Lab Report 4

Amazon Web Services Cloud 9

CSC 452-001

Andrew Almond

11/1/19

Introduction

From the lab, the further use of Amazon's services through Cloud 9 greatly increase the scope of capabilities from the last lab. Cloud 9 leads to possibilities of managing the former used EC2 instances and their corresponding data as well as the now learned S3 service with its share of buckets with their attributes.

Methods & Algorithms

For each problem, Amazon's Cloud 9 service was utilized to run Python programs for managing and viewing other services' instances/buckets and their corresponding information.

Problem 1:

All of the requirements from this problem called for management of resources from the EC2 service. The first three (parts a-d) controlled the actual EC2 instance targeted by gathering information, stopping, starting, or rebooting said instance. Parts e through g had to deal with the creation or deletion of the key pair to the given instance as well as the information. The final parts did the same for the desired security group.

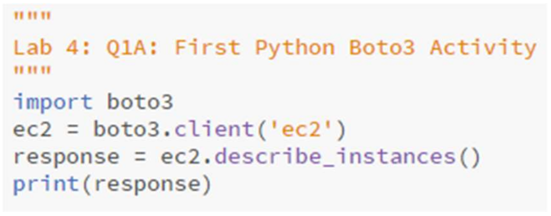
Problem 2:

Unlike problem 1, the focus is on using the S3 service of Amazon's. Here, a "bucket" can be made (like in part a) to where items can be stored and retrieved (parts c and d). Like in problem 1, the information of the desired target(s) can be gathered as well (the buckets in part b, the policy in part f, and the control list in part g). Finally, the bucket can be allowed access to temporarily with a presigned URL like in part e.

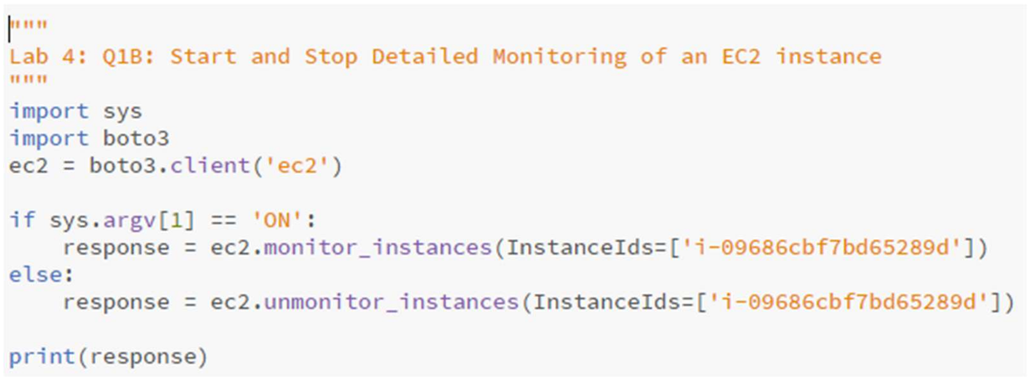
Experiments

Problem 1:

