

Assignment 1 Report

Lecturer: Dr. Tom Huynh

Student Name: Do Truong An

Student sID: s3878698

Submission Due Date: 4 Dec 2023.

Introduction

Devops is the combination of the development and operation processes that makes the delivery of any applications and services faster, securer and more effective. One of the key features of Devops model is rapid delivery, the quicker a company can fix bugs, release new features according to the users' needs, the more competitive their products are. In order to accomplish that, engineers will utilise the Continuous Integration (CI) and Continuous Delivery (CD) practices to automate the process from building to deploying an application [1]. The CI practice will let the developers merge the code in one place, after that the builds and tests will be run automatically [2]. While with CD practice, the changes in code will be automatically prepared for the releases in production [3].

To implement this project, software and tools including Git, BashScript, Java, Maven, Tomcat and AWS will be used.

- **Git** is a famous open source version control system and is used for tracking the code changes.
- **Bash Scripting** is a file containing a list of Shell commands that will be executed sequentially.
- **Java** is a multi-platforms and object-oriented programming language that is a popular choice between the developer for creating web and mobile applications.
- **Apache Maven** is a powerful management tool that can be used to build and manage any Java-based application, especially web application.
- **Apache Tomcat** is a web server and servlet container that is used to deploy and host web applications.
- **Amazon Web Services** is a cloud computing platform that offers countless services including storage, database, management tools, etc. Organisations can use these services to quickly update their products, making them more competitive.

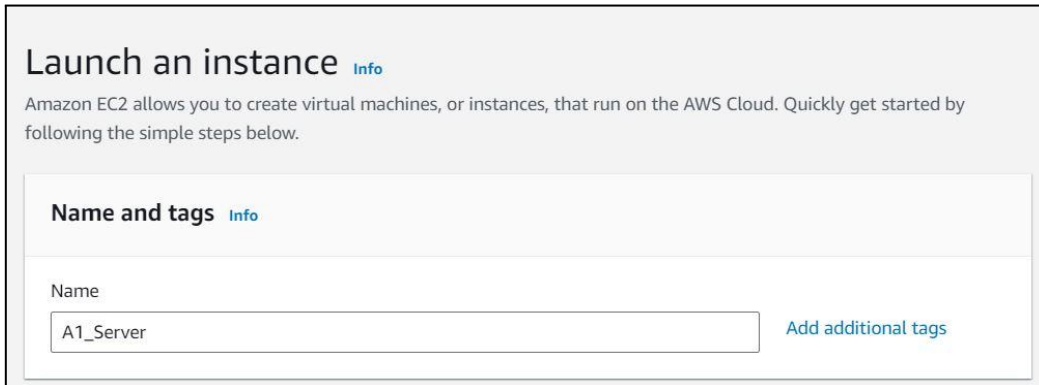
Objective

The main objective of this project is to automate the process of building and deploying a personal website on AWS EC2 instance with bash script. This project requires three steps. The first step is to set up an AWS EC2 instance. The second is bash script writing. The bash script file has to be able to automate the processes including installing Java, install and set up Maven to Path and install Tomcat. The final step is to execute and test the bash script.

Main Requirement Section

Step 1: AWS EC2 Instance Setup

After pressing the Launch Instance button, the setup of an EC2 instance will begin. First, we type in the name of the instance.



Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

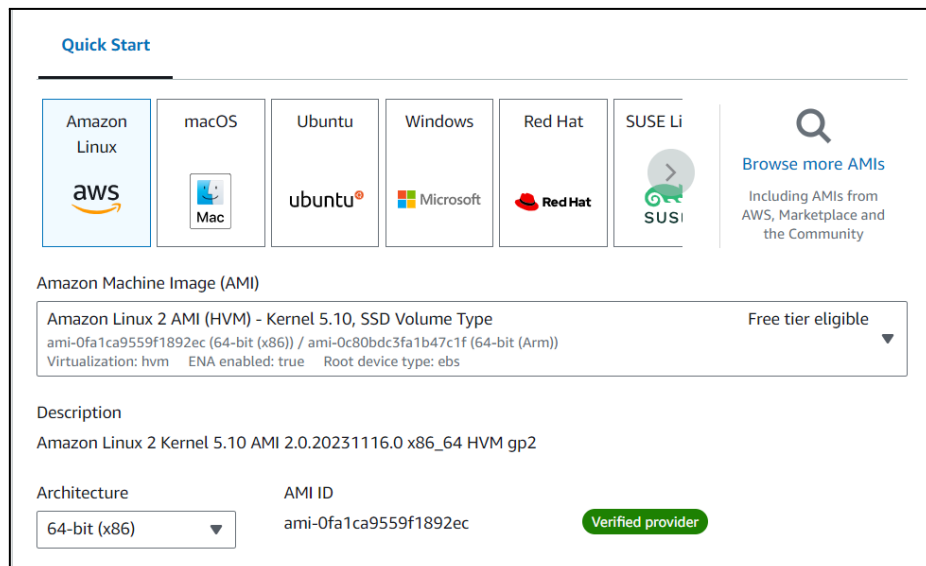
Name and tags [Info](#)

Name

A1_Server [Add additional tags](#)

Figure 1: Set instance name

The second step is choosing the application and operating system image of the instance. In this project, the Amazon Linux 2 Amazon Machine Image for instance because firstly it is in the free tier so it costs less and secondly, the project wants to simulate the process of deploying the website like in a Linux operating system.



Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Li

aws Mac ubuntu Microsoft Red Hat SUSE

[Browse more AMIs](#)
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type Free tier eligible

ami-0fa1ca9559f1892ec (64-bit (x86)) / ami-0c80bdc3fa1b47c1f (64-bit (Arm))

Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2 Kernel 5.10 AMI 2.0.20231116.0 x86_64 HVM gp2

Architecture AMI ID

64-bit (x86) ami-0fa1ca9559f1892ec Verified provider

Figure 2: Choosing the Amazon Linux 2 AMI

The next step is choosing the instance type t2.micro with 1GB is good enough.

▼ Instance type

Info | Get advice

Instance type

t2.micro

Family: t2

1 vCPU

1 GiB Memory

Current generation: true

On-Demand Windows base pricing: 0.0162 USD per Hour

On-Demand SUSE base pricing: 0.0116 USD per Hour

On-Demand RHEL base pricing: 0.0716 USD per Hour

On-Demand Linux base pricing: 0.0116 USD per Hour

Free tier eligible

○ All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Figure 3: Pick a instance type

For the network setting step, the box to allow HTTP and HTTPS needs to be ticked because we are using the instance as a web server. After that, press the Edit button.

▼ Network settings

Info

Edit

Network

Info

vpc-0c0d6536daf9aueb3

Subnet

Info

No preference (Default subnet in any availability zone)

Auto-assign public IP

Info

Enable

Firewall (security groups)

Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

○ Create security group

○ Select existing security group

We'll create a new security group called 'launch-wizard-1' with the following rules:

✓ Allow SSH traffic from

Helps you connect to your instance

Anywhere

0.0.0.0/0

✓ Allow HTTPS traffic from the internet

To set up an endpoint, for example when creating a web server

✓ Allow HTTP traffic from the internet

To set up an endpoint, for example when creating a web server

Figure 4: Allow HTTP and HTTPS traffic

The first thing is, we want to create a new security by adding the name and description. Here the name A1_Server_Security_Group is added.

Firewall (security groups)

Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

○ Create security group

○ Select existing security group

Security group name - required

A1_Server_Security_Group

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and .-:/()#,@[]+=&;!\$*

Description - required

Info

A1_Server_Security_Group

Figure 5: Create New Security Group

And another important thing to setup is adding the 8080 port and allow it to be accessed anywhere since Tomcat uses port 8080 by default as its HTTP connector port. With this setup, when the Tomcat server is set up, we can access it with the public IP provided within the instance.

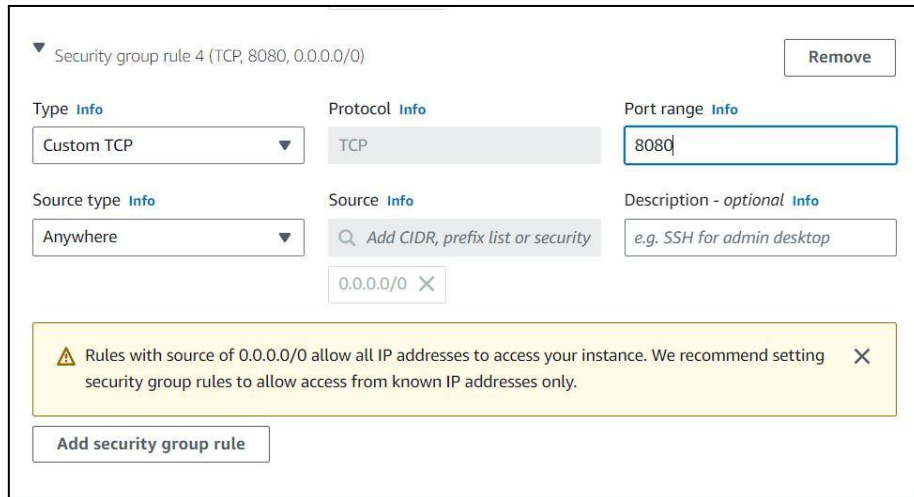


Figure 6: Add port 8080

The next setup is the storage configuration. It is left default with the 8 GB, which is good for our project so we left it as it is.

