Lee Brunovsky

CSCI 6910 Cloud Computing and Security

CH7 *Python for Cloud*, **CH8** *Cloud Application Development in Python*: Lab Exercises

HW4

Due: July 19, 2021

Submitted: July 19, 2021

Chapter 7 (Lab Exercises): Provide screenshots for every step of the question

HW4CH7Q1:

1. Create a static website with one or two HTML pages and copy the pages and dependent files in an Amazon S3 bucket.

Py_Launch_EC2.py | index.html | bashScript.sh

```html
<!DOCTYPE html>
<html>
<body>

<img src="beer.jpg" style="float:right; margin:-10px 300px 0px 0px; width="900" height="400"/>
<h2>Do you like beer?</h2>
<p>What is your favorite style?</p>

<ol>
  <li><button>West Coast IPA</button></li>
  <br>
  <li><button>East Coast IPA (NEPA)</button></li>
  <br>
  <li><button>Milk, Coffee, or Chocolate Stout</button></li>
  <br>
  <li><button>Heffe or Whit</button></li>
  <br>
  <li><button>Peanut Butter Porter</button></li>
  <br>
  <li><button>Lager</button></li>
  <br>
  <li><button>Pilsner</button></li>
  <br>
  <li><button>Other</button></li>
</ol>

</body>
</html>
```

hw4ch7 Info

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
beer.jpg	jpg	July 15, 2021, 20:10:38 (UTC+09:00)	94.2 KB	Standard
index.html	html	July 17, 2021, 12:52:09 (UTC+09:00)	679.0 B	Standard

2. Create a startup script to install Apache server and copy the website files from the S3 to the EC2 instance.

```
#!/bin/bash
sudo apt-get update
sudo apt-get install apache2 -y
sudo apt  install awscli -y
cd /var/www/html
sudo aws configure set aws_access_key_id AKIA35TGKHOUG6MH65ER
sudo aws configure set aws_secret_access_key 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
sudo aws s3 cp s3://hw4ch7/index.html /var/www/html
sudo aws s3 cp s3://hw4ch7/beer.jpg /var/www/html
sudo /etc/init.d/apache2 restart
```

3. Create a Python program using boto to launch a m1.small Ubuntu instance. Supply the startup script you created in previous step while launching a new instance from the program. Use a security group with port 80 open.

Source code amended from (Bahga & Madisetti, 2014).