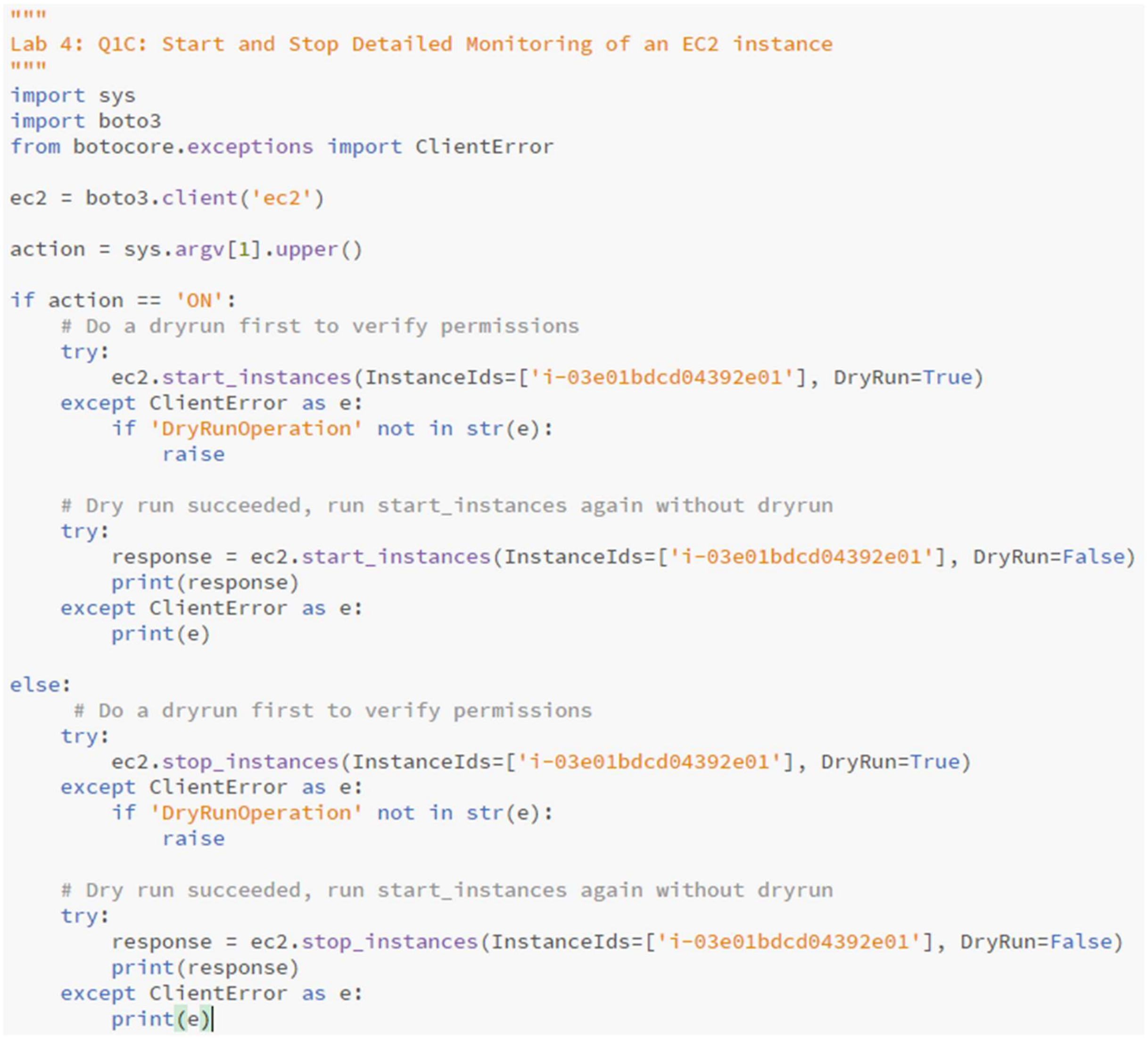
All of the parts in this problem could be implemented and run without extra arguments besides part c. This part required specifying "ON" or "OFF" to set the state of the targeted instance.



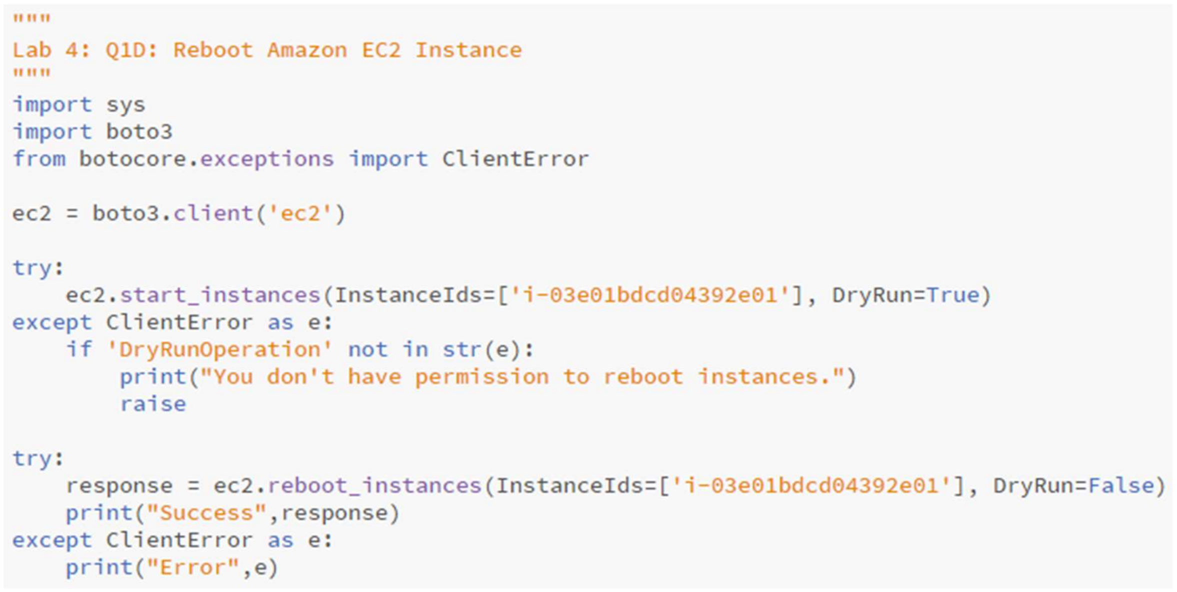
```
"""
Lab 4: Q1A: First Python Boto3 Activity
"""
import boto3
ec2 = boto3.client('ec2')
response = ec2.describe_instances()
print(response)
```



