EEL 6761: Cloud Computer Sys Apps Spring 2024 Instructor: J. Fortes
Homework 3 Assigned: 1/11/2024 Due: 3/19/2024 @ 5 Pm (use UF E-learning - Canvas)

Deploying an Elastic Cloud Service

Preamble:

In this homework, you will deploy an elastic cloud service as described in Section 5.2 of Cloud Computing for Science and Engineering by Ian Foster and Dennis B. Gannon. The cloud service will provide users with an integrated development environment (IDE) through which they can issue jobs to a backend Apache Spark engine. As more users connect and computational demand increases, the cloud will allocate more compute resources. Conversely, excess compute resources will be deallocated as computational demand falls.

Your cloud will be composed of virtual machine instances that provide exactly one standalone Spark cluster* and one Jupyter Notebook server. The Jupyter Notebook server runs a web application (that is, the IDE) for users to connect to. The IDE supports many languages, but for this homework, you will use Python. The Jupyter Notebook server will communicate with the Spark engine using PySpark, an interface that allows Spark jobs to be issued using Python APIs. You will also set up a load balancer to route users to different instances in your cloud according to their computational loads.

You will use Amazon Web Services (AWS) to implement each of the components of your cloud. The Elastic Compute Cloud (EC2) module of AWS provides all the capabilities needed to complete this homework, including requesting resources for new instances, connecting to instances, defining networking rules, enabling elasticity, and setting up load balancing. To avoid paying for these services, you will need to sign up for AWS Free Tier by creating an account at https://aws.amazon.com/free. AWS Free Tier provides a limited amount of free resources each month for 12 months. If you have signed up in the past and your 12 months have expired, please sign up again using a different email address. Note that you will need to provide credit card information to Amazon in order to sign up, but as long as you stay within the free tier limits, you should not incur any charges. If you use only AWS EC2 services exactly as instructed in the tasks below, you should not exceed the free tier limits.

Important notes (Read carefully before you start the assignment):

1. This homework requires basic knowledge of the Linux command line and file system. If you are not comfortable with either of these, see https://ubuntu.com/tutorials/command-line-for-beginners before attempting the tasks below.
2. As part of this assignment, you might incur charges by AWS. However, if you follow the instructions of this assignment precisely, you should not incur any charges.
3. Hints are provided throughout the assignment to help you finish the assignment.
4. For task 1.B, if you are unable to access the load balancer or if there is an issue with the auto scaling, it is better for you to delete all the templates including the Amazon Machine Image (AMI) and re-create everything. Make sure you follow the instructions provided especially when you create a template and when you create the auto scaling group as part of task 1.B.

For this homework, **do not configure Spark to use "cluster mode". In "standalone mode", the Spark cluster is still called a "cluster" because it uses several independent processes in order to simulate a distributed cluster, but all the processes are located on the same virtual machine. In "cluster mode", the processes would be distributed across different virtual machines.*

Tasks:

There are 2 tasks in this homework, described below:

Task 1 – Deploying an Elastic Cloud Service on AWS
This task has two parts. In 1.A, you will create an EC2 instance running a Jupyter Notebook server alongside a standalone Spark cluster. Once completed, you will be able to connect to the Jupyter Notebook server and issue a Spark job that computes the value of *pi* within a specified degree of precision. In 1.B, you will use your instance from 1.A to deploy an elastic cloud service.

1.A – Deploy a single Spark+Jupyter service on AWS: In this task, you will create an EC2 instance running a Jupyter Notebook server alongside a standalone Spark cluster. Once completed, you will be able to connect to the Jupyter Notebook server and issue a Spark job that computes the value of *pi* within a specified degree of precision.

It is recommend that you first complete Step 2 (below) locally on your own computer before attempting it on an EC2 instance, as 1) it will not waste AWS Free Tier resources and 2) EC2 introduces networking obstacles that may make it more difficult to debug your setup. You will need an Ubuntu 22.04 environment with Python 3.9 installed. If your computer runs Windows, MacOS, or anything other than Ubuntu, consider using a virtual machine created from an Ubuntu 22.04 image as described in https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview.