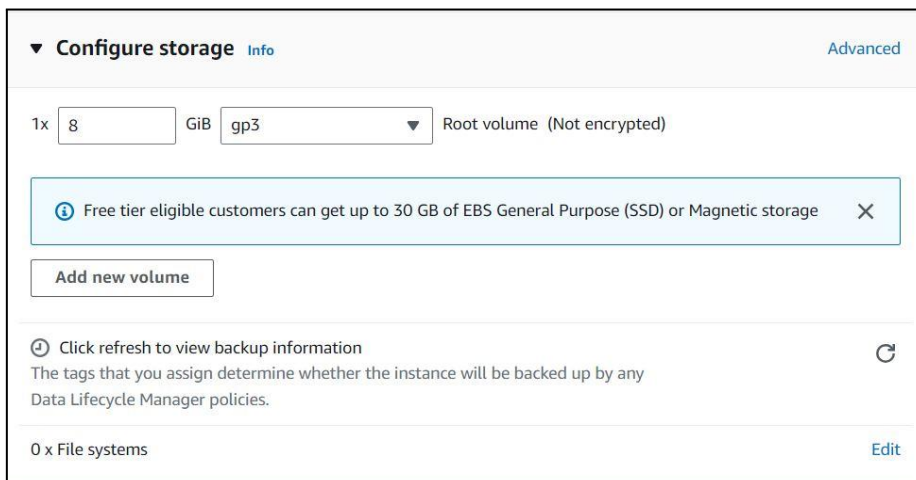


Figure 7: Storage Configuration

Finally, we press the Launch Instance button. The instance will be pending and will be up and running after a moment. When the instance is running, we can click on the box in front of its title, and click the button Connect on the top right of the screen, where it shows the way that we can SSH our instance from the terminal.

The command will be like this:

```
ssh -i "git-devops-key.pem" ec2-user@ec2-3-94-3-176.compute-1.amazonaws.com
```

Figure 8: SSH to the instance

Step 2: Bash Script Setup

In order to pull the bash script stored in the Github repository to automate the setup of building and deploying the personal website. First, we have to SSH (Secure Shell or Secure Socket Shell) to our instance, with the command given previously.

[illegible]

Figure 9: Login to the virtual server

After successfully accessing our virtual server, we type in the “`sudo su -`” command to go to the root directory. We type in the command “`yum install git`” in order to install the Git.

```
Complete!  
[root@ip-172-31-38-233 ~]# git clone https://github.com/andtr-2021/COSC2767_Assignment1_BashScript_AWS_s3878698.git  
Cloning into 'COSC2767_Assignment1_BashScript_AWS_s3878698'...  
Username for 'https://github.com': |
```

Figure 10: Clone Github repository

After a while, Git is installed successfully. We then need to go to the Github repository to get the Git URLs of the repo. Then we type in the command “git clone https://github.com/andtr-2021/COSC2767_Assignment1_BashScript_AWS_s3878698.git” to the terminal.

Because this Github repository is in private mode and we are accessing remotely. So that Github asks us for the authentication, in order to do that, we need to set up your Github Token from our Github account.

First we need to login to our Github account, then go to the Setting section and click on it. Then we need to scroll down and look on the bottom of the menu, we will see the Developer settings, we need to click on it.

Then we go to the Personal access token section and click on the Token (classic). Next, we will click on the Generate new token button which will also choose the classic mode.

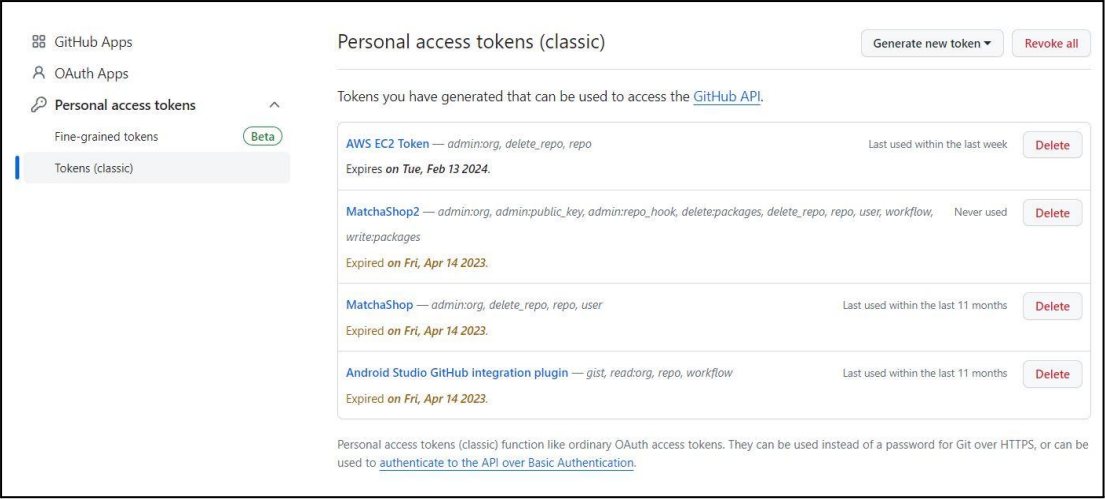


Figure 11: Create Github Token

The token form will appear. In this form, we need to give this Token a name and the number of days before it expires.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

A1 EC2 AWS

What's this token for?

Expiration *

90 days The token will expire on Wed, Feb 28 2024

Figure 12: Set Token Name and Expired Time

In the scope section, we need to check three boxes which are the repo, the admin:org and the delete_repo for its allowance that we can do anything with the repo we try to access.

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

Scope	Description
<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input checked="" type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input checked="" type="checkbox"/> read:org	Read org and team membership, read org projects
<input checked="" type="checkbox"/> manage_runners:org	Manage org runners and runner groups

<input checked="" type="checkbox"/> delete_repo	Delete repositories
---	---------------------

Figure 13: Set up Token properties

Finally we will get the token, which will disappear soon for security reasons so that we need to copy it and save it somewhere like in the text or word file.

