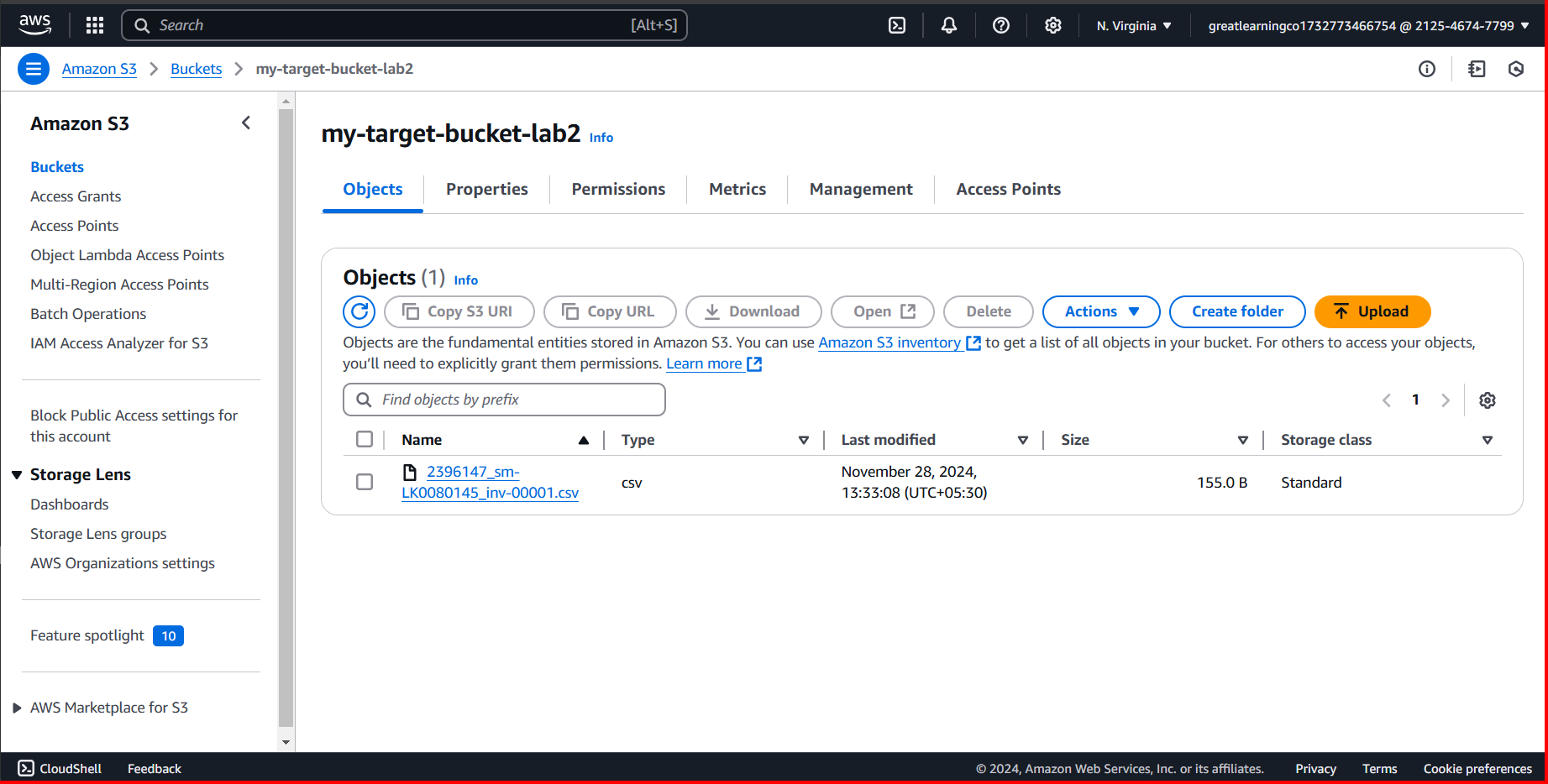
|  |  |
| --- | --- |
| Step number | c |
| Step name | Generation of CSV file |
| Instructions | 1. Download the file **docproc-invoice.txt** provided with this workbook 2. Navigate to S3 in the AWS Console 3. Upload the sample invoice file to the source S3 bucket using the default options 4) Verify that a CSV file is generated in the target S3 bucket. This may take a few minutes   5) (Optional) Login to the RDS instance using your preferred MySQL client and check the table created inside the specified database. |
| Expected screenshots | 1) Generated CSV file in the target S3 bucket |

