Deployment Portfolio Task 2

SWE40006 – SOFTWARE DEVELOPMENT AND EVOLUTION

SUMMER – 2024 SUBMITTED ON 2ND OF JUNE

Student and Lecturer Details

Name	ID	Lecturer	Class
Vi Luan Dang	103802759	Dr. Thomas Hang Nsam@swin.edu.au	Monday 13:00 PM

Self-Assessment Details

Declaration ò task level attempted (P/C/D/HD)

	Pass	Credit	Distinction	High Distinction
Self-Assessment				✓

	Included & attempted
Task 2.1: Pass	✓
Task 2.2: Credit	✓
Task 2.3: Distinction	✓
Task 2.4: High Distinction	✓

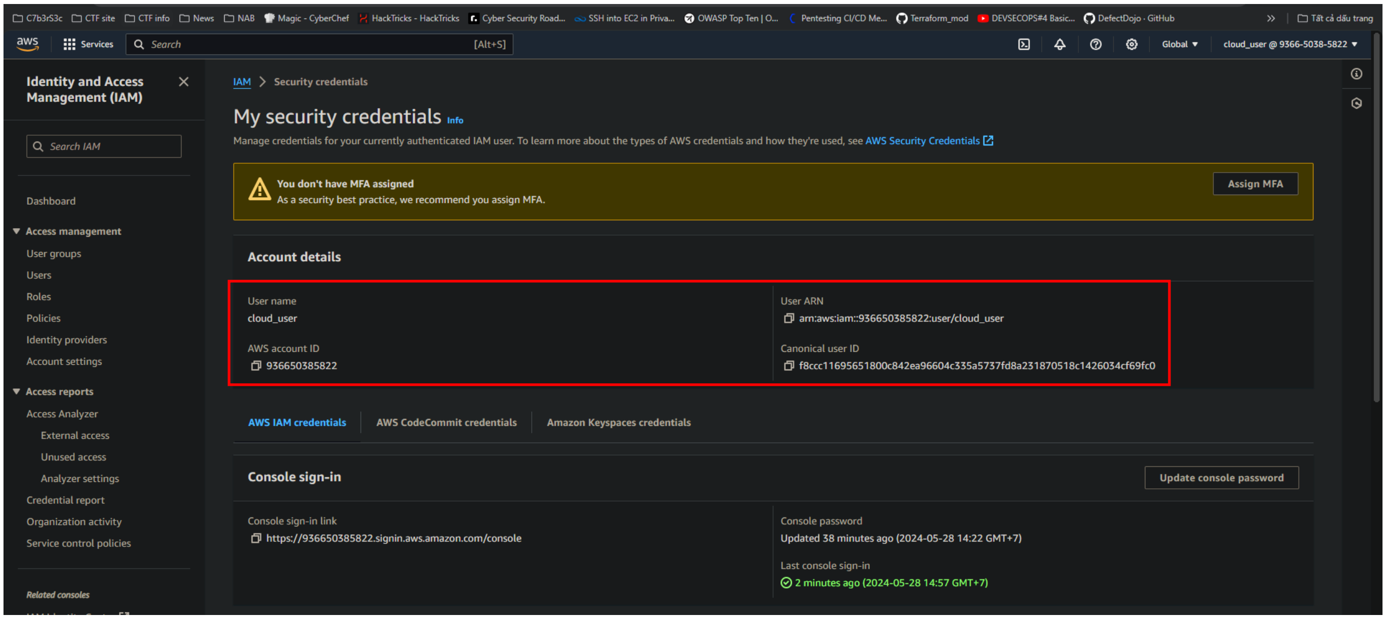
Table of Content

Assignment Report

Task 2.1P: By following AWS documentation on the links provided complete the following tasks

2.1A. Open an AWS account

This task is fairly simple, as I already have an account set up on AWS, we can also use AWS command line interface (CLI) to verify the account, the following figure will prove the completion of this task.



My security credentials Info

Manage credentials for your currently authenticated IAM user. To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials

You don't have MFA assigned
As a security best practice, we recommend you assign MFA.