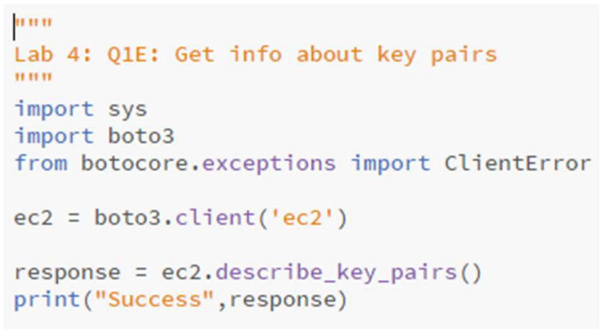
```python
"""
Lab 4: Q1B: Start and Stop Detailed Monitoring of an EC2 instance
"""
import sys
import boto3
ec2 = boto3.client('ec2')

if sys.argv[1] == 'ON':
    response = ec2.monitor_instances(InstanceIds=['i-09686cbf7bd65289d'])
else:
    response = ec2.unmonitor_instances(InstanceIds=['i-09686cbf7bd65289d'])

print(response)
```



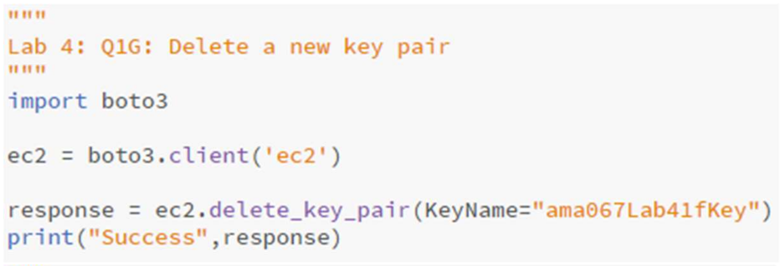
```python
"""
Lab 4: Q1C: Start and Stop Detailed Monitoring of an EC2 instance
"""
import sys
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

action = sys.argv[1].upper()

if action == 'ON':
    # Do a dryrun first to verify permissions
    try:
        ec2.start_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=True)
    except ClientError as e:
        if 'DryRunOperation' not in str(e):
            raise

    # Dry run succeeded, run start_instances again without dryrun
    try:
        response = ec2.start_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=False)
        print(response)
    except ClientError as e:
        print(e)

else:
     # Do a dryrun first to verify permissions
    try:
        ec2.stop_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=True)
    except ClientError as e:
        if 'DryRunOperation' not in str(e):
            raise

    # Dry run succeeded, run start_instances again without dryrun
    try:
        response = ec2.stop_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=False)
        print(response)
    except ClientError as e:
        print(e)
```



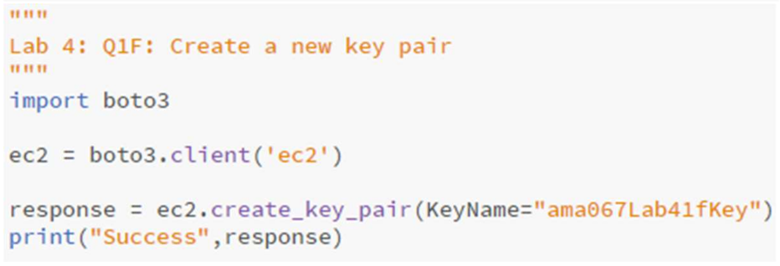
```
"""
Lab 4: Q1D: Reboot Amazon EC2 Instance
"""
import sys
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

try:
    ec2.start_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=True)
except ClientError as e:
    if 'DryRunOperation' not in str(e):
        print("You don't have permission to reboot instances.")
        raise

try:
    response = ec2.reboot_instances(InstanceIds=['i-03e01bdcd04392e01'], DryRun=False)
    print("Success",response)
except ClientError as e:
    print("Error",e)
```



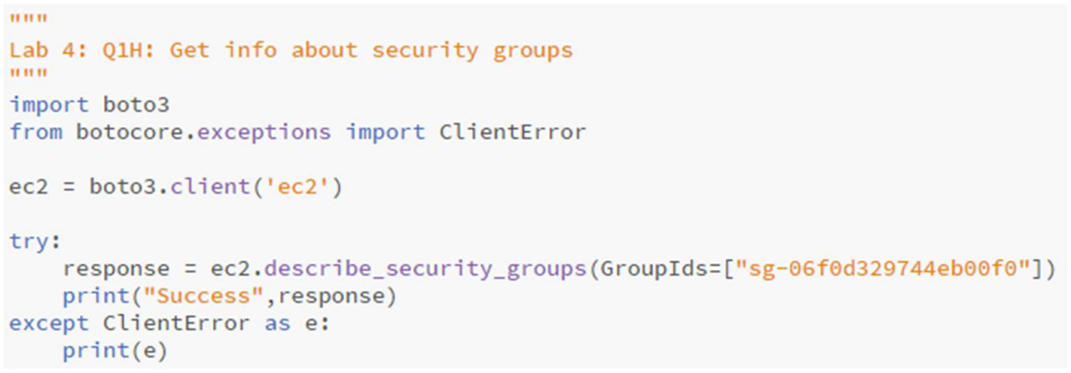
```
"""
Lab 4: Q1E: Get info about key pairs
"""
import sys
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

response = ec2.describe_key_pairs()
print("Success",response)
```