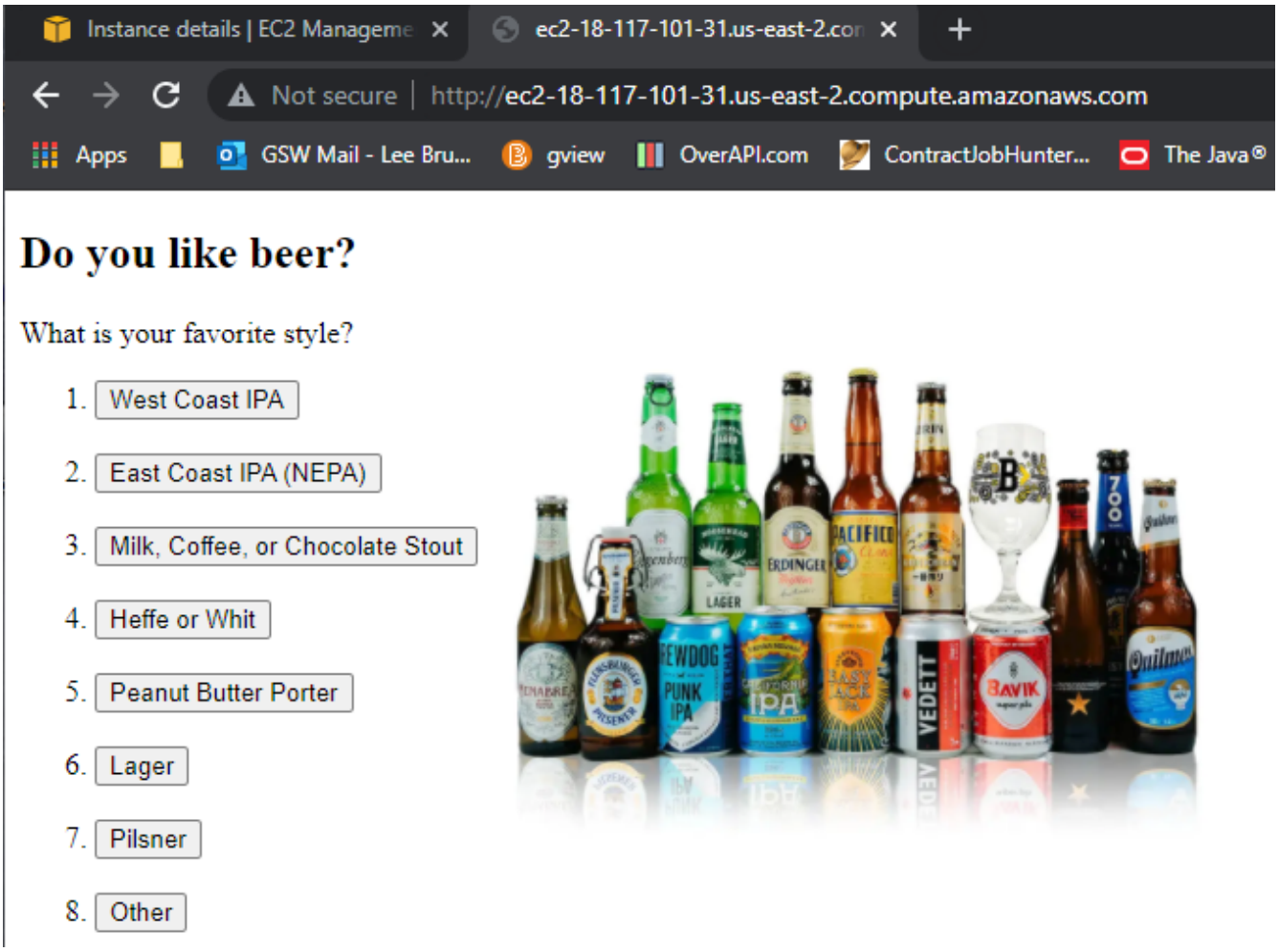
```
import boto.ec2

user_data_script = """#!/bin/bash
sudo apt-get update
sudo apt-get install apache2 -y
sudo apt  install awscli -y
cd /var/www/html
sudo aws configure set aws_access_key_id AKIA35TGKHOUG6MH65ER
sudo aws configure set aws_secret_access_key
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
sudo aws s3 cp s3://hw4ch7/index.html /var/www/html
sudo aws s3 cp s3://hw4ch7/beer.jpg /var/www/html
sudo /etc/init.d/apache2 restart"""

ACCESS_KEY="AKIA35TGKHOUG6MH65ER"
SECRET_KEY="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"

REGION="us-east-2"
AMI_ID ="ami-00399ec92321828f5"
EC2_KEY_HANDLE = "HW4, CH7"
INSTANCE_TYPE="t2.micro"
SECGROUP_HANDLE="launch-wizard-1"    #Use a security group with SSH port 22, HTTP port 80
and HTTPs port 443 open.

print("Connecting to EC2")
```

```
conn = boto.ec2.connect_to_region(REGION,
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY)

print("Launching instance with AMI_ID %s, with keypair %s, instance type %s, security group
%s " %(AMI_ID,EC2_KEY_HANDLE,INSTANCE_TYPE,SECGROUP_HANDLE))

reservation = conn.run_instances(image_id=AMI_ID,
    key_name=EC2_KEY_HANDLE,
    instance_type=INSTANCE_TYPE,
    security_groups=[SECGROUP_HANDLE,],
    user_data=user_data_script)

instance = reservation.instances[0]

print("Waiting for instance to be up and running")

status = instance.update()
while status == 'pending':
    sleep(10)
    status = instance.update()

if status == 'running':
    print("\n instance is now running. Instance details are:")
    print("Instance Size: " + str(instance.instance_type))
    print("Instance State: " + str(instance.state))
    print("Instance Launch Time: " + str(instance.launch_time))
    print("Instance Public DNS: " + str(instance.public_dns_name))
    print("Instance Private DNS: " + str(instance.private_dns_name))
    print("Instance IP: " + str(instance.ip_address))
    print("Instance Private IP: " + str(instance.private_ip_address))
```