Step 1: Configure an EC2 instance on AWS:
You will first need to launch an EC2 instance, which is a virtual machine hosted on Amazon servers. This can be done using the EC2 console at https://console.aws.amazon.com/ec2. See https://aws.amazon.com/ec2/getting-started/ for instructions on launching, configuring, and connecting to instances. Every instance must be launched from an Amazon Machine Image (AMI), which determines the operating system and software that will be pre-installed on the instance. We recommend selecting "Ubuntu Server 22.04 LTS (HVM), SSD Volume Type – 64-bit (x86)". When asked to choose an instance type (which determines the resources that will be allocated to the instance), choose "t2.micro", as this option is included in the free tier. After setting the AMI and instance type, the remaining configuration defaults are acceptable but may be modified at your discretion.

Step 2: Install Spark, PySpark, and Jupyter notebook:
1. Connect to your new EC2 instance from step 1. For assistance, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html. Note that, for the Ubuntu AMI, the default username is "ubuntu".
   Note: Update the package information for all the sources before proceeding to the next step (Hint: *sudo apt-get update)*.

2. Install Java 11 or 17. For assistance, see https://ubuntu.com/tutorials/install-jre. Review https://spark.apache.org/docs/latest/ for more information about which versions of Java are supported by Spark.

3. Install Spark 3.5 by downloading Spark (see https://spark.apache.org/downloads.html), unzipping the downloaded archive, and adding the contained "bin" and "sbin" folders to your system path. PySpark is included in the Spark archive within the "bin" folder. Once that folder is added to the system path, PySpark can be started using the `pyspark` command.
   Hint: Check 'wget' command for downloading Spark 3.5 into the EC2 instance.

4. Install Jupyter notebook. For assistance, see the "Installing Jupyter with pip" section in https://docs.jupyter.org/en/latest/install/notebook-classic.html. This page suggests installing the Anaconda distribution for Python, but this is not necessary, as Python 3.8 or later comes pre-installed in the Ubuntu AMI.
   Hint: pip3 is not pre-installed. Install python3-pip before attempting to install Jupyter notebook.

5. Configure PySpark to use Jupyter notebook as its driver (rather than the default, a Python shell). This can be done by setting the following environment variables:

   ```
   PYSPARK_DRIVER_PYTHON="jupyter"
   PYSPARK_DRIVER_PYTHON_OPTS="notebook --ip=* --NotebookApp.token=''"
   ```

   Note 1: After this, you must logout and log back in for the changes and the jupyter notebook installation to be visible globally.

   Note 2: To protect your jupyter notebook server, set a password using the following command

   ```
   jupyter notebook password
   ```

   See https://jupyter-notebook.readthedocs.io/en/stable/config.html for the available Jupyter Notebook command line options. Be sure to set a password that you do not plan to use for your personal accounts, as you will be asked to provide it by Canvas email sent to the instructor and TAs (when composing a message in Canvas, you can select TAs from the contacts icon next to the To box). Do **not** leave the password blank, as this would allow anyone in the world to connect to your server and execute arbitrary Python code under your identity.

6. Using the EC2 console, configure your instance's security group to allow incoming TCP traffic on port 8888 (the default for Jupyter notebooks). For assistance, see https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html.

7. Run the `pyspark` command. This will start a Jupyter notebook server that listens on port 8888. Connect to the server by entering http://YOUR-INSTANCE'S-PUBLIC-IP:8888 into a web browser. Your instance's public IP address is shown in the EC2 console.

8. In Jupyter, create a notebook and run the following script:

   ```
   import math
   from pyspark.sql import SparkSession

   spark = SparkSession.builder.getOrCreate()
   sc = spark.sparkContext

   x = 0
   i = 0
   while True:
       n = 10000
       k_values = [i + x for x in range(1, n+1)]
       num_partitions = 100
       dat = sc.parallelize(k_values, num_partitions)
       sqrs = dat.map(lambda k: (1.0/k)**2)
   ```

```
x += sqrs.reduce(lambda a,b: a+b)
pi = math.sqrt(x * 6)
i += n
print(pi)
```

The code above approximates π as described in Section 2.8.1 of Cloud Computing for Science and Engineering by Ian Foster and Dennis B. Gannon. See the textbook for more information.