Account details

User name: cloud_user

User ARN: arn:aws:iam::936650385822:user/cloud_user

AWS account ID: 936650385822

Canonical user ID: f8ccc11695651800c842ea96604c335a5737fd8a231870518c1426034cf69fc0

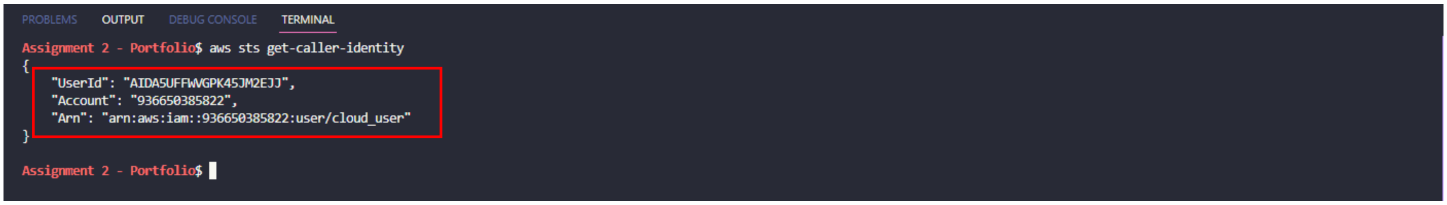
Console sign-in

Console sign-in link: https://936650385822.signin.aws.amazon.com/console

Console password: Updated 38 minutes ago (2024-05-28 14:22 GMT+7)

Last console sign-in: 2 minutes ago (2024-05-28 14:57 GMT+7)

Figure 1: AWS Account

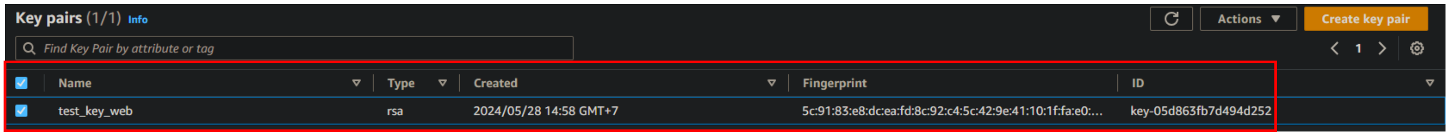


```
Assignment 2 - Portfolio$ aws sts get-caller-identity
{
    "UserId": "AIDA5UFFWGPK45JM2EJJ",
    "Account": "936650385822",
    "Arn": "arn:aws:iam::936650385822:user/cloud_user"
}

Assignment 2 - Portfolio$ 
```

Figure 2: AWS Account verification on AWS CLI

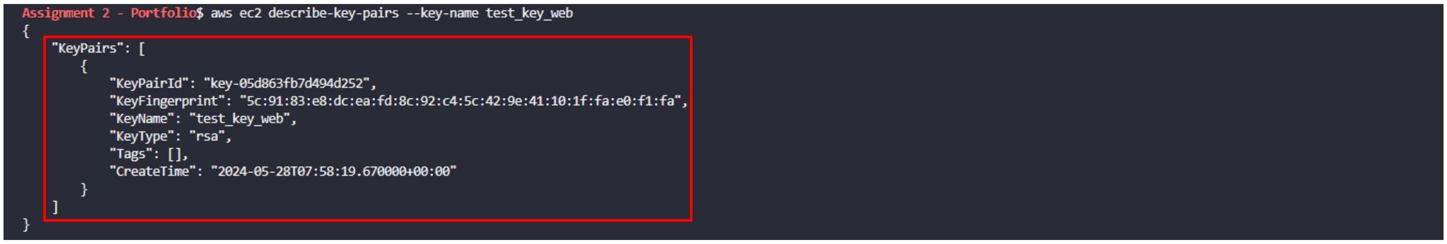
2.1B. Create an SSH key pair

For this task, we will create a key pair named “test_key_web” and use it for the rest of our assignment. After the creation, we can use AWS CLI and GUI to check for the completion of this task.