4. Your program should check the status of the newly launched instance and return the public DNS address of the instance when the status changes to ‘running’.

```
PS C:\Users\leebr\Desktop\HW4>  c:; cd 'c:\Users\leebr\Desktop\HW4'; & 'C:\Python39\python.exe' 'c:\Users\leebr\.vscode\extensions\
W4\Py_Launch_EC2_copy.py'
Connecting to EC2
Launching instance with AMI_ID ami-00399ec92321828f5, with keypair HW4, CH7, instance type t2.micro, security group launch-wizard-1
Waiting for instance to be up and running

 instance is now running. Instance details are:
Instance Size: t2.micro
Instance State: running
Instance Launch Time: 2021-07-17T05:17:23.000Z
Instance Public DNS: ec2-18-117-101-31.us-east-2.compute.amazonaws.com
Instance Private DNS: ip-172-31-3-101.us-east-2.compute.internal
Instance IP: 18.117.101.31
Instance Private IP: 172.31.3.101
```

5. Open the public DNS of the newly launched instance in a browser and verify if the static website works.



HW4CH7Q2: Note amended instructions deem only step 3 necessary: *For Q2 Python program you don't have to scale up and down the policies, supply the startup script and use a security group. Just write the code for Amazon autoscaling group using boto.*

Steps 1 & 2 not necessary per amended and repeated same as Q1

3. Create a Python program using boto that creates an Amazon AutoScaling group. Define scale up and scale down policies and the corresponding CloudWatch alarms. Supply the startup script you created in previous step while launching a new instance from the program. Use a security group with port 80 open.

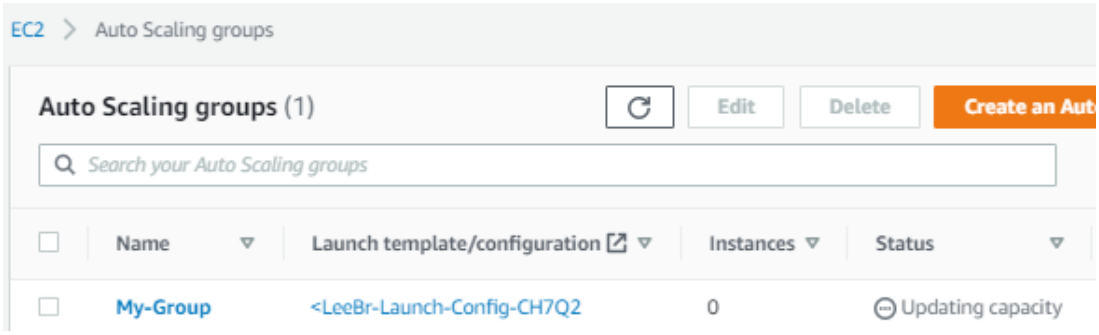
Source code amended from (Bahga & Madisetti, 2014).