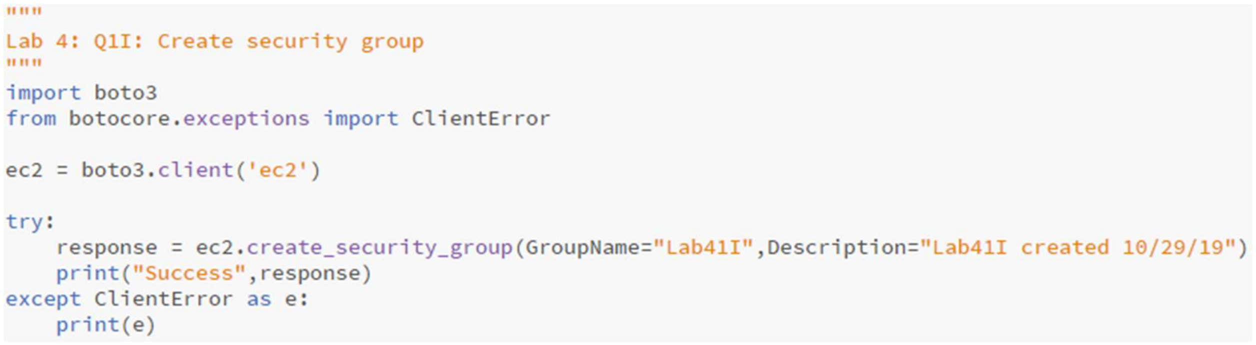
```
"""
Lab 4: Q1G: Delete a new key pair
"""
import boto3

ec2 = boto3.client('ec2')

response = ec2.delete_key_pair(KeyName="ama067Lab41fKey")
print("Success",response)
```



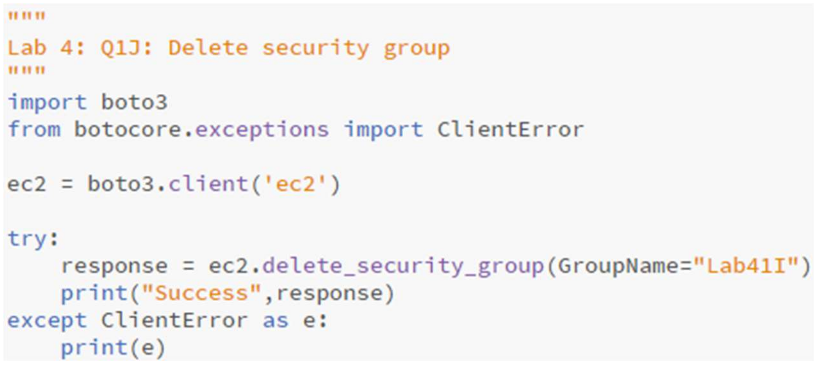
```
"""
Lab 4: Q1F: Create a new key pair
"""
import boto3

ec2 = boto3.client('ec2')

response = ec2.create_key_pair(KeyName="ama067Lab41fKey")
print("Success",response)
```



```
"""
Lab 4: Q1H: Get info about security groups
"""
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

try:
    response = ec2.describe_security_groups(GroupIds=["sg-06f0d329744eb00f0"])
    print("Success",response)
except ClientError as e:
    print(e)
```



