Marked Assignment 1 (50%)
IaaS and Python

Objective:

You will write python programs that allow you to automate and manage your AWS cloud services. This assignment will test your ability to use the Boto3 package and Ansible to interact with AWS cloud services, as well as your general Python scripting skills. Use free-tier resources on AWS in all cases where possible.

Boto3 (85%):

Requirements:

1. You will write a python program for interacting with AWS services using the Boto3 package. The program will initially prompt the user with a choice of logging in or registering a new account. A valid username and password must be provided to login to the system. Valid usernames and passwords are to be stored in a file called “passwords.txt”, each line of which will contain a username+password pair, and their AWS access key id and secret access key, all separated by tabs, as well as anything else you may want to store. e.g. the content of a sample “passwords.txt” may be:

```
David       12345pass       AKIKAOD798      PPB7952+AmaYUd+824nmW
John        aaaaaaaaa       AKOP67NKAF      CpA9752SDF+709+fa09faAfG
Joan        bcbcbcddddddd           AIL67NK8NM      KMALF75+95mml+7+89052
```

When registering a new account, the user should be prompted to enter appropriate information to be added to the “passwords.txt” file. When logging in, if the user provides an incorrect username and/or password, issue an appropriate error message and loop, prompting for login credentials again. If the user provides an empty/blank username, exit the program.

- **Include your “passwords.txt” file with your submission, including valid credentials for your AWS account, if it is a new free-tier eligible account.**

2. After providing valid login credentials, the program will use the AWS access key id and secret key associated with that login to form the connection to AWS. Then, the program will present the user with a Python command-line text-based menu for managing AWS. This will direct the user to subsequent submenus outlined below. A back option should be provided to allow the user to easily navigate the menus and to choose another service after an initial choice. There should also be an option to exit the menus and end the program. Ease of use should be taken into account when designing the menu system.

- You may use any python packages to modify the appearance of your program (but note it will be tested in a Windows 10 environment where many appearance modifying packages do not function). **<u>You cannot use any packages that were not covered in the lectures to assist with generation or management of the menu system.</u>**

The Menu should contain the following items:

- EC2 Instances
  - List all instances for the AWS account. Separately display the *running* instances from the others. (i.e. display the running instances first, then display all the other instances afterwards). For each instance, show the state, the instance ID, instance type, region the instance is deployed in and the time it was launched (example screenshot provided) if running.
  - Start a specific instance (prompt user to identify which instance ID, from a list)
  - Stop a specific instance (prompt user to identify which instance ID, from a list)
  - Create an AMI from an existing Instance (prompt user to identify which instance ID, from a list)
  - Delete an AMI (prompt user to identify which instance ID, from a list)

- EBS Storage
  - List all existing volumes
  - Create a new volume (volume parameters specified by the user).
  - Attach an existing volume to an instance (specified by the user)
  - Detach a volume from an instance (specified by the user)
  - Modify a volume to increase its capacity (volume ID and new capacity specified by the user)
  - Take a snapshot of a specific volume (specified by the user)
  - Create a volume from a snapshot(specified by the user)

- S3 Storage
  - List all buckets (make sure your AWS account has at least 1 bucket available for testing)
  - List all objects in a bucket (specified by user)
  - Upload an object
  - Download an object (make it easy to select from the existing objects)
  - Delete a bucket (if the bucket contains any objects, prompt user to confirm the action, otherwise delete it without any confirmation).

- Monitoring
  - Display the *EBSReadBytes* and *EBSByteBalance%* performance metrics gathered for a particular EC2 instance (Prompt the user for the EC2 instance in question), averaged over the last 20 minutes.
  - Set an alarm such that if the *NetworkPacketsOut* is greater than or equal to 1,000 the alarm will be raised. When this alarm is raised, it should stop the instance that triggered it.

  - Select **one** of the following free tier Amazon services.
    - Autoscaling
    - AmazonDB
    - Relational DB Service
    - Elastic Load Balancing

For your chosen service: Provide an implementation that demonstrates how it can be manipulated using boto3. Include a comment which describes the purpose of the service. You should implement 4 short example functions which provide different functionality for the service (e.g. starting it, stopping it, and manipulating it in some ways while active, etc.) and provide a way for the user to call each of the 4 functions.

3. You should use a boto3 "Waiter" when starting an EC2 instance and provide output telling the user to wait until the operation completes, then permit the user to resume navigating the menu after it completes and the instance is active.

4. Provide a pdf "ReadMe" document which serves as a user manual, including any necessary requirements/details for running the script (e.g. what subdirectory to place "passwords.txt" file in, if not in the main directory) and specifying which file to run to open the main menu. **It should also include screenshots demonstrating all working functionality, and indicate any sections which are incomplete or non-functional.** If you do not have a free-tier eligible AWS account, indicate this clearly at the start of the pdf document.