scp -i "key281124.pem" ~/docproc-new.zip [ec2-user@ec2-54-227-23-17.compute-1.amazonaws.com:/home/ec2-user/](mailto:ec2-user@ec2-54-227-23-17.compute-1.amazonaws.com:/home/ec2-user/)

to remove file

rm <filename>

# <Insert screenshot c(1) here>



**Answer the following questions**

Q1 Which of the following properties of an AWS resource is sufficient and necessary to uniquely identify it across all of AWS?

a) ARN

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1. Region and ARN 2. ARN and Account number 3. Depends on the resource used | | |
|  | Enter your answer here | a) ARN |  |
| Q2 | Which of the following step numbers in Step 1 allowed S3 to publish to the SNS topic created?   1. 1(a) 2. 1(c) 3. 1(d) 4. 1(b) | | |
|  | Enter your answer here | 1(c) |  |
| Q3 | Which port is being used by SNS to send the notification to the custom program?   1. 8081 2. 80 3. 8080 4. 8065 | | |
|  | Enter your answer here | c) 8080 |  |
| Q4 | How many IAM roles can be attached to an EC2 instance at a time?   1. 2 2. 3 3. 1 4. Depends on the policies required | | |
|  | Enter your answer here | c) 1 |  |

Q5 As a product manager, how would you describe the benefits of this architecture to an client, as compared to an equivalent on-premises architecture?