☑	Name	Type	Created	Fingerprint	ID
☑	test_key_web	rsa	2024/05/28 14:58 GMT+7	5c:91:83:e8:dc:ea:fd:8c:92:c4:5c:42:9e:41:10:1f:fa:e0:...	key-05d863fb7d494d252

Figure 3: SSH Key pair

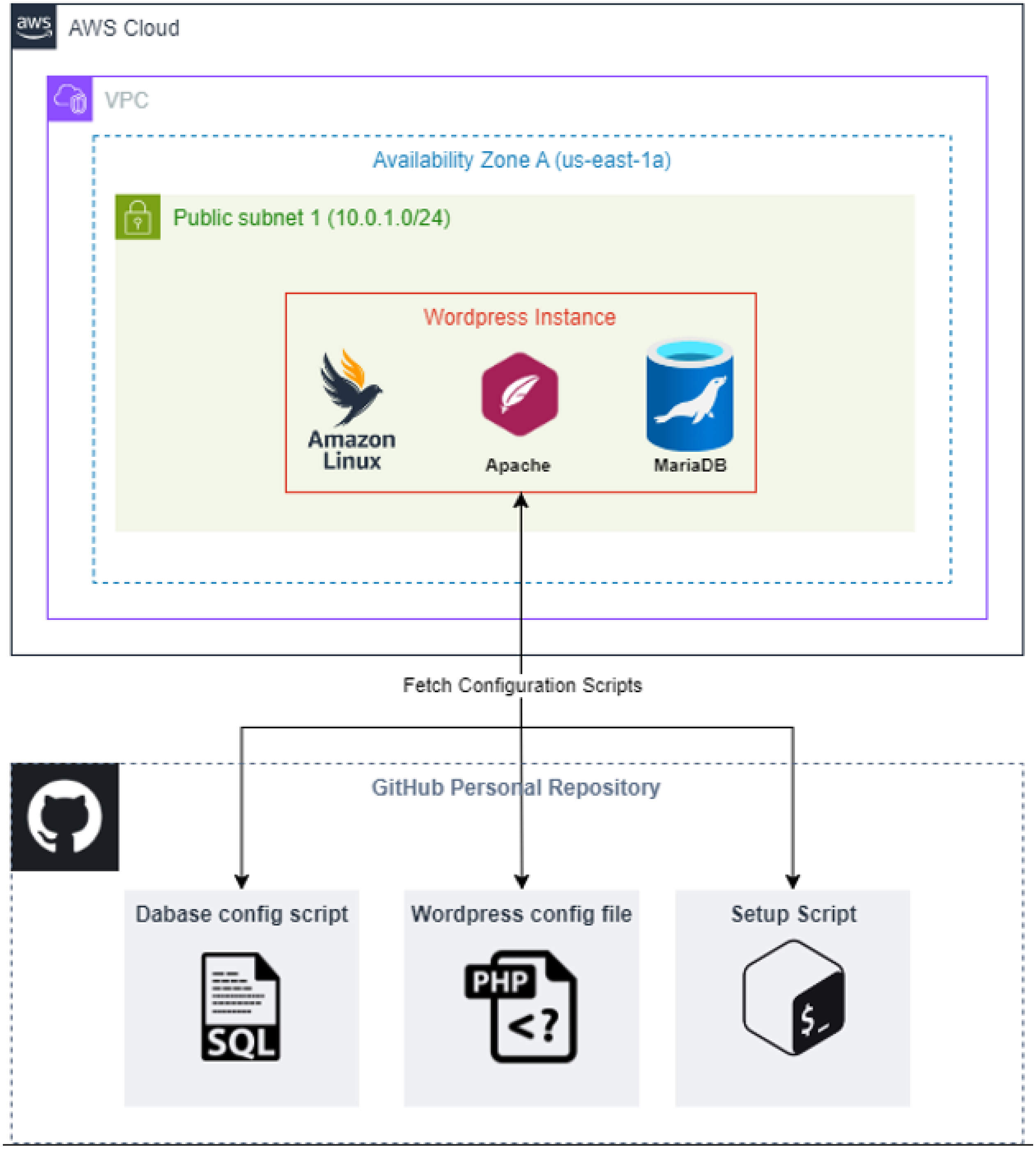


```
Assignment 2 - Portfolio$ aws ec2 describe-key-pairs --key-name test_key_web
{
    "KeyPairs": [
        {
            "KeyPairId": "key-05d863fb7d494d252",
            "KeyFingerprint": "5c:91:83:e8:dc:ea:fd:8c:92:c4:5c:42:9e:41:10:1f:fa:e0:f1:fa",
            "KeyName": "test_key_web",
            "KeyType": "rsa",
            "Tags": [],
            "CreateTime": "2024-05-28T07:58:19.670000+00:00"
        }
    ]
}
```