1.B – Deploy an elastic cloud service on AWS: In this part, you will use your instance from 1.A to deploy an elastic cloud service. This will require first modifying your instance to automatically start the Jupyter and Spark servers on startup, so that new servers may be started automatically by simply creating new copies of your instance. Copying an instance requires a machine image; that is, a snapshot of an existing instance from which new instances can be initialized. With the machine image, you will create an autoscaling group that automatically creates new instances in response to rising computational demand. You will then create a load balancer with a fixed IP address to forward user HTTP(S) requests to the dynamically created Jupyter servers running on the instances in your autoscaling group. When finished, you will have created a cloud service that allows users to write and execute code in a Spark-enabled Jupyter Notebook environment.

Step 1: Create an Amazon Machine Image:

1. Configure your EC2 instance to automatically run the `pyspark` command after booting up. This can be done using the systemd (http://systemd.io/), a suite of tools that enable automation by coordinating and executing user-defined "services". Systemd comes pre-installed with the Ubuntu AMI. Create a file at `/etc/systemd/system/pyspark.service` with the following contents in INI format, replacing the file paths with your own as needed:

```
# [Unit] tells systemd *what* this service is and *when* to start it
[Unit]
Description=PySpark with Jupyter

# Don't start this service until the machine has Internet access
After=network.target
After=systemd-user-sessions.service
After=network-online.target

# [Service] tells systemd *how* to start this service
[Service]
User=ubuntu
#Should usually be the same as User
Group=ubuntu
# Wherever you want Jupyter to save user files
WorkingDirectory=/home/ubuntu/
# Run the pyspark command to start this service
ExecStart=/PATH/TO/pyspark

# Services do not inherit the user's environment variables (including
# PATH!); they must be explicitly defined
```

```
Environment=PYSPARK_DRIVER_PYTHON="/PATH/TO/jupyter"
Environment=PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=*
--NotebookApp.token="""

# [Install] tells systemd *why* it should start this service by
# defining a dependency chain with other services
[Install]
WantedBy=multi-user.target
```

2. Note that the systemd service does not inherit user environment variables, so they need to be set manually using `Environment=` statements. If you don't know where your `jupyter` and `pyspark` executables are located, you can find them using the "`which`" command (for usage instructions, run "`man which`" in your terminal).

3. Enable the service by running "`sudo systemctl enable pyspark`". Note that the `pyspark` option here refers to your `pyspark.service` file, not the executable file within Spark's "bin" folder. Test your setup by using your local web browser to connect to http://YOUR-INSTANCE'S-PUBLIC-IP:8888. If this fails, examine the service logs with "`systemctl status pyspark`" or "`journalctl -u pyspark`". For a list of systemd commands that may help with debugging, see https://access.redhat.com/sites/default/files/attachments/12052018_systemd_6.pdf. Reboot the EC2 instance and verify that service automatically starts by connecting to the Jupyter interface at http://YOUR-INSTANCE'S-PUBLIC-IP:8888, as before.

   Note: If you do not remember your jupyter notebook password, do not proceed further until you have reset the password. Use jupyter notebook password command to reset your password before proceeding further.

4. Using the EC2 console, create the following:
   a. An AMI from your instance. See https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/tkv-create-ami-from-instance.html.
   b. A launch template from your instance. Make sure you choose the AMI that you created in step a (by default the Ubuntu image on your instance will be listed). Verify that the same instance type and security group as your instance are listed. The storage volume should automatically populate after selecting your AMI. See https://docs.aws.amazon.com/autoscaling/ec2/userguide/create-launch-template.html.
   c. An autoscaling group that
      i. Instantiates instances using your launch template.
      ii. Instantiates t2.micro instances. **WARNING: choosing the wrong instance type requirements may incur charges by allowing the creation of instances that are not free-tier eligible!** In step 2, under the "Instance Type Requirements" section, make sure that t2.micro is listed. For availability zones (AZ) and subnet, refer to your instance dashboard. Choose the same AZ as listed in the instance dashboard.
      iii. It is easiest to create both the load balancer and target group during step 3 of the autoscaling group creation process. You will be creating an application load balancer which is internet-facing. You will also need to create a target group to allow your load balancer to send requests to the instances in your autoscaling group. You need to choose at least two AZ and subnets. See

https://docs.aws.amazon.com/autoscaling/ec2/userguide/attach-load-balancer-asg.html.Always contains between 1 and 3 instances.

iv. Uses a CPU-utilization-based scaling policy. A target CPU utilization of around 30% should suffice.