|  |  |
| --- | --- |
|  | **Benefits of AWS Cloud Architecture vs. On-Premises Architecture**   1. **Scalability and Flexibility**:    * **Cloud**: AWS allows you to scale resources **up or down** as needed, providing flexibility to handle varying workloads. For example, if the customer experiences a sudden spike in invoice uploads, you can scale up resources like EC2 instances or S3 storage easily, without worrying about capacity planning.    * **On-Premises**: Scaling in an on-premises environment often requires purchasing and installing additional hardware, which can be time-consuming and costly. 2. **Cost Efficiency (Pay-as-You-Go)**:    * **Cloud**: With AWS, you're on a **pay-as-you-go** model, meaning you only pay for the resources you use (like storage in S3, compute power in EC2, etc.). This is more cost-effective because there are no upfront hardware costs or long-term commitments. You can also take advantage of **auto-scaling** to only use the resources you need.    * **On-Premises**: On-premises infrastructure often requires a significant initial investment in hardware and ongoing maintenance costs. Additionally, you may over-provision resources to handle peak loads, leading to inefficiencies and wasted costs. 3. **High Availability and Reliability**:    * **Cloud**: AWS provides built-in redundancy and **high availability**. For example, data in S3 is automatically replicated across multiple availability zones (AZs), ensuring that even if one AZ goes down, your data is safe and accessible. You can also use features like **RDS multi-AZ deployments** for database high availability.    * **On-Premises**: Achieving high availability on-premises typically requires complex configurations, such as setting up failover servers, redundant networking, and disaster recovery systems, which can be expensive and difficult to manage. 4. **Automated Management and Monitoring**:    * **Cloud**: AWS offers automated monitoring and management tools (like **CloudWatch** for logs and metrics) that help you track system performance, set alarms, and troubleshoot issues in real-time. For example, **AWS Lambda** can automatically trigger processes based on S3 events, allowing you to handle invoice data without manual intervention.    * **On-Premises**: Managing on-premises infrastructure requires manual monitoring and troubleshooting. This often means investing in additional software and hardware, and relying on your team to monitor and respond to issues 24/7. 5. **Security and Compliance**:    * **Cloud**: AWS has built-in security features like **IAM** for access control, **encryption** for data protection, and **audit logs** for compliance. AWS also complies with a wide range of industry standards and certifications, which makes it easier to meet regulatory requirements (e.g., GDPR, HIPAA).    * **On-Premises**: Achieving the same level of security and compliance on-premises requires investing in security infrastructure, monitoring, and regular audits, which can be costly and resource-intensive. 6. **Faster Time to Market**:    * **Cloud**: AWS provides a wide range of pre-built services and features (like S3 for storage, EC2 for compute, RDS for managed databases, SNS for messaging, etc.) that can be quickly integrated into your application. This speeds up development and deployment times, allowing the business to get up and running faster.    * **On-Premises**: With on-premises infrastructure, you need to provision and configure hardware, set up networks, and ensure everything works together, which can take months and delay time-to-market. 7. **Ease of Maintenance**:    * **Cloud**: With AWS, maintenance tasks like software updates, patching, and backup are automated or can be handled easily with managed services. For example, AWS **RDS** automatically handles database patching, backups, and scaling.    * **On-Premises**: On-premises infrastructure requires a dedicated team to handle maintenance tasks such as patching operating systems, upgrading software, and managing backups, which increases operational overhead. 8. **Global Reach and Disaster Recovery**:    * **Cloud**: AWS has a global network of data centers, which allows you to deploy applications in multiple regions to serve global customers with low latency. You can also easily set up **cross-region replication** for disaster recovery.    * **On-Premises**: Setting up global availability and disaster recovery for on-premises infrastructure is complex and costly, often requiring multiple data centers in different regions and additional resources to manage the infrastructure. 9. **Innovation and Integration**:    * **Cloud**: AWS continuously releases new features, services, and tools that allow you to integrate advanced capabilities like **machine learning** (e.g., Amazon SageMaker), **data analytics** (e.g., Amazon Redshift), and **serverless computing** (e.g., AWS Lambda) into your architecture without having to maintain complex infrastructure.    * **On-Premises**: Integrating new technologies with on-premises systems often requires significant manual effort, upgrades, and potential changes to the underlying hardware, making it slower and more expensive to innovate. |
|  |

|  |  |
| --- | --- |
| **Grades distribution** |  |
| MCQs | 6 (1.5 mark each) |
| Subjective questions | 2 marks |
| Implementation screenshots | 12 marks (1 marks each) |
| Total | 20 marks |

**Steps followed**  
  
**Step 1: SNS and S3 Topic Creation**

**Step 1a: Creation of Source and Target S3 Buckets**

1. **Navigate to S3**:
   * Open the **AWS Management Console**.
   * In the **Services** search bar at the top, type **S3** and select it from the list.
2. **Create the Source Bucket**:
   * Click on **Create Bucket**.
   * In the **Bucket name** field, enter a unique name for your source bucket (e.g., my-source-bucket).
   * Leave the rest of the settings at their default values.
   * Click **Create Bucket**.
3. **Create the Target Bucket**:
   * Repeat the same process to create a **Target Bucket** (e.g., my-target-bucket).

**Expected Outcome**: You should have two buckets created:

* + **Source bucket**: For uploading invoice data.
  + **Target bucket**: Where the processed CSV files will be saved.

**Expected Screenshots**:

* Screenshot showing both source and target buckets.

**Step 1b: Creation of SNS Subscription**

1. **Navigate to SNS**:
   * In the **AWS Management Console**, search for **SNS** and select **Simple Notification Service**.
2. **Create the SNS Topic**:
   * Click on **Create topic**.
   * For **Topic type**, select **Standard**.
   * In the **Topic name** field, enter S3toEC2Topic.
   * Leave other fields with default settings (you can ignore for now).
   * Click **Create Topic**.