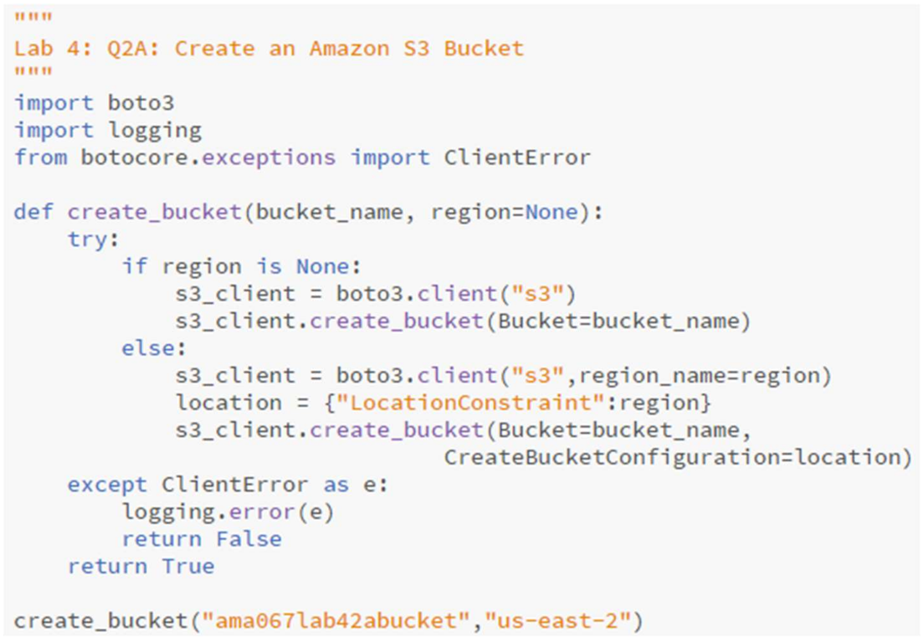
```
"""
Lab 4: Q1I: Create security group
"""
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

try:
    response = ec2.create_security_group(GroupName="Lab41I",Description="Lab41I created 10/29/19")
    print("Success",response)
except ClientError as e:
    print(e)
```



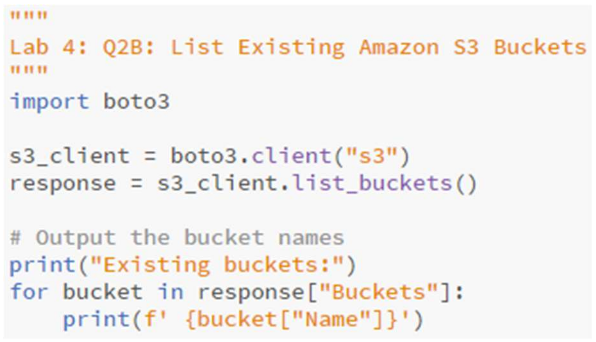
```
"""
Lab 4: Q1J: Delete security group
"""
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

try:
    response = ec2.delete_security_group(GroupName="Lab41I")
    print("Success",response)
except ClientError as e:
    print(e)
```

Problem 2:

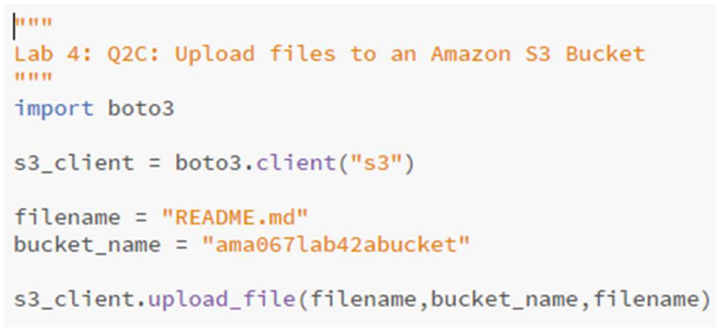
Each solution for this problem didn't require extra arguments to properly run. The target of each program was called for inside the program instead of needing to be read.



```python
"""
Lab 4: Q2A: Create an Amazon S3 Bucket
"""
import boto3
import logging
from botocore.exceptions import ClientError

def create_bucket(bucket_name, region=None):
    try:
        if region is None:
            s3_client = boto3.client("s3")
            s3_client.create_bucket(Bucket=bucket_name)
        else:
            s3_client = boto3.client("s3",region_name=region)
            location = {"LocationConstraint":region}
            s3_client.create_bucket(Bucket=bucket_name,
                                    CreateBucketConfiguration=location)
    except ClientError as e:
        logging.error(e)
        return False
    return True

create_bucket("ama067lab42abucket","us-east-2")
```



```python
"""
Lab 4: Q2B: List Existing Amazon S3 Buckets
"""
import boto3

s3_client = boto3.client("s3")
response = s3_client.list_buckets()

# Output the bucket names
print("Existing buckets:")
for bucket in response["Buckets"]:
    print(f' {bucket["Name"]}')
```



```python
"""
Lab 4: Q2C: Upload files to an Amazon S3 Bucket
"""
import boto3

s3_client = boto3.client("s3")

filename = "README.md"
bucket_name = "ama067lab42abucket"

s3_client.upload_file(filename,bucket_name,filename)
```

```python
"""
Lab 4: Q2D: Download files from an Amazon S3 Bucket
"""
import boto3
import botocore

s3 = boto3.resource("s3")

filename = "README2.md"
bucket_name = "ama067lab42abucket"
key = "README.md"

s3.Bucket(bucket_name).download_file(key,filename)
```