You may find it helpful to reduce the "default cooldown" of the autoscaling group, as this will affect how long it takes for an instance to be created or recycled in response to changing CPU utilization. For assistance with configuring autoscaling groups, see https://docs.aws.amazon.com/autoscaling/ec2/userguide/create-asg.html.

d. Once the load balancer is instantiated. Make sure the load balancer is listening to port 8888 where jupyter notebook server is running. By default, it listens to port 80. You can always add a listener to the load balancer.

5. Using a web browser, connect to http://YOUR-LOAD_BALANCER'S-DNS:8888, using your load balancer's DNS name (shown in the EC2 console) rather than the IP address of any particular instance. Once connected, run the script you saved inside the notebook created in 1.A step 2.8. CPU utilization should increase, prompting the creation of another instance in the autoscaling group. Note that it may take a few minutes for the autoscaling to act. Open the EC2 console (https://console.aws.amazon.com/ec2) and confirm that a new instance appears in list of instances in the "Instances" view. The total number of active instances in your autoscaling group is shown in the "Autoscaling Groups" view. If new instances do not appear, it may help to connect to the instance in the autoscaling group and examine its CPU utilization using, for example, the `htop` utility.

Questions for task 1:

a. When configuring PySpark to use Jupyter as its driver, you were asked to set its execution options using the `PYSPARK_DRIVER_PYTHON_OPTS` environment variable. Why is the `--ip=*` option needed? What happens if it is omitted?

b. What is the difference between "Custom TCP" and "All TCP" types in the EC2 Security Groups? How can you protect your jupyter notebook apart from the credentialed access? Provide the steps.

c. When configuring the pyspark service used by systemd, you were asked to include the following lines in your `pyspark.service` file:

```
After=network.target
After=systemd-user-sessions.service
After=network-online.target
```

Why are these lines needed? What happens if they are omitted?

d. Does increasing the number of instances in your cloud decrease the execution time of Spark jobs? Why or why not?

e. What happens if the minimum number of instances in the autoscaling group is set to zero?

Task 2: Using Amazon EC2 API to deploy the elastic service

As part of this task, you will write either a Python script or a Unix shell script (e.g., Bash or zsh, available for Linux and MacOS) to deploy the same elastic cloud service described earlier, this time using

Amazon's APIs instead of the AWS web interface. This will require you to build a series of signed HTTP requests, send them to AWS, and process the responses. The requests will instruct AWS to create an Amazon Machine Image (AMI), a security group, a launch template, a load balancer, a target group, and an auto scaling group. When creating the AMI, you may reuse the instance you created in 1.B, where you configured it to automatically start a Jupyter server with Spark capabilities. You are **not** allowed to use existing libraries designed specifically for the AWS API, such as the command line interface (AWS CLI) or software development kits (SDKs) provided by Amazon.

For most of this extension, you will need to consult Amazon's online documentation for the functions (called "Actions" in AWS), data types, and error codes that define the following APIs:

- EC2: https://docs.aws.amazon.com/AWSEC2/latest/APIReference.
- Load balancing: https://docs.aws.amazon.com/elasticloadbalancing/latest/APIReference
- Auto scaling: https://docs.aws.amazon.com/autoscaling/ec2/APIReference

You may find helpful examples of how to use each function at the bottom of its dedicated page in the documentation.

Important notes (Read carefully before you start the assignment):

1. Many of the actions triggered using the API take some time to complete. For example, after requesting the creation of a new Amazon Machine Image (AMI), the new AMI may not be ready until several minutes *after* you receive an "OK" response. Your script must be able to handle such delays, either by polling "DescribeX" API functions (e.g., DescribeImages in the EC2 API) for updates or by other means.
2. You can only create and request information for objects that are located in the exact region (e.g., us-east-2, us-east-1, ap-south-1, etc., each corresponding to an availability zone) you direct your requests to. Each region is associated with a service endpoint for handling requests. For more information and a list of available regions and associated service endpoints, see https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Using_Endpoints.html.
3. Be quick to delete any unused AWS resources (e.g., AMIs, load balancers, auto scaling groups, etc.) you request when testing your API calls and running your script. For example, having too many AMIs or auto scaling groups available at the same time may quickly break the AWS free tier limits and incur unnecessary charges. Except for instances created by your auto scaling groups, aim to have at most two of each type of resource (i.e., one you manually created in task 1 and one created using the API) in your cloud at a time. The EC2 dashboard located at https://us-east-2.console.aws.amazon.com/ec2/v2/home and the global view located at https://us-east-2.console.aws.amazon.com/ec2globalview/home provide concise overviews of your total resource usage.
4. Use resource names that are different from the ones you used in your manual setup. For example, if you named your original AMI "HW3 image", you might name your scripted AMI "HW3 API image".