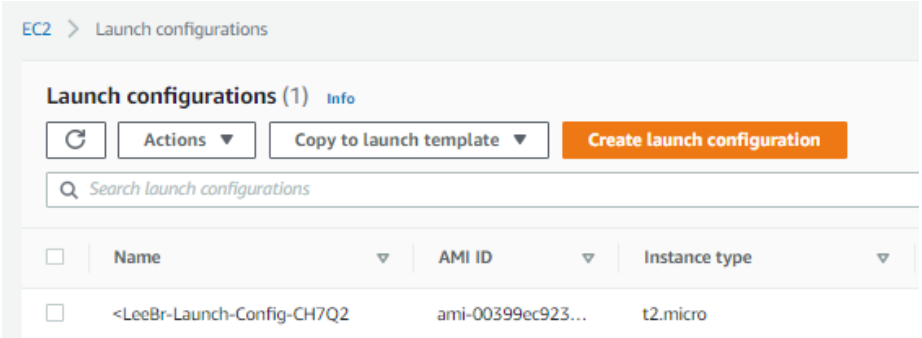
```
import boto.ec2.autoscale
from boto.ec2.autoscale import LaunchConfiguration
from boto.ec2.autoscale import AutoScalingGroup
from boto.ec2.cloudwatch import MetricAlarm
from boto.ec2.autoscale import ScalingPolicy
import boto.ec2.cloudwatch

user_data_script = """#!/bin/bash
sudo apt-get update
sudo apt-get install apache2 -y
sudo apt  install awscli -y
cd /var/www/html
sudo aws configure set aws_access_key_id AKIA35TGKHOUG6MH65ER
sudo aws configure set aws_secret_access_key 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
sudo aws s3 cp s3://hw4ch7/index.html /var/www/html
sudo aws s3 cp s3://hw4ch7/beer.jpg /var/www/html
sudo /etc/init.d/apache2 restart"""

ACCESS_KEY="AKIA35TGKHOUG6MH65ER"
SECRET_KEY="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"

REGION="us-east-2"
AMI_ID ="ami-00399ec92321828f5"
EC2_KEY_HANDLE = "HW4, CH7"
INSTANCE_TYPE="t2.micro"
SECGROUP_HANDLE="launch-wizard-1"    #Use a security group with SSH port 22, HTTP port 80 and
HTTPs port 443 open.

print("Connecting to Autoscaling Service")

conn = boto.ec2.autoscale.connect_to_region(REGION,
   aws_access_key_id=ACCESS_KEY,
   aws_secret_access_key=SECRET_KEY)

print("Creating launch configuration")

lc = LaunchConfiguration(name='<LeeBr-Launch-Config-CH7Q2',
   image_id=AMI_ID,
   key_name=EC2_KEY_HANDLE,
   instance_type=INSTANCE_TYPE,
   security_groups=[SECGROUP_HANDLE,],
   user_data=user_data_script)

conn.create_launch_configuration(lc)

print("Creating auto-scaling group")
```

```
ag = AutoScalingGroup(group_name='My-Group',
   availability_zones=['us-east-2b'],
   launch_config=lc,min_size=1, max_size=2,
   connection=conn)

conn.create_auto_scaling_group(ag)

print("Creating auto-scaling group")

scale_up_policy = ScalingPolicy(name='scale_up',
   adjustment_type='ChangeInCapacity',
   as_name='My-Group',
   scaling_adjustment=1,
   cooldown=180)

scale_down_policy = ScalingPolicy(name='scale_down',
   adjustment_type='ChangeInCapacity',
   as_name='My-Group',
   scaling_adjustment=-1,
   cooldown=180)

conn.create_scaling_policy(scale_up_policy)
conn.create_scaling_policy(scale_down_policy)

scale_up_policy = conn.get_all_policies(as_group='My-Group',
policy_names=['scale_up'])[0]
scale_down_policy = conn.get_all_policies(as_group='My-Group',
policy_names=['scale_down'])[0]

print("Connecting to CloudWatch")

cloudwatch = boto.ec2.cloudwatch.connect_to_region(REGION,
   aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY)

alarm_dimensions = {"AutoScalingGroupName":'My-Group'}

print("Creating scale-up alarm")

scale_up_alarm = MetricAlarm(
   name='scale_up_on_cpu',  namespace='AWS/EC2',
   metric='CPUUtilization', statistic='Average',
   comparison='>', threshold='70',
   period='60', evaluation_periods=2,
   alarm_actions=[scale_up_policy.policy_arn],
```

```
    dimensions=alarm_dimensions)

cloudwatch.create_alarm(scale_up_alarm)

print("Creating scale-down alarm")

scale_down_alarm = MetricAlarm(
    name='scale_down_on_cpu',  namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='>', threshold='50',
    period='60', evaluation_periods=2,
    alarm_actions=[scale_down_policy.policy_arn],
    dimensions=alarm_dimensions)

cloudwatch.create_alarm(scale_down_alarm)
print("Done!")
```

```
PS C:\Users\leebr\Desktop\HW4>  c:; cd 'c:\Users\leebr\Desktop\HW4'; & '
42' '--' 'c:\Users\leebr\Desktop\HW4\Py_Launch_EC2_AutoScaling_Group.py'
Connecting to Autoscaling Service
Creating launch configuration
Creating auto-scaling group
Creating auto-scaling group
Connecting to CloudWatch
Creating scale-up alarm
Creating scale-down alarm
Done!
```