Personal access tokens (classic)
Generate new token ▼
Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_kgIwD0Ee2cEcp7GiJUQqhCOq7cE9iA29cGp6	Delete
AWS EC2 Token — admin:org, delete_repo, repo Expires on Tue, Feb 13 2024.	Last used within the last week Delete

Figure 14: Token Create Successfully

Now, pass the token into the Username, if it is preceded to the Password, just press enter one more time. And it's done. We can check it with the command "ls" and we will see the repo as a folder. We can also try to look inside it to see the script.

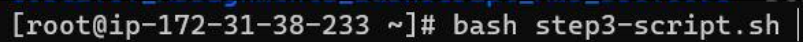
```
Complete!
[root@ip-172-31-38-233 ~]# git clone https://github.com/andtr-2021/COSC2767_Assignment1_BashScript_AWS_s3878698.git
Cloning into 'COSC2767_Assignment1_BashScript_AWS_s3878698'...
Username for 'https://github.com': ghp_kgIwD0Ee2cEcp7GiJUQqhCOq7cE9iA29cGp6
Password for 'https://ghp_kgIwD0Ee2cEcp7GiJUQqhCOq7cE9iA29cGp6@github.com':
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (18/18), 5.64 KiB | 5.64 MiB/s, done.
Resolving deltas: 100% (5/5), done.
[root@ip-172-31-38-233 ~]# ls
COSC2767_Assignment1_BashScript_AWS_s3878698
[root@ip-172-31-38-233 ~]# ls COSC2767_Assignment1_BashScript_AWS_s3878698/
step3-script.sh
[root@ip-172-31-38-233 ~]# mv COSC2767_Assignment1_BashScript_AWS_s3878698/step3-script.sh .
[root@ip-172-31-38-233 ~]# ls
COSC2767_Assignment1_BashScript_AWS_s3878698  step3-script.sh
```

Figure 15: Git Clone Successfully

One of the thing we can do is move the script to the root folder with the command "mv COSC2767_Assignment1_BashScript_AWS_s3878698/step3-script.sh ." where the . means this location.

Step 3: Bash Script in Action

After the bash script is moved to the root directory, to execute the bash script just type the command “bash + script_name.sh”.



```
[root@ip-172-31-38-233 ~]# bash step3-script.sh |
```

Figure 16: Execute the step3-script file

Inside the bash script is a series of commands designed to set up a development environment on an Amazon Linux EC2 instance. The environment includes OpenJDK 11, Apache Maven 3.9.5, and Apache Tomcat 9.0.83. Additionally, it generates a basic Maven web application, configures environment variables, and deploys the application to Tomcat. Here's a breakdown of the script:

1. **Java and Maven Installation:** Install OpenJDK 11 using the amazon-linux-extras command and Apache Maven 3.9.5 by downloading and extracting the binary distribution.
2. **Environment Variable Setup:** Defines environment variables (M2_HOME, M2, JAVA_HOME, PATH) needed for Maven and Java. It modifies the ~/.bash_profile file to include these variables, ensuring they persist across sessions.
3. **Tomcat Installation:** Downloads and extracts Apache Tomcat 9.0.83, renaming the directory to tomcat. Creates symbolic links (“tomcatup” and “tomcatdown”) for starting and stopping Tomcat conveniently.
4. **Tomcat Testing:** Tests Tomcat startup and shutdown using the symbolic links.
5. **Maven Web Application Generation:** Uses Maven to generate a basic web application template (anProfile) with a directory structure and configuration.
6. **Web Application Content:** Creates a simple HTML profile page with information about the developer, programming skills, and contact details. This content is written to the index.jsp file in the webapp directory.
7. **Building and Deploying the Web Application:** Compiles and packages the web application using Maven (mvn package). The resulting WAR (Web Application Archive) file (anProfile.war) is then copied to the Tomcat webapps directory, allowing Tomcat to deploy and run the web application.

Moreover, to deploy the personal website, the Bash script first fetched the web application's source code from the GitHub repository using the git clone command. The project structure, including HTML content and configuration files, was then populated within the generated "anProfile" directory. Subsequently, the Maven build

tool was employed to compile and package the web application into a Web Application Archive (WAR) file. The script navigated to the project directory and executed the mvn package command, resulting in the creation of the "anProfile.war" file containing the compiled artefacts.