Figure 4: SSH key pair on AWS CLI

2.1C&D. Create an EC2 Instance and deploy WordPress to it

The diagram below illustrates my overall set up for this task:

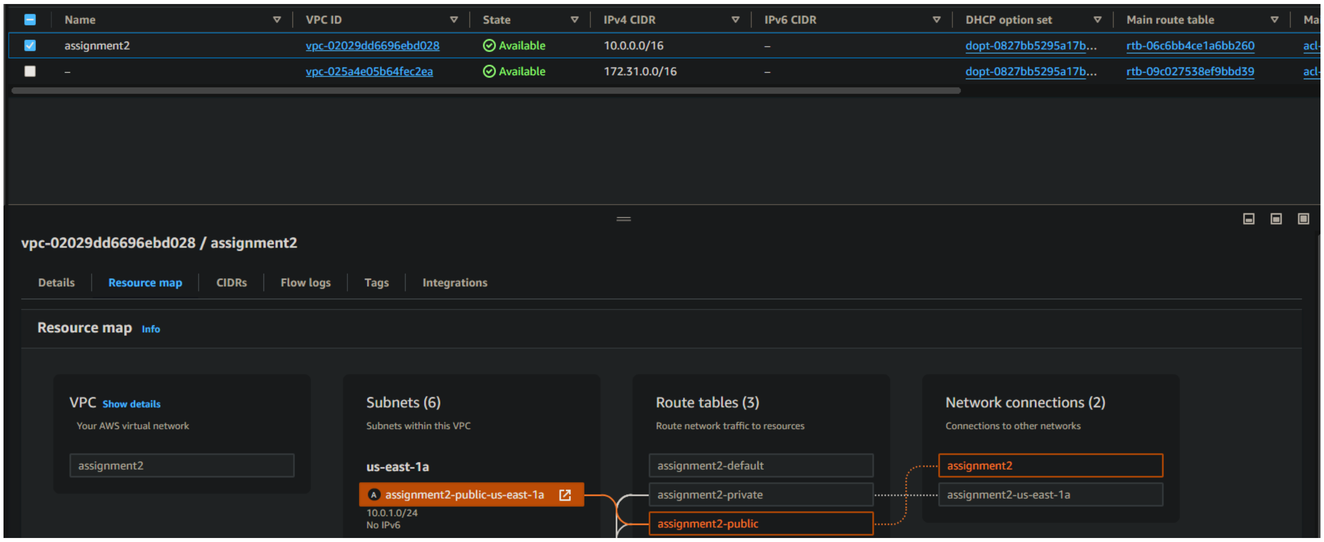


AWS Cloud
VPC
Availability Zone A (us-east-1a)
Public subnet 1 (10.0.1.0/24)
Wordpress Instance
Amazon Linux
Apache
MariaDB
Fetch Configuration Scripts
GitHub Personal Repository
Dabase config script
SQL
Wordpress config file
PHP
Setup Script

Figure 5: WordPress Instance setup diagram.

Our WordPress Instance on AWS will be comprised of 3 main components: **Amazon Linux 2** (OS), **Apache** (Web Server), and **MariaDB** (Database). My aim for this task is to use just one script to set up our instance, therefore, the instance will fetch scripts from my personal GitHub repository to set up its service. Before taking a look at the script, let's provision our infrastructure on AWS.