**Expected Outcome**: You will have an SNS topic called S3toEC2Topic.

**Expected Screenshot**:

* Screenshot showing the SNS topic creation screen.

**Step 1c: Modification of SNS Access Policy**

1. **Navigate to SNS Topic**:
   * In the **SNS Dashboard**, find the **S3toEC2Topic** you just created.
2. **Edit Access Policy**:
   * Click on the **Topic ARN** (Amazon Resource Name) of the SNS topic.
   * Click on **Edit**.
   * Under **Access Policy**, select **Edit** and replace the JSON with the following:

json

Copy code

{ "Version": "2012-10-17", "Id": "example-ID", "Statement": [ { "Sid": "example-statement-ID", "Effect": "Allow", "Principal": { "AWS": "\*" }, "Action": [ "SNS:Publish" ], "Resource": "SNS-topic-ARN", "Condition": { "ArnLike": { "aws:SourceArn": "arn:aws:s3:\*:\*:bucket-name" }, "StringEquals": { "aws:SourceAccount": "bucket-owner-account-id" } } } ] }

* + Replace the placeholders with:
    - SNS-topic-ARN: Your SNS topic ARN (you can find this under **Topic details**).
    - bucket-name: The source bucket name you created earlier.
    - bucket-owner-account-id: Your AWS account ID.

1. **Save Changes**:
   * Click on **Save Changes**.

**Expected Outcome**: The SNS topic's access policy will now allow S3 to publish events.

**Expected Screenshot**:

* Screenshot showing the modified JSON editor screen with your values replaced.

**Step 1d: Configuring SNS Notifications for S3**

1. **Navigate to the Source S3 Bucket**:
   * Go to **S3** in the AWS Management Console.
   * Select the source bucket you created earlier.
2. **Configure Event Notification**:
   * Under the **Properties** tab, scroll down to **Event Notifications**.
   * Click on **Create Event Notification**.
   * Fill in the following details:
     + **Name**: S3PutEvent.
     + **Event types**: Select **PUT** (to trigger events when files are uploaded).
     + **Destination**: Select **SNS Topic**.
     + **SNS**: Select the SNS topic S3ToEC2Topic you created earlier.
   * Click **Save changes**.

**Expected Outcome**: The S3 bucket is now configured to send event notifications to the SNS topic whenever an object is uploaded.

**Expected Screenshot**:

* Screenshot showing the Event Notification configuration screen.

**Step 2: Run the Custom Program in the EC2 Instance**

**Step 2a: Creation of EC2 and RDS Instances**

1. **Create EC2 Instance**:
   * Navigate to the **EC2 Dashboard** in the AWS Console.
   * Click **Launch Instance**.
   * Choose the **Amazon Linux 2** AMI.
   * Select **t2.micro** for the instance type (or another free-tier eligible instance type).
   * In the **Network** section, select **default VPC**.
   * In the **Security group** section, create a new security group allowing **SSH (port 22)** and **HTTP (port 8080)** access.
   * Click **Launch** and select or create a new key pair for SSH access.
2. **Create RDS Instance**:
   * Navigate to **RDS** in the AWS Management Console.
   * Click **Create database**.
   * Choose **MySQL** as the engine type.
   * Under **Templates**, select **Dev/Test**.
   * Set the **DB instance class** to **t3.micro**.
   * Set **Public access** to **Yes** (to allow access from EC2).
   * **VPC security group**: Create a new security group or use an existing one to allow **port 3306** for MySQL access.
   * Set the initial database name (e.g., invoices).
   * Click **Create database**.

**Expected Outcome**:

* The EC2 instance and RDS instance should be successfully created.

**Expected Screenshots**:

1. List of EC2 instances after creation.
2. List of RDS instances after creation.

**Step 2b: Assignment of IAM Role for EC2 Instance**

1. **Navigate to EC2**:
   * In the **EC2 Dashboard**, select the EC2 instance you just created.
   * Under **Actions**, choose **Security** → **Modify IAM role**.
   * Select the IAM role LabInstanceProfile (or the one that grants necessary permissions like AmazonS3FullAccess, AmazonRDSFullAccess).
   * Click **Save**.