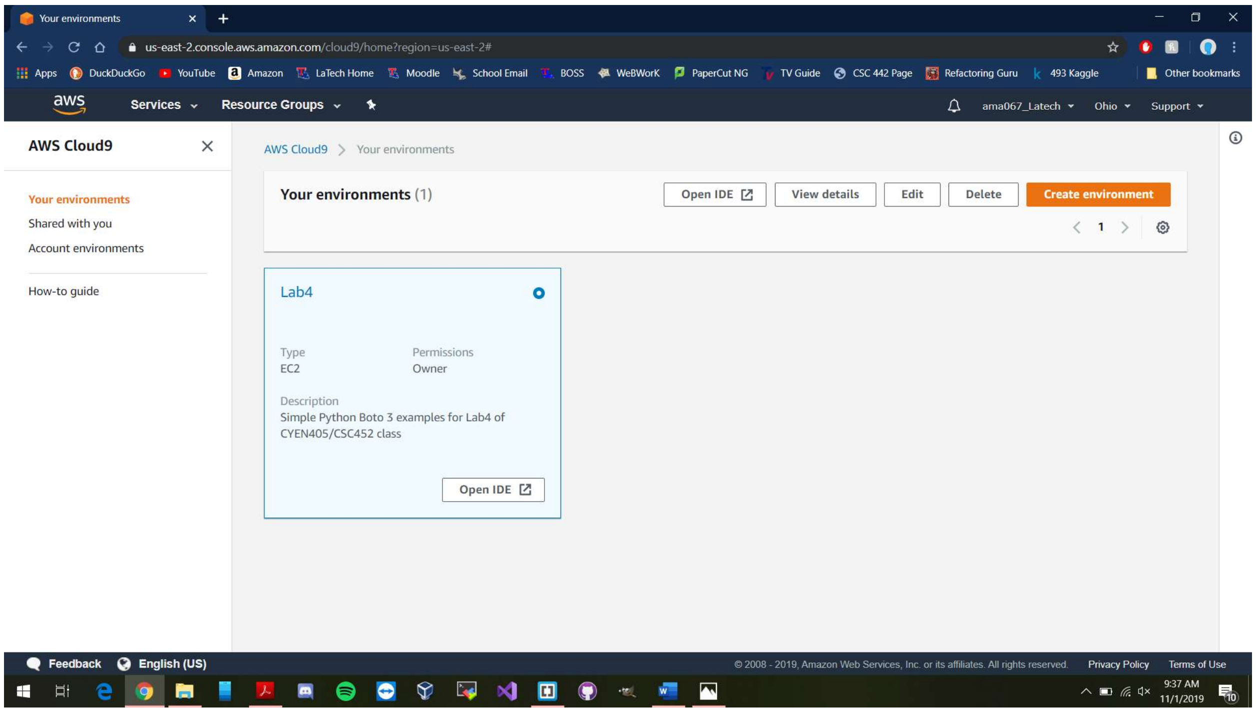
```python
"""
Lab 4: Q2E: Create Presigned URLs
"""
import boto3
import logging

bucket_name = "ama067lab42abucket"
key = "README.md"

s3 = boto3.client("s3")

response = s3.generate_presigned_url(ClientMethod="get_object",Params={
    "Bucket": bucket_name,
    "Key": key})

print(response)
```