Following the successful build, the script facilitated the deployment of the web application on Apache Tomcat. It copied the generated WAR file to the Tomcat "webapps" directory, allowing Tomcat to automatically recognize and deploy the application. Symbolic links for starting and stopping Tomcat were created to simplify management. Finally, with Tomcat running, the website became accessible via a browser using the public IP address and the default port configured for Tomcat (usually 8080). Users could access the personal website by navigating to "http://<public_ip>:8080/anProfile" in their web browser, where "<public_ip>" represents the public IP address of the hosting EC2 instance.

```
Downloaded from central: https://repo.maven.apache.org/maven2/com/github/luben/zstd-jni/1.5.5
[INFO] Packaging webapp
[INFO] Assembling webapp [anProfile] in [/root/anProfile/target/anProfile]
[INFO] Processing war project
[INFO] Copying webapp resources [/root/anProfile/src/main/webapp]
[INFO] Building war: /root/anProfile/target/anProfile.war
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.118 s
[INFO] Finished at: 2023-11-30T09:27:25Z
[INFO] -----
[root@ip-172-31-38-233 ~]# |
```

Figure 17: anProfile Build Success

After all, the packing and building is complete and the website is hosted on Tomcat. To access the personal website, go to the Network section of the instance where the public ip is located.

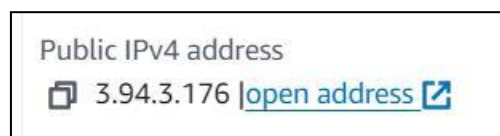


Figure 18: Instance Public IP

Copy and paste it to the web browser like this <http://3.94.3.176:8080/anProfile/> and the page will show up.

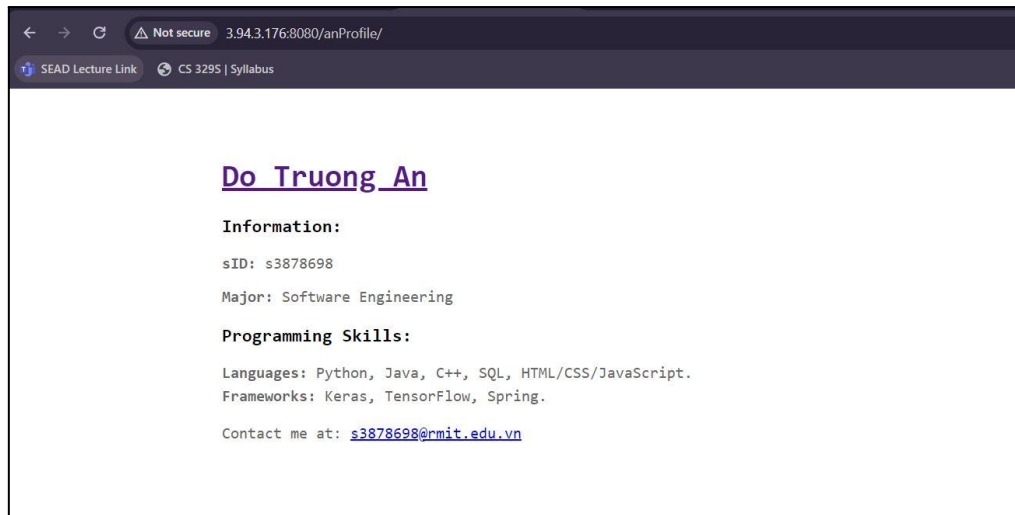


Figure 19: anProfile Personal Website

Advanced Requirement Section:

Automate EC2 Instance Creation and Launch with Bash Script on AWS CloudShell:

In order to create and launch the AWS EC2 with the bash script on AWS CloudShell. We have to collect different import metadata from our AWS Learner Lab account including VPC Id, Subnet Id, AMI Id. Here is how to do that:

VPC Id: go to the AWS Management Console. In the "Find Services" search bar, type "VPC" and select the "VPC" option. To find your VPC Id, click to the Your VPCs section and copy the sequence under the VPC ID column.

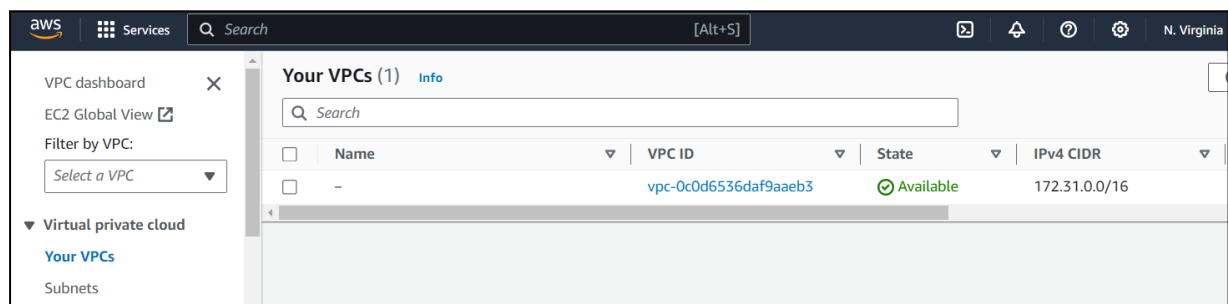


Figure 20: Collect VPC Id

Subnet Id: from the VPC Dashboard, click on "Subnets" in the left navigation pane. Look for the subnet associated with your VPC by parsing the VPC Id to the search bar. Finally, Note down the "Subnet ID" of the desired subnet.