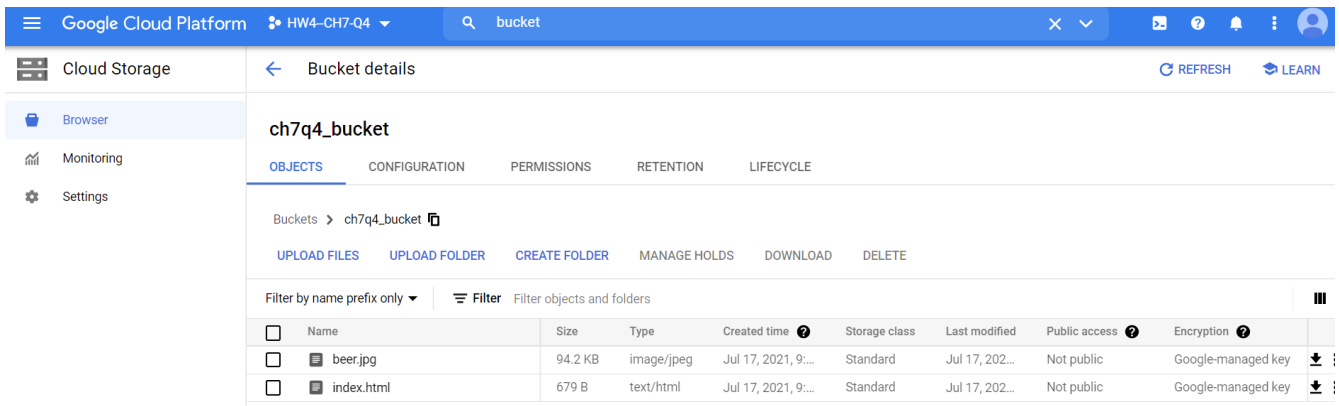


Step 4 not necessary per amended instruction, but I checked and it works :)

HW4CH7Q4:

1. Create a static website with one or two HTML pages and copy the pages and dependent files in Google Cloud Storage.

Same HTML page and image as Q1



2. Create a startup script to install Apache server and copy the website files from GCS to the Google Compute Engine instance.

gcloud auth application-default login

```
    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=764086051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googl
oauth%3A2.0%3Aoob&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fc
Fauth%2Faccounts.reauth&state=9PFts0opv05wIPyT5pTB4M5ogpsHiQ&prompt=consent&access_type=offline&code_challenge=W7KXvGCqg07rajZZAwFy
56

Enter verification code: 4/1AX4XfWjdXwx6z0NOW3PyhMjTaBbipNweEys5Fb3YA4FgFMK9gnSGlYQ-sQU

Credentials saved to file: [/tmp/tmp.t5DjLUy5K1/application_default_credentials.json]

These credentials will be used by any library that requests Application Default Credentials (ADC).

Quota project "hw4-ch7-q4" was added to ADC which can be used by Google client libraries for billing and quota. Note that some serv
rce.
```

-Enable the Cloud Storage API.

gcloud services enable pubsub.googleapis.com

```
leebrunovsky@cloudshell:~ (hw4-ch7-q4)$ gcloud services enable pubsub.googleapis.com
```

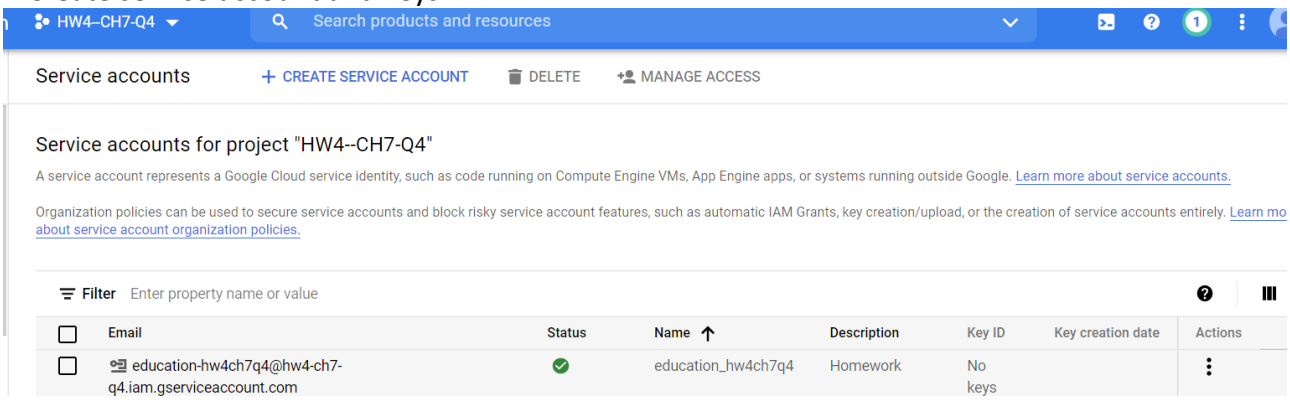
```
#!/bin/bash
sudo apt-get update
sudo apt-get install apache2 -y
sudo apt  install awscli -y
cd /var/www/html
gsutil -m cp -r dir gs://my-bucket
sudo /etc/init.d/apache2 restart
```