Tasks:

2.A – Use the API to create each component of your elastic cloud service: You must re-implement step 4 of task 1.B using the EC2 API instead of the AWS web interface. The creation of each component of your cloud will require you to issue a carefully constructed HTTP request to either the EC2, ELBV2 (elastic load balancing), or auto scaling API, as described in the steps below.

Before proceeding, you must create an access key using Amazon's Identity and Access Management (IAM) console in order to access the API. In a web browser, go to https://console.aws.amazon.com/iam/home#/security_credentials, find the "Access Keys" tab, and choose "Create New Access Key." You will be shown two codes: an "access key ID," which is a public identifier for the key, and a "secret access key," which you will use to sign your requests to the API. Store these somewhere safe, as you will **not be able to retrieve** the secret access key if you lose it. Do not include your access key in your script; instead, assign them to system environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, which you can then read in your script (see the example code in step 1, below).

Do not share your secret access key with anyone. When you no longer need your key, or if you accidentally shared it online, you should deactivate it immediately using the IAM console.

Step 1: Create an AMI:

You must build, sign, and send an HTTP request to invoke the CreateImage action of the EC2 API. The action requires at least two parameters: "InstanceId", the ID of the instance that will be used to construct the image, and "Name", the name that the new image will be given. For InstanceId, use the ID of the instance you created in task 1.
Several optional parameters are also available and are documented online at https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_CreateImage.html. For this task, code is provided below – you need only to fill in placeholders with your own text as needed and remove the DryRun flag to allow execution of the request. If you choose to use a Unix shell script, you must install *awscurl* using pip install (pip install awscurl).

Bash

```
awscurl -v --access_key $AWS_ACCESS_KEY --secret_key $AWS_SECRET_ACCESS_KEY --service
ec2"https://ec2.REGION.amazonaws.com/\
?Action=CreateImage\
&Version=2016-11-15\
&InstanceId=i-INSTANCE_ID\
&Name=MadeWithShell\
&DryRun=true"
```

Python

```
from requests_aws4auth import AWS4Auth
access_key = os.environ.get("AWS_ACCESS_KEY_ID")
secret_key = os.environ.get("AWS_SECRET_ACCESS_KEY")
auth = AWS4Auth(access_key, secret_key, 'us-east-2', 'ec2')

import requests
response = requests.request("GET",
                            url="http://ec2.us-east-2.amazonaws.com"
                                "?Action=CreateImage"
                                "&Version=2016-11-15"
                                "&InstanceId=i-0b9c149dbc5cd107b"
                                "&Name=MadeWithPython"
                                "&DryRun=true",
                            auth=auth)
```

Step 2: Create a security group:

Create a security group using EC2's "CreateSecurityGroup" action. As earlier, you need to open port 8888 to allow clients to access the Jupyter server (you may also want to open port 22 to allow SSH connections to your instances). Use the "AuthorizeSecurityGroupIngress" and "AuthorizeSecurityGroupEgress" actions to open incoming ("ingress") and outgoing ("egress") ports for your security group.

Step 3: Create a launch template:

Create a launch template using EC2's "CreateLaunchTemplate" action. As earlier, configure the launch template to use your AMI, the t2.micro instance type, and your security group. Note that these properties are embedded within the "LaunchTemplateData" parameter, which takes a data structure of type "RequestLaunchTemplateData".

Step 4: Create a load balancer:

Create an internet-facing application load balancer using ELBV2's "CreateLoadBalancer" action. Set the first three elements of the "Subnets.member" array (i.e., set parameters Subnets.member.0, Subnets.member.1, and Subnets.member.2) to be the names of the existing subnets in your Virtual Private Cloud (VPC). Use EC2's "DescribeSubnets" action and parse the API's response to find the IDs of your subnets. Notice that each subnet is mapped to a different availability zone. Do **not** hardcode the subnet IDs into your script, as they are specific to your account and will not be accessible to others who wish to run your script.