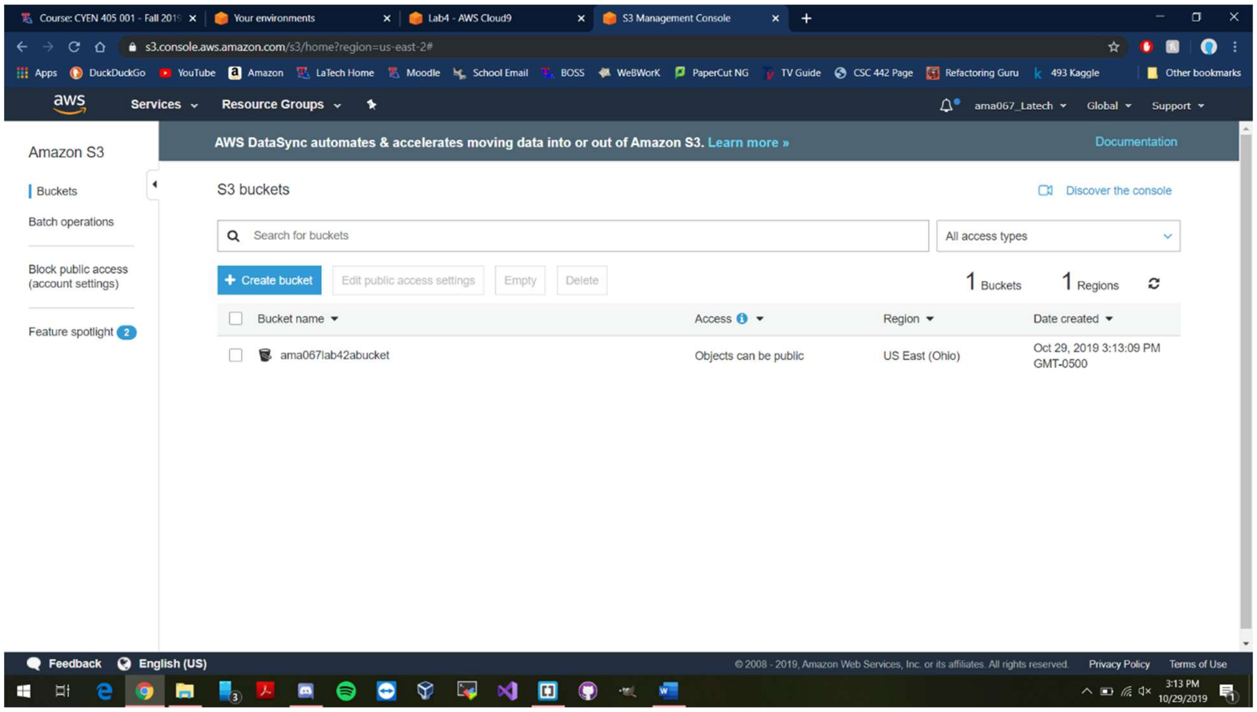
```python
"""
Lab 4: Q2F: Retrieve a Bucket Policy
"""
import boto3
import json

s3 = boto3.client("s3")

bucket_name = "ama067lab42abucket"

bucket_policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "AddPerm",
        "Effect": "Allow",
        "Principal": "*",
        "Action": ["s3:GetObject"],
        "Resource": "arn:aws:s3:::%s/*" % bucket_name
    }]
}

bucket_policy = json.dumps(bucket_policy)
s3.put_bucket_policy(Bucket=bucket_name,Policy=bucket_policy)

response = s3.get_bucket_policy(Bucket=bucket_name)

print(response["Policy"])
```

```
"""
Lab 4: Q2G: Retrieve a Bucket Policy
"""
import boto3
import json

s3 = boto3.client("s3")

bucket_name = "ama067lab42abucket"

response = s3.get_bucket_acl(Bucket=bucket_name)

print(response["Policy"])
```

Results

For both problems 1 and 2, the outputs of each parts' programs are included in the zip file and Github due to their length. Instead, below will be the Cloud 9 interface, the S3 interface showing buckets (Lab4_2A), the stored data into the bucket (Lab4_2C) and the README file downloaded from the bucket (Lab4_2D).



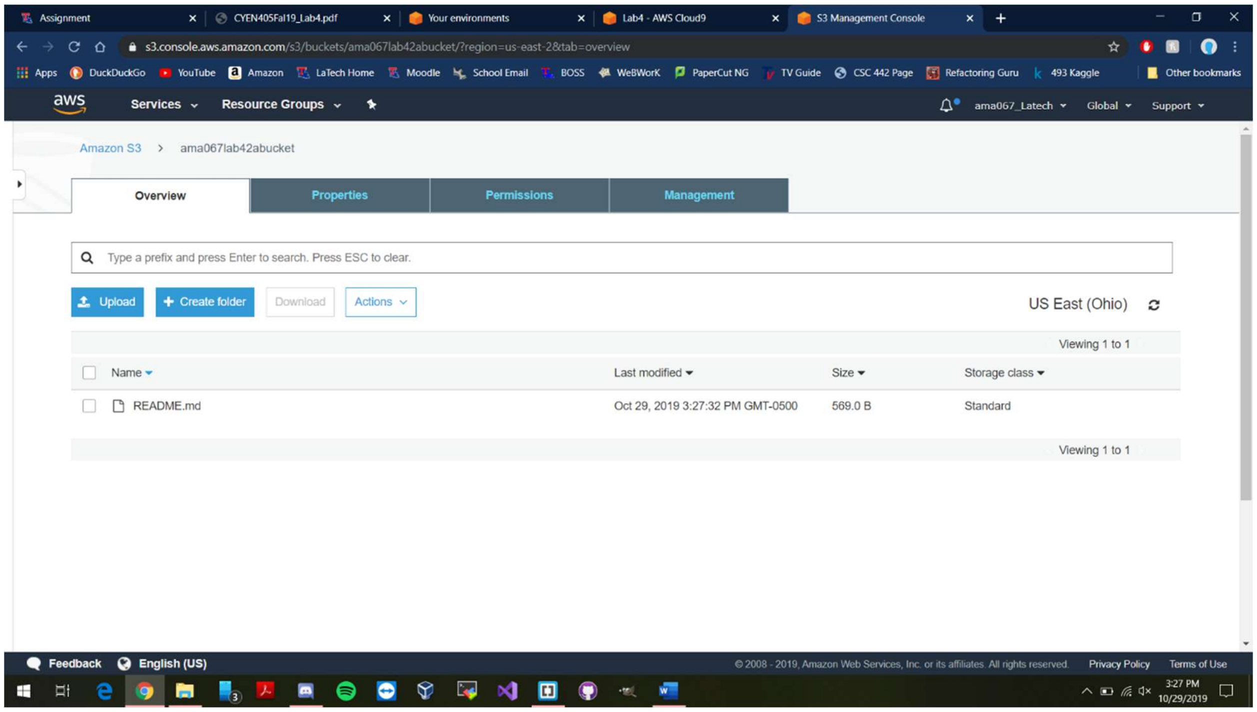


AWS DataSync automates & accelerates moving data into or out of Amazon S3. Learn more »