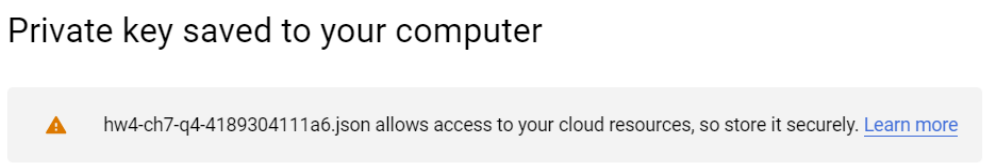
3. Create a Python programto launch a Ubuntu instance. Supply the startup script you created in previous step while launching a new instance from the program.

Install the google-api-python-client library if not already (it was)

$ pip install --upgrade google-api-python-client

Create service account and keys





set credentials PATH


```
{
  "type": "service_account",
  "project_id": "hw4-ch7-q4",
  "private_key_id": "4189304111a60d1b788b9579d7bf07da46a95009",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCfLWaSR9rTtXif\nr7/Mvw6
  "client_email": "education-hw4ch7q4@hw4-ch7-q4.iam.gserviceaccount.com",
  "client_id": "116485311935322284531",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/education-hw4ch7q4%40hw4-ch7-q4.iam.gservic
}
```


```
see https://cloud.google.com/do> set GOOGLE_APPLICATION_CREDENTIALS "C:\Users\leebr\Desktop\HW4\Q4\hw4-ch7-q4-41
89304111a6.json"r\Desktop\HW4\Q4>
```

Code base from google cloud compute docs python-guide [2]

```
import argparse
import os
import googleapiclient
from googleapiclient.discovery import build
from oauth2client.client import GoogleCredentials
from argparse import ArgumentParser, RawDescriptionHelpFormatter
from google.oauth2 import service_account

client= service_account.Credentials.from_service_account_file("C:/Users/leebr/Desktop/HW4/Q4/hw4-
    ch7-q4-4189304111a6.json")

def list_instances(compute, project, zone):
  result = compute.instances().list(project=project, zone=zone).execute()
  return result['items'] if 'items' in result else None

def create_instance(compute, project, zone, name, bucket):
  # Get the latest Debian Jessie image.
  image_response = compute.images().getFromFamily(
    project='debian-cloud', family='debian-9').execute()
  source_disk_image = image_response['selfLink']

  # Configure the machine
  machine_type = "zones/%s/machineTypes/n1-standard-1" % zone
  startup_script = open(
    os.path.join(
      os.path.dirname(__file__), 'startup-script.sh'), 'r').read()
  image_url = "http://storage.googleapis.com/gce-demo-input/photo.jpg"
  image_caption = "Ready for dessert?"

  config = {
    'name': name,
    'machineType': machine_type,

    # Specify the boot disk and the image to use as a source.
    'disks': [
      {
        'boot': True,
        'autoDelete': True,
        'initializeParams': {
          'sourceImage': source_disk_image,
        }
      }
    ],
```

```
        # Specify a network interface with NAT to access the public
        # internet.
        'networkInterfaces': [{
          'network': 'global/networks/default',
          'accessConfigs': [
            {'type': 'ONE_TO_ONE_NAT', 'name': 'External NAT'}
          ]
        }],

        # Allow the instance to access cloud storage and logging.
        'serviceAccounts': [{
          'email': 'default',
          'scopes': [
            'https://www.googleapis.com/auth/devstorage.read_write',
            'https://www.googleapis.com/auth/logging.write'
          ]
        }],

        # Metadata is readable from the instance and allows you to
        # pass configuration from deployment scripts to instances.
        'metadata': {
          'items': [{
            # Startup script is automatically executed by the
            # instance upon startup.
            'key': 'startup-script',
            'value': startup_script
          }, {
            'key': 'url',
            'value': image_url
          }, {
            'key': 'text',
            'value': image_caption
          }, {
            'key': 'bucket',
            'value': bucket
          }]
        }
    }

    return compute.instances().insert(
        project=project,
        zone=zone,
        body=config).execute()

def main(project, bucket, zone, instance_name, wait=True):
```

```
    compute = googleapiclient.discovery.build('compute', 'v1')

    print('Creating instance.')

    operation = create_instance(compute, project, zone, instance_name, bucket)
    wait_for_operation(compute, project, zone, operation['name'])

    instances = list_instances(compute, project, zone)

    print('Instances in project %s and zone %s:' % (project, zone))
    for instance in instances:
        print(' - ' + instance['name'])

    print("""
Instance created.
It will take a minute or two for the instance to complete work.
Check this URL: http://storage.googleapis.com/{}/output.png
Once the image is uploaded press enter to delete the instance.
""".format(bucket))

    if wait:
        input()

    print('Deleting instance.')

    operation = delete_instance(compute, project, zone, instance_name)
    wait_for_operation(compute, project, zone, operation['name'])

if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description=__doc__,
        formatter_class=argparse.RawDescriptionHelpFormatter)
    parser.add_argument('project_id', help='Your Google Cloud project ID.')
    parser.add_argument(
        'bucket_name', help='Your Google Cloud Storage bucket name.')
    parser.add_argument(
        '--zone',
        default='us-central1-f',
        help='Compute Engine zone to deploy to.')
    parser.add_argument(
        '--name', default='demo-instance', help='New instance name.')

    args = parser.parse_args()

    main(args.project_id, args.bucket_name, args.zone, args.name)
```