Step 5: Create a target group:

Create a target group for your load balancer using ELBV2's "CreateTargetGroup". Recall that the target group tells your load balancer how to reach the instances in your auto scaling group.

Step 6: Create a load balancer listener:

Create a listener for your load balancer using ELBV2's "CreateListener". When you created your load balancer in task 1, the EC2 console handled this step for you. However, when using the API, you are responsible for creating and configuring a listener. The listener is the part of the load balancer that listens for client connection requests directed for a particular port and forwards the requests according to a set of rules. For this extension, you only need one rule: incoming requests should be forwarded to the target group. When calling "CreateListener", set the following parameters:

- LoadBalancerArn: to assign the listener to your load balancer using it's Amazon Resource Name (ARN; look for a URI starting with the "arn:" prefix)
- Port: to configure the listener to listen on the same port your Jupyter server listens on (8888 by default)
- DefaultActions.member.0: to set the default rule to forward all requests to the target group. Note that the DefaultActions.member.0 parameter – used to configure the default rule – takes an "Action" data structure, which has its own set of configurable properties.

You only need to set the "Type" property of the action to the string value "forward" and the "TargetGroupArn" to the ARN of your target group.

For more information about load balancer listeners, see the documentation at https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-listeners.html

Step 6: Create an auto scaling group:

Create an auto scaling group using the Auto Scaling API's "CreateAutoScalingGroup" action. Set the AutoScalingGroupName (whatever you like), LaunchConfigurationName (the name of the launch template you created in step 3), AvailabilityZones.member.0 (your region, e.g. us-east-1), MinSize (1), MaxSize (3), and DefaultCooldown (30 seconds or so).

The scaling group needs to create and tear down instances according to the average CPU utilization of the group. This scaling policy can be set using the "PutScalingPolicy" action. Configure the scaling policy by setting the parameters "PolicyName" (whatever you like), "AutoScalingGroupName" (the name of your auto scaling group), "PolicyType" (set the string value to "TargetTrackingScaling"), and "TargetTrackingConfiguration". Note that the TargetTrackingConfiguration parameter takes a "TargetTrackingConfiguration" data structure as its value. For examples of how to enter values for these parameters, review the example requests at the bottom of https://docs.aws.amazon.com/autoscaling/ec2/APIReference/API_PutScalingPolicy.html.

2.B – Create a script to deploy an elastic cloud service:

Step 1: Combine the API calls used in Task 1 into a single script:

Create a script named either deploy.sh (Unix shell script) or deploy.py (Python). The script must satisfy three requirements:

- It must take one parameter: a decimal value between 0 and 1 specifying the CPU utilization threshold setting for the scaling policy.
- It must create a file named "messages.txt" and fill it with the raw, plaintext HTTP requests and their corresponding responses in the order that they were sent and received.
- Before the script exits, it should print a URL that can be used to connect to the load balancer following the format http://YOUR-LOAD_BALANCER'S-DNS:8888 (e.g., https://my-load-balancer-692617587.us-east-2.elb.amazonaws.com:8888). You may find the load balancer's DNS name either in response to the CreateLoadBalancer action or using the DescribeLoadBalancers action of the ELBV2 API.

Step 2: Verify the script works:

Run the script and use your web browser to connect to the load balancer's URL. Just as you did in task 1, once connected to the Jupyter server, run the script saved inside the notebook. CPU utilization should increase, prompting the creation of another instance in the auto scaling group. Note that it may take a few minutes for the auto scaling to act. Open the EC2 console (https://console.aws.amazon.com/ec2) and confirm that a new instance eventually appears in the

list of instances in the “Instances” view. The total number of active instances in your auto scaling group is shown in the “Auto Scaling Groups” view.

Additional guidelines:

1. Your script (from 2.B) MUST run using the command line arguments as shown below. It must accept the input argument ***X*** as the CPU utilization for the scaling policy, then print a URL that may be used to connect to the load balancer. Below are examples for running your script for the allowed scripting languages.

   Shell script: ./deploy.sh X
   Python: python deploy.py X

2. A text file containing the sequence of HTTP requests and responses transmitted while executing your script, as detailed in 2.B.1.

3. You will meet with the TA for a live demo where you will also be expected to explain the different processes involved while setting up a scalable system on AWS as well the different processes involved while deploying the cloud service using the script you wrote. The demo will be after the submission deadline. Date and time will be announced closer to the submission deadline.