Importantly, please note that you should use best practice when writing your code i.e. use classes/methods where sensible, minimise hardcoding paths, and minimising code duplication. Code should be robust and include error checking so it can deal accordingly with error messages (e.g. if an operation requires that an instance be shut down first, it should not cause the program to crash if we try to run the operation on an active instance). Marks will be awarded for well written robust code.

Your code must include clear concise comments, for functions and also for any complex parts of code which explain what they are doing. These should clearly demonstrate your understanding of the submitted code, i.e. that you understand what type of objects you're working with, why you're calling a method with certain arguments etc...

Ansible (15%):

Your task is to write a python program which performs the following tasks:

1. Prompt the user to input a value between 1-4 inclusive and a group name, and then invoke a playbook that will launch the specified number of AWS EC2 t2.micro (Free-Tier) instances simultaneously. These newly created instances should be added under the provided group name in the hosts file (/etc/ansible/hosts). You must write/generate the playbook yourself. Some examples of launching instances in Ansible have been given at: https://docs.ansible.com/ansible/latest/collections/amazon/aws/docsite/guide_aws.html.

o *Hint:* The `key` parameter for the ec2 module is looking for the key pair name that has been already uploaded to AWS, not a local key i.e. id_rsa.

2. Invoke a second playbook (again, written by you) which will prompt the user to pick a group name (from that group in the hosts file), and then install and run apache2 server on only that group of instances.

o *Hint:* You will need to dynamically create the inventory file for the new hosts. http://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html

1. Provide a pdf ReadMe file which specifies any instructions necessary to correctly run your python script used to run your 2 YAML playbooks. **Include screenshots of the running python script and YAML playbooks, if working, for different user input values. Indicate any non-functional sections. Show how you verify the apache2 servers are running on only the correct group of instances.** You can use the windows “Snipping Tool” to crop images from your screen.

Submission Details:

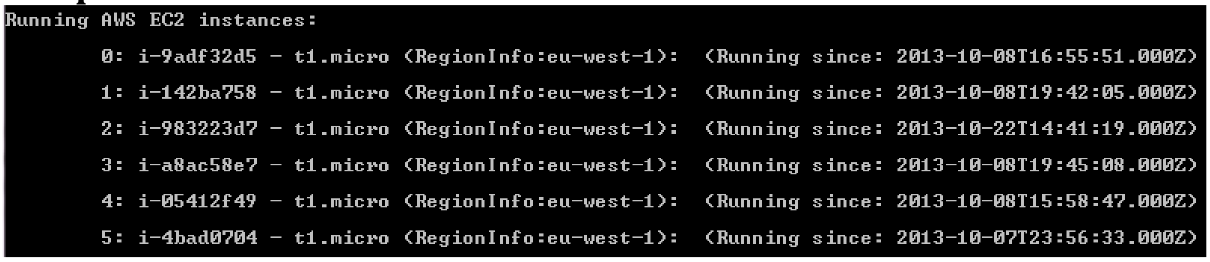
Due Date: 17th November 2024, 23:59.

Submit your code (all Python scripts and Ansible playbooks) along with the ReadMe file in a single zipped directory via Canvas. This directory should be named to have only your name and student number e.g. “David Stynes R00012345”. Marks will be deducted for incorrectly named zip file submissions. An assignment submission facility will be made available in Canvas, provide the same name when uploading your file.

At the discretion of the examiner, students may be briefly interviewed to demonstrate their understanding of the submitted code.

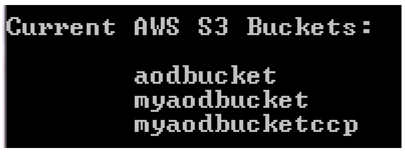
Sample images are for reference only, you are not required to have exactly identical output, you may format it as you wish.

Sample Screen for List all Instances:



```
Running AWS EC2 instances:
        0: i-9adf32d5 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-08T16:55:51.000Z)
        1: i-142ba758 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-08T19:42:05.000Z)
        2: i-983223d7 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-22T14:41:19.000Z)
        3: i-a8ac58e7 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-08T19:45:08.000Z)
        4: i-05412f49 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-08T15:58:47.000Z)
        5: i-4bad0704 - t1.micro (RegionInfo:eu-west-1):  (Running since: 2013-10-07T23:56:33.000Z)
```

Sample Screen for List all Buckets:



```
Current AWS S3 Buckets:

        aodbucket
        myaodbucket
        myaodbucketccp
```