4. Your program should check the status of the newly launched instance and return when the status changes to ‘running’.

I was unable to complete this step on account of not getting past the credential validation. I even tried hard coding the file. I also attempted with the class text code, which took a lot of time to reproduce, and found it to be very deprecated compared to the Google documentation. I can provide it if desired.

RUN : python create_instance.py --name hw4ch7q4 --zone us-central1-f hw4-ch7-q4 ch7q4_bucket

```
  File "C:\Python39\lib\site-packages\google\auth\_default.py", line 483, in default
    raise exceptions.DefaultCredentialsError(_HELP_MESSAGE)
google.auth.exceptions.DefaultCredentialsError: Could not automatically determine credentials. Please set GOOGLE_
APPLICATION_CREDENTIALS or explicitly create credentials and re-run the application. For more information, please
 see https://cloud.google.com/docs/authentication/getting-started
```

Ch8_Q4 extra credit Extend the Social Media Analytics app described in section 7.4 to display top keywords and top users (users who tweet the most in a day).

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
   doc_id, content = line.splity('')

   words = content.split()
   for word in words:
      print ('%s%s' % (word, doc_id))
      userID = []
      userID.append(doc_id)
      keywWords = []
      keyWords.append(word)

      counts = []
      for uID in userID:
         if uID not in counts:
            count = 0
               if c == uID:                    # Test if user ID equal to the current user ID
                  count += 1                    # Increment count
         counts.append(counts)

      idCountDict = {}                          # Create Dict of counts / unique to sort
      for key, val in zip(userID, counts):
         wordCountDict.setdefault(key, []).append(val)
      return idCountDict

     # print top 10 user ID
```

```
idCountDict_Sorted = sorted(idCountDict,
        key=idCountDict.get, reverse=True)     #Sort by dict key, descending
for i in idCountDict_Sorted[:10]:              # Print top 10
    print(i, idCountDict[i])

Wcounts = []
for w in words:
    if w not in Wcounts:
        count = 0
            if c == w:                         # Test if word equal to the current unique word
                count += 1                     # Increment count
    counts.append(Wcounts)

wCountDict = {}                                # Create Dict of counts / unique to sort
for key, val in zip(words, wcounts):
    wordCountDict.setdefault(key, []).append(val)
return wCountDict

# print top 10 key words
wCountDict_Sorted = sorted(wCountDict,
        key=wCountDict.get, reverse=True)     #Sort by dict key, descending
for i in wCountDict_Sorted[:10]:              # Print top 10
    print(i, wCountDict[i])
```

References

Bahga, A., & Madisetti, V. (2014). *Cloud Computing: A Hands-On Approach*. Self published, Arshdeep Bahga & Vijay Madisetti.

Google. (n.d.). *Using the Cloud Client Libraries for Python*. Google. https://cloud.google.com/compute/docs/tutorials/python-guide.