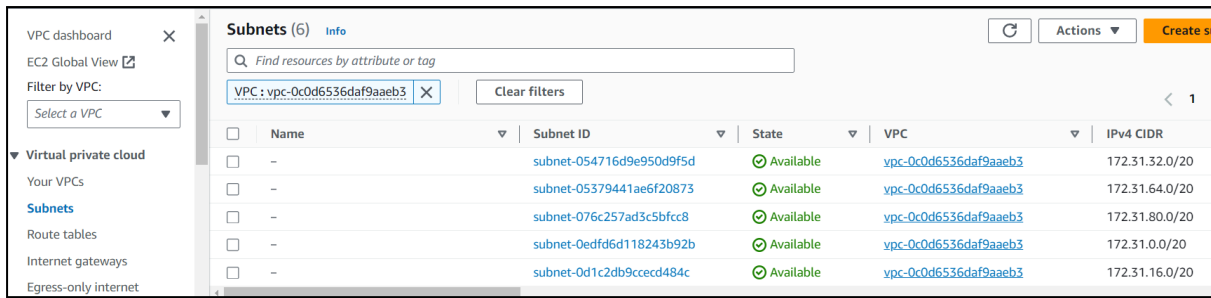


Figure 21: Collect Subnets of the VPC Id

AMI Id: in the AWS Management Console, go to the "Find Services" search bar and type "EC2," then select the "EC2" option. In the EC2 Dashboard, click on "AMI Catalog" in the left navigation pane. Choose the appropriate AMI based on your requirements. In this project, we will choose the Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type for 64-bit x86 computing architecture. And lastly, note down the "AMI ID" of the selected AMI.

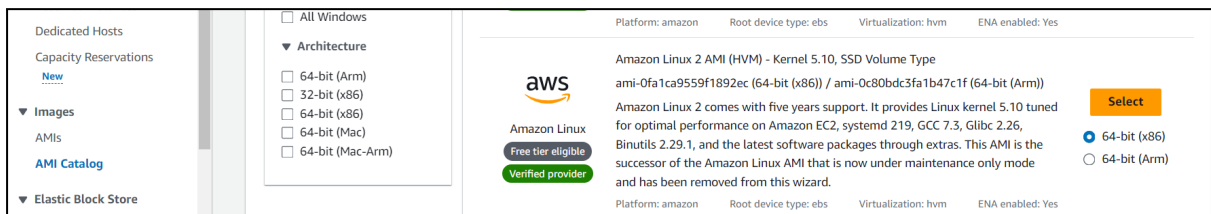


Figure 22: Pick a AMI Id

After collecting all of the above information, we will login to the AWS Management Console Dashboard, by pressing the CloudShell button located at the bottom left of the page. The CloudShell terminal will appear.

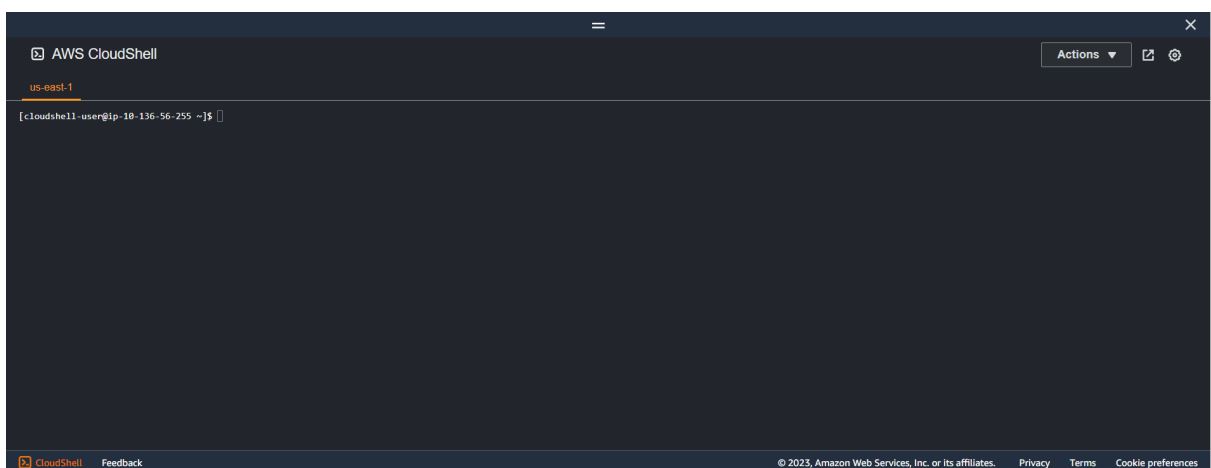


Figure 23: CloudShell UI

Create Key Pair:

The next step is to create a key pair with the command: **aws ec2 create-key-pair --key-name \$KEY_NAME --query 'KeyMaterial' --output text > "\$KEY_NAME.pem"** where the KEY_NAME is the variable for the name of the key pair. To change the name of the key pair, just use the command: **KEY_NAME = "MyKeyPair"**. This command will generate a .pem file in the current directory where we are at.

Create Security Group:

The we needs to create a security group with the command: **aws ec2 create-security-group --group-name \$SECURITY_GROUP_NAME --description "My security group" --vpc-id \$VPC_ID** where the SECURITY_GROUP_NAME and VPC_ID variables are the name of the security group and the VPC Id of the AWS account respectively. When the command is pasted into the AWS CloudShell, it will generate a security group id and we need to save it in order to edit the security group rules including adding necessary ports.