Firstly, we will create a VPC with a public subnet in accordance with the diagram above.

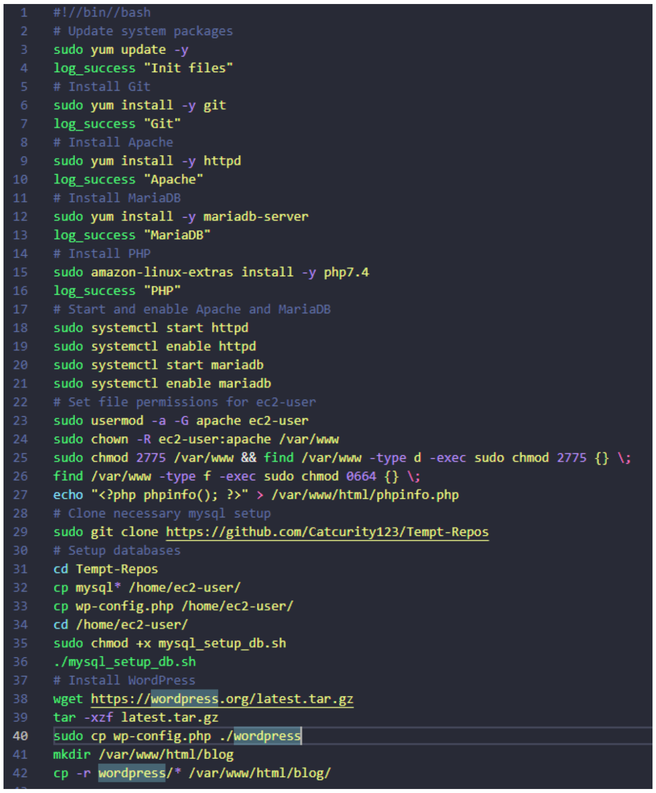


☐	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table	Ma
☑	assignment2	vpc-02029dd6696ebd028	✓ Available	10.0.0.0/16	–	dopt-0827bb5295a17b...	rtb-06c6bb4ce1a6bb260	acl-
☐	–	vpc-025a4e05b64fec2ea	✓ Available	172.31.0.0/16	–	dopt-0827bb5295a17b...	rtb-09c027538ef9bbd39	acl-

vpc-02029dd6696ebd028 / assignment2

Figure 6: VPC and Public Subnet.

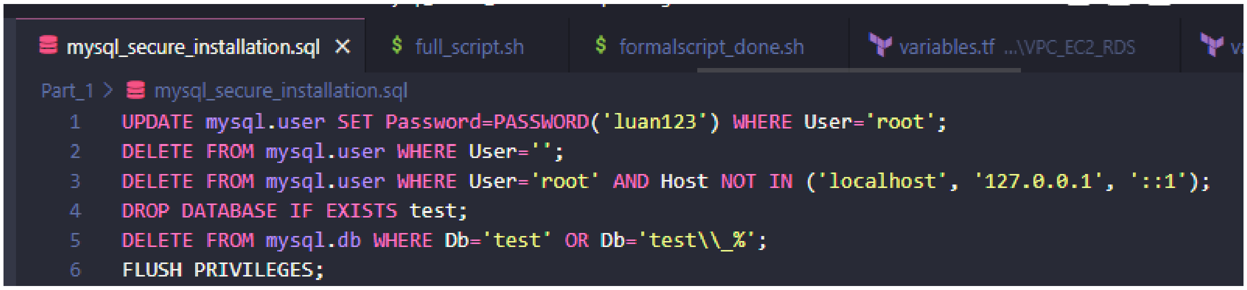
Then we will launch an EC2 instance in the public subnet, we will also include user data, which can be thought of as a shell script for instance at launch. Our user data is as follows:



```
#!//bin//bash
# Update system packages
sudo yum update -y
log_success "Init files"
# Install Git
sudo yum install -y git
log_success "Git"
# Install Apache
sudo yum install -y httpd
log_success "Apache"
# Install MariaDB
sudo yum install -y mariadb-server
log_success "MariaDB"
# Install PHP
sudo amazon-linux-extras install -y php7.4
log_success "PHP"
# Start and enable Apache and MariaDB
sudo systemctl start httpd
sudo systemctl enable httpd
sudo systemctl start mariadb
sudo systemctl enable mariadb
# Set file permissions for ec2-user
sudo usermod -a -G apache ec2-user
sudo chown -R ec2-user:apache /var/www
sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
find /var/www -type f -exec sudo chmod 0664 {} \;
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
# Clone necessary mysql setup
sudo git clone https://github.com/Catcurity123/Tempt-Repos
# Setup databases
cd Tempt-Repos
cp mysql* /home/ec2-user/
cp wp-config.php /home/ec2-user/
cd /home/ec2-user/
sudo chmod +x mysql_setup_db.sh
./mysql_setup_db.sh
# Install WordPress
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
sudo cp wp-config.php ./wordpress
mkdir /var/www/html/blog
cp -r wordpress/* /var/www/html/blog/
```