S3 buckets

1 Buckets 1 Regions

Bucket name	Access	Region	Date created
ama067lab42abucket	Objects can be public	US East (Ohio)	Oct 29, 2019 3:13:09 PM GMT-0500



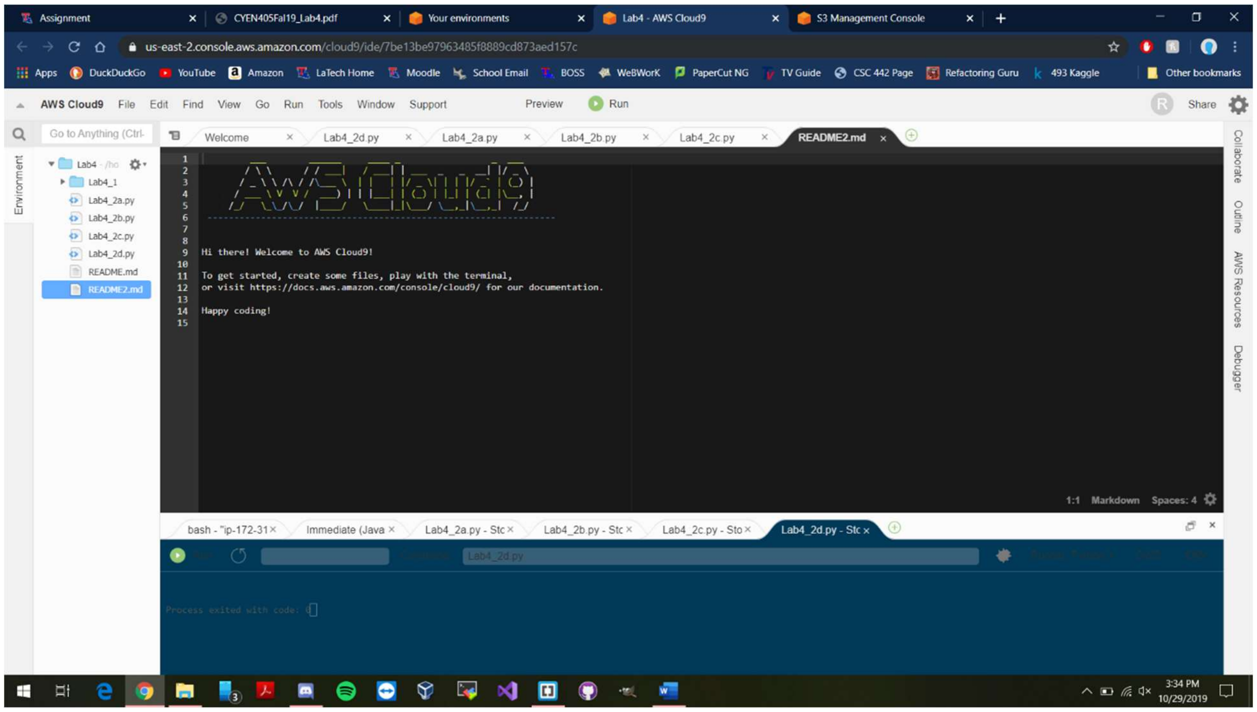
Amazon S3 > ama067lab42abucket

US East (Ohio)

Viewing 1 to 1

Name	Last modified	Size	Storage class
README.md	Oct 29, 2019 3:27:32 PM GMT-0500	569.0 B	Standard

Viewing 1 to 1



```
Hi there! Welcome to AWS Cloud9!

To get started, create some files, play with the terminal,
or visit https://docs.aws.amazon.com/console/cloud9/ for our documentation.

Happy coding!
```

```
Process exited with code: 0
```

Conclusion

Given the outcomes of the lab, it's clear to see the great uses of the Cloud 9 service provided by Amazon Web Services. The ability to manage and review information and instances of other services given through one interface opens the possibilities to near endless measures. Usage of this can lead to easier control over managed services without need to manually move to said services. Pairing this up with what was learned over the last lab shows again how usage of services like these given by Amazon can save time and money by offloading the work and data elsewhere.

Github Link

https://github.com/M3JPUV/CSC452_Labs.git