Modify Security Group Rules:

With the security group id, we will edit the security group rules with this command: **aws ec2 authorize-security-group-ingress --group-id \$group_id --protocol tcp --port 22 --cidr 0.0.0.0/0**. This command will add the port 22 to the security group and allow it to be accessed anywhere, allowing us to ssh from anywhere with aws cli to the virtual server. We will do the same for the port 8080, which Tomcat server needs to be accessed locally.

Launch EC2 Instance:

Finally, we need to launch and run the instance with this command: **aws ec2 run-instances --image-id \$IMAGE_ID --count 1 --instance-type t2.micro --key-name \$KEY_NAME --security-group-ids \$group_id --subnet-id \$SUBNET_ID**. This command carries all the information about the instance and launches it. After a while, the instance will be up and running. But before SSH to the instance, we need to modify the permission to use the key pair with the command: **chmod 600 "\$KEY_NAME.pem"** to make it executable for the file owner only.

SSH to the Instance:

Now, we can ssh to the instance with the command: **ssh -i <file_name>.pem ec2-user@<public_ip>**

```

# Create key pair
aws ec2 create-key-pair --key-name $KEY_NAME --query 'KeyMaterial' --output text > "$KEY_NAME.pem"
# Create a security group
command_output=$(aws ec2 create-security-group --group-name $SECURITY_GROUP_NAME --description "My security group" --vpc-id $VPC_ID)
# Extract GroupId from the command output using jq (assuming JSON format)
group_id=$(echo "$command_output" | jq -r '.GroupId')
# Print or use the GroupId as needed
echo "GroupId: $group_id"

# Authorize security group rules
authorize_security_group() {
    aws ec2 authorize-security-group-ingress \
        --group-id $group_id \
        --protocol tcp \
        --port $1 \
        --cidr 0.0.0.0/0
}

# Authorize SSH port
authorize_security_group $PORT_SSH
# Authorize Tomcat port
authorize_security_group $PORT_TOMCAT
# Launch the instance
aws ec2 run-instances --image-id $IMAGE_ID --count 1 --instance-type t2.micro --key-name $KEY_NAME --security-group-ids $group_id --subnet-id $SUBNET_ID
# Edit permissions for the file owner
chmod 600 "$KEY_NAME.pem"

```

Figure 23: EC2 Instance Creation Script

Executing the bash script file, it generates the GroupId as expected and it shows the modification on the security group rules.

```

[cloudshell-user@ip-10-138-169-39 ~]$ bash step1-script.sh
GroupId: sg-001e9884d70c1038f
{
  "Return": true,
  "SecurityGroupRules": [
    {
      "SecurityGroupRuleId": "sgr-0f1048615fe11299d",
      "GroupId": "sg-001e9884d70c1038f",
      "GroupOwnerId": "409419035620",
      "IsEgress": false,
      "IpProtocol": "tcp",
      "FromPort": 22,
      "ToPort": 22,
      "CidrIpv4": "0.0.0.0/0"
    }
  ]
}

```


Figure 24: Execute The EC2 Instance Creation Script

After a while, the instance is up and running, However, with the aws cli approach, we don't have a command to set the instance name from the bash, so if we want to do so, we can add the name under the Name column.

Instances (1/1) Info								
Find Instance by attribute or tag (case-sensitive)								
<input checked="" type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input checked="" type="checkbox"/>		i-Obad01316422a54d4	Running	t2.micro	Initializing	No alarms	us-east-1f	ec2-3-238

Figure 25: a Instance Created and Lunachne

Public IP address

 3.238.204.66

User name

Enter the user name defined in the AMI used to launch the instance.
ec2-user.

Figure 26: Instance IP address and Username

```
[cloudshell-user@ip-10-138-169-39 ~]$ ssh -i MyKeyPair101.pem ec2-user@3.238.204.66
The authenticity of host '3.238.204.66 (3.238.204.66)' can't be established.
ECDSA key fingerprint is SHA256:0ffx+CyajazZACM8RunWad1ORxsx9naSQCqgW+5Zoe0.
ECDSA key fingerprint is MD5:5c:ba:8e:4b:26:ad:b6:7e:65:53:d0:a0:99:25:98:d1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '3.238.204.66' (ECDSA) to the list of known hosts.
```

```
#
~\##### Amazon Linux 2
nu \#####\
nu \###| AL2 End of Life is 2025-06-30.
nu \
nu V~' '->
nuw /
nuw _-
nuw _/_/_/_/_
m/'
```

A newer version of Amazon Linux is available!

Amazon Linux 2023, GA and supported until 2028-03-15.
<https://aws.amazon.com/linux/amazon-linux-2023/>

```
[ec2-user@ip-172-31-78-71 ~]$ █
```

Figure 27: SSH to the Instance Successfully

To automate the process of cloning a private GitHub repository using a personal access token for authentication. Here's a concise breakdown of each step:

`repo_url`: Specifies the URL of the GitHub repository to be cloned.