Submission (3/19/2024@5 pm): **No deadline extension requests will be considered**

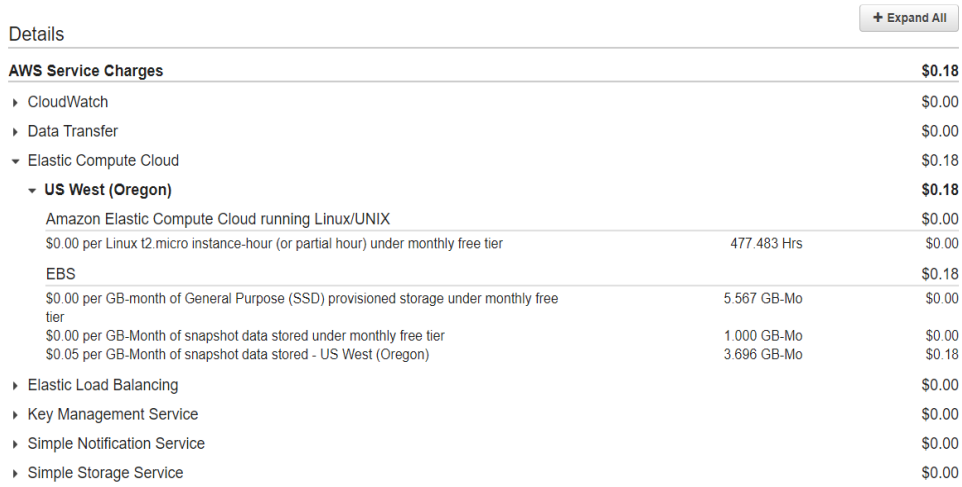
There are two files that you will be uploading as part of the homework submission. A tar file and a PDF file. **Do not tar the PDF file.**

1. Upload the following files within a SINGLE tar filed named “yourLastName_hw3.tar”:
   a. The script from task 2.
   b. The output text file generated from your testing.
   c. Both these files should be tarred into a single file, i.e., the following command should work to create the tar file:

      *$> tar cvf <yourLastName>_hw3.tar **

2. A PDF document (<yourLastName>_hw3.pdf) that includes the answers to the questions as well as the following screenshot(s) of:
   a. Your EC2 instance page specifically capturing the public IPv4 address and the private IPv4 address.
      Note: If you are using two different instances to complete a task, you must provide the screenshot of each instance capturing the public and private IPv4 addresses.
   b. Your terminal that you use to SSH into the EC2 instance to start jupyter notebook in task 1. Your screenshot must show the jupyter notebook server running along with the information and warning messages.
   c. Your browser executing the jupyter notebook showing the files available in the EC2 instance. Your screenshot must include the address bar showing the IP address and the port it is connected to.

d. Your jupyter notebook executing the script provided as part of task 1 which estimates the value of pi. You must provide 3 screenshots capturing the value of pi estimated at different times.
e. Include screenshots of the bill summary (or monthly invoice) for the months of January, February and March, up to the date of submission. The screenshot should include the charges for each service as demonstrated below:



Details — + Expand All

AWS Service Charges		$0.18
▸ CloudWatch		$0.00
▸ Data Transfer		$0.00
▾ Elastic Compute Cloud		$0.18
▾ **US West (Oregon)**		**$0.18**
Amazon Elastic Compute Cloud running Linux/UNIX		$0.00
$0.00 per Linux t2.micro instance-hour (or partial hour) under monthly free tier	477.483 Hrs	$0.00
EBS		$0.18
$0.00 per GB-month of General Purpose (SSD) provisioned storage under monthly free tier	5.567 GB-Mo	$0.00
$0.00 per GB-Month of snapshot data stored under monthly free tier	1.000 GB-Mo	$0.00
$0.05 per GB-Month of snapshot data stored - US West (Oregon)	3.696 GB-Mo	$0.18
▸ Elastic Load Balancing		$0.00
▸ Key Management Service		$0.00
▸ Simple Notification Service		$0.00
▸ Simple Storage Service		$0.00

Submission policy:

- Submit a single PDF file named as specified above. Do not submit the screenshots as individual files. Incorrect submission formats will lead to a grade reduction.
- All submissions are expected by the deadline specified in the homework assignment. Grade is automatically reduced by **25% for every late day**.