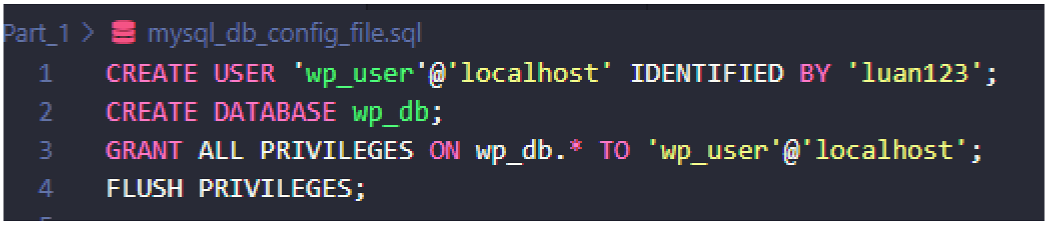
Figure 7: EC2 Instance user data.

Our user data will automatically install Git, Apache Web Server, MariaDB, and PHP7.4 onto our instance, the required services are then set to start even when our instance restarts. We will also need to set appropriate permissions for the user – ec2-user, so that we can modify our instances when needed. Finally, in order to set up the database and WordPress, our instance will clone necessary files from my personal GitHub repository and execute them. The following scripts will be cloned.


mysql_secure_installation.sql
full_script.sh
formalscript_done.sh
variables.tf ...\VPC_EC2_RDS
Part_1 > mysql_secure_installation.sql

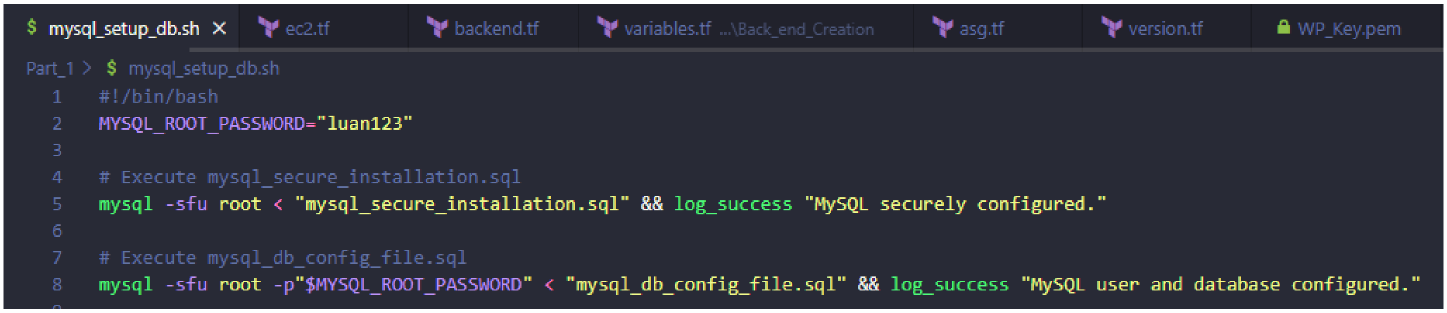
```sql
UPDATE mysql.user SET Password=PASSWORD('luan123') WHERE User='root';
DELETE FROM mysql.user WHERE User='';
DELETE FROM mysql.user WHERE User='root' AND Host NOT IN ('localhost', '127.0.0.1', '::1');
DROP DATABASE IF EXISTS test;
DELETE FROM mysql.db WHERE Db='test' OR Db='test\\_%';
FLUSH PRIVILEGES;
```