2. Configure Git Credentials:

3. Perform Repository Clone:

4. Cleanup:

Removes the stored credentials file (~/.git-credentials) after the clone operation is completed. This step is a security measure to avoid leaving sensitive information in the system. Outputs a success message indicating that the clone operation has been completed successfully.

```
repo_url="https://github.com/andtr-2021/COSC2767_Assignment1_BashScript_AWS_s3878698.git"
github_token="ghp_3kEKCvvs0FGH4FdjUL0JDvupWBQikt2wM074"

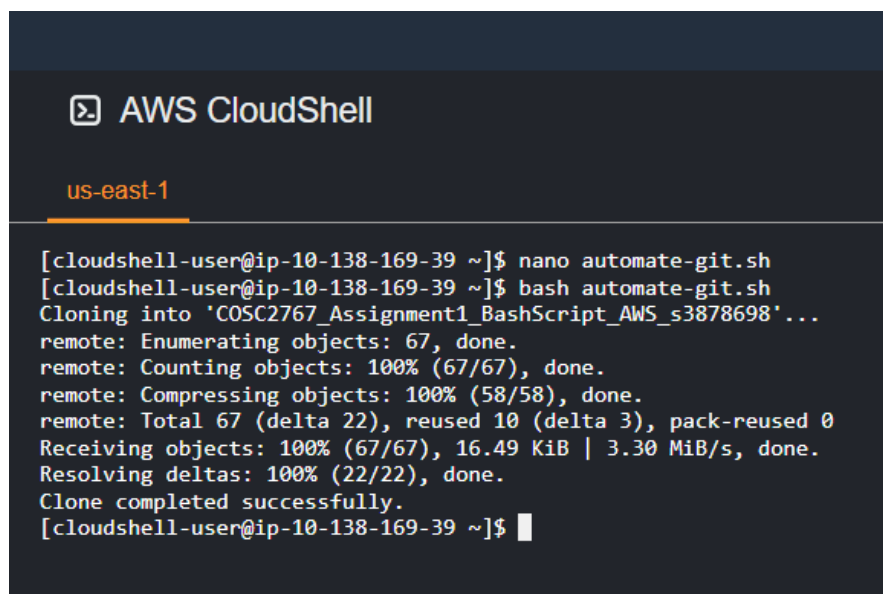
# Set the GitHub token as the username for authentication
git config --global credential.helper store
echo "https://$github_token:x-oauth-basic@github.com" > ~/.git-credentials

# Perform the clone again, the token will be used for authentication
git clone "$repo_url"

# Cleanup: remove the stored credentials after the clone
rm -f ~/.git-credentials

echo "Clone completed successfully."
```

Figure 28: Automate Git Clone with Bash Script



```
AWS CloudShell
us-east-1

[cloudshell-user@ip-10-138-169-39 ~]$ nano automate-git.sh
[cloudshell-user@ip-10-138-169-39 ~]$ bash automate-git.sh
Cloning into 'COSC2767_Assignment1_BashScript_AWS_s3878698'...
remote: Enumerating objects: 67, done.
remote: Counting objects: 100% (67/67), done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 67 (delta 22), reused 10 (delta 3), pack-reused 0
Receiving objects: 100% (67/67), 16.49 KiB | 3.30 MiB/s, done.
Resolving deltas: 100% (22/22), done.
Clone completed successfully.
[cloudshell-user@ip-10-138-169-39 ~]$
```

Figure 29: Automate Git Clone Successfully

Advantages and Challenges

The advanced solution provides several notable advantages over the main requirements. Firstly, automation streamlines the entire EC2 instance creation process, reducing the human errors and increasing the efficiency of resource provisioning. This is particularly beneficial for scenarios where multiple instances need to be created consistently.

The automated Git cloning process, utilising a personal access token, not only speeds up the process but also improves security by eliminating the need for manual credential input.

Different challenges come up in the process of creating the bash script for launching an EC2 instance. Firstly, it requires reading and understanding the aws cli syntax while writing each command and making modifications according to our needs. Secondly, writing the bash script requires a lot of testing, everytime it fails such as when the instance is not even launched or when the instance is launched but can not ssh to it, it requires updating a new name for the keypair and the security group name, which is quite problematic.

Conclusion

After completing this assignment, I have gained valuable insights into key principles of DevOps practices including becoming better at scripting and configuration from writing the bash script to automate the deployment of a personal website and automate the Github authorization process as well as problem solving and troubleshooting from setting up a bash script to create an EC2 instance. The idea of automating programs always makes me feel very intrigued, even AI is also just a smarter automation tool. From this assignment, I learned a lot and really started doing automated programs, which makes me very happy. I really enjoy the time spent on this assignment.

References

- [1] AWS. "What is DevOps?" aws.amazon.com.
<https://aws.amazon.com/devops/what-is-devops/> (accessed Nov. 21, 2023)
- [2] AWS. "What is Continuous Integration?" aws.amazon.com.
<https://aws.amazon.com/devops/continuous-integration/> (accessed Nov. 21, 2023)
- [3] AWS. "What is Continuous Delivery?" aws.amazon.com.
<https://aws.amazon.com/devops/continuous-delivery/> (accessed Nov. 21, 2023)