**Expected Outcome**: The EC2 instance will now have the necessary permissions to interact with S3 and RDS.

**Expected Screenshot**:

* Screenshot showing the IAM role assignment screen.

**Step 2c: Configuration and Uploading of Custom Program**

1. **Download and Unzip the Program**:
   * Download the docproc-new.zip file from your LMS (or wherever the code is provided).
   * Unzip the downloaded file on your local machine.
2. **Edit the**views.py**File**:
   * Open the views.py file located in the API folder within the unzipped directory.
   * On lines 19-24, update:
     + **Target bucket name**: Set this to the target S3 bucket you created.
     + **RDS details**: Set the hostname, username, password, and database name according to your RDS instance.
3. **Upload the Code to EC2**:
   * Use scp (Secure Copy Protocol) to upload the folder to the EC2 instance:

bash

Copy code

scp -i <pem-file> -r ./docproc-new ec2-user@<ec2-public-ip>:/home/ec2-user

**Expected Outcome**: The code should be uploaded to the EC2 instance and ready for execution.

**Expected Screenshots**:

* Screenshot showing the modified views.py file.
* Screenshot showing the scp upload command.

**Step 3: Creation and Verification of SNS Subscription and Generation of CSV File**

**Step 3a: Starting the EC2 Custom Program**

1. **SSH into the EC2 Instance**:
   * SSH into your EC2 instance:

bash

Copy code

ssh -i <pem-file> ec2-user@<ec2-public-ip>

1. **Run the Custom Program**:
   * Run the following commands to set up and start the server:

bash

Copy code

sudo cp -r docproc-new /opt sudo chown ec2-user:ec2-user -R /opt cd /opt/docproc-new sudo yum update sudo yum install python-pip -y python -m pip install --upgrade pip setuptools sudo pip install virtualenv virtualenv ~/.virtualenvs/djangodev source ~/.virtualenvs/djangodev/bin/activate pip install django boto3 mysql-connector-python-rf python manage.py runserver 0:8080

* + Keep this terminal open to maintain the server running.

**Expected Outcome**: The EC2 server should be in a waiting state, ready to process incoming SNS messages.

**Expected Screenshot**:

* Screenshot showing the EC2 server in a waiting state.

**Step 3b: Creation of SNS Subscription**

1. **Navigate to SNS**:
   * Go to **SNS** → **Topics**.
   * Select the S3ToEC2Topic you created earlier.
2. **Create a Subscription**:
   * Click **Create Subscription**.
   * Set **Protocol** to **HTTP**.
   * For **Endpoint**, enter:

bash

Copy code

http://<ec2-public-ip>:8080/sns

Replace <ec2-public-ip> with your EC2 instance's public IP address.

* + Click **Create Subscription**.

1. **Confirm Subscription**:
   * In the EC2 terminal, look for the **SubscribeURL** and open the link in your browser to confirm the subscription.

**Expected Outcome**: The SNS subscription will be created and confirmed.

**Expected Screenshot**:

* Screenshot showing the subscription URL in the EC2 terminal window.

**Step 3c: Generation of CSV File**

1. **Upload Sample Invoice File**:
   * Download the docproc-invoice.txt file from your workbook.
   * Go to **S3** and upload the file to the **source bucket**.
2. **Verify CSV Generation**:
   * Wait for a few minutes for the EC2 program to process the file and generate a CSV.
   * Check the **target S3 bucket** for the generated CSV file.
3. **Verify RDS Database** (Optional):
   * Log in to your RDS instance using MySQL and verify that the data has been inserted into the database.

**Expected Outcome**: A CSV file should appear in the target S3 bucket.

**Expected Screenshot**:

* Screenshot showing the generated CSV file in the target S3 bucket.

**Notes:**  
If need to upload folder on cloud shell then use zip and unzip process