Figure 9: Script to install MySQL.



```sql
CREATE USER 'wp_user'@'localhost' IDENTIFIED BY 'luan123';
CREATE DATABASE wp_db;
GRANT ALL PRIVILEGES ON wp_db.* TO 'wp_user'@'localhost';
FLUSH PRIVILEGES;
```

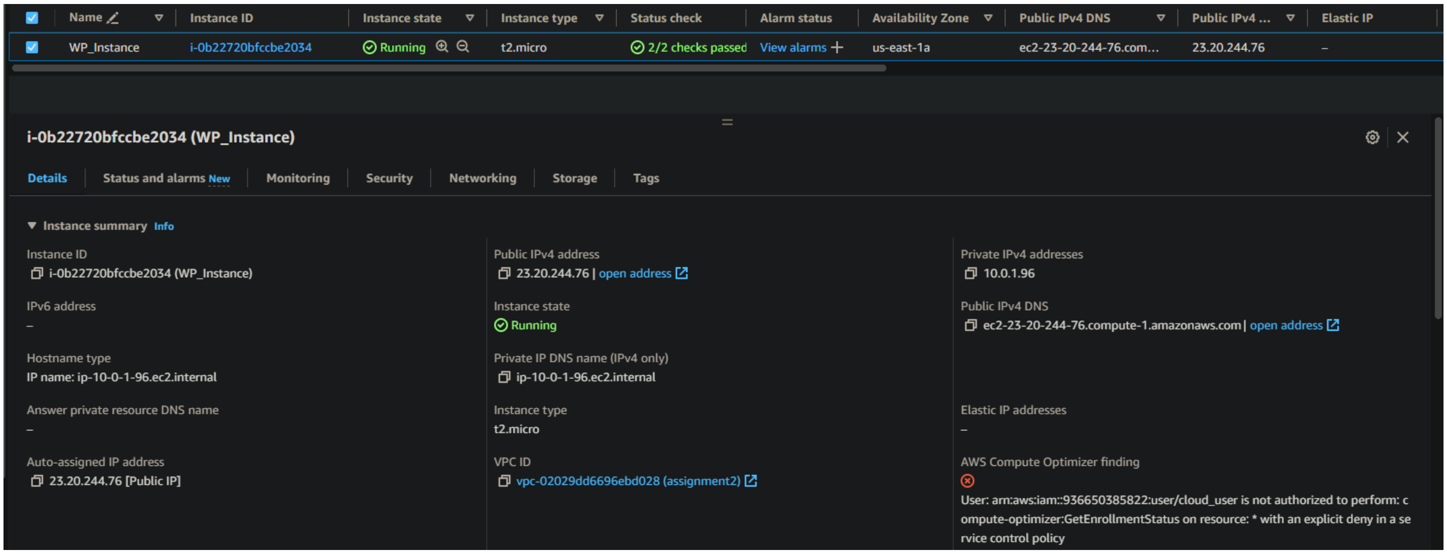
Figure 8: Script to create user and database for WordPress.


mysql_setup_db.sh
ec2.tf
backend.tf
variables.tf ...\Back_end_Creation
asg.tf
version.tf
WP_Key.pem
Part_1 > mysql_setup_db.sh

```bash
#!/bin/bash
MYSQL_ROOT_PASSWORD="luan123"

# Execute mysql_secure_installation.sql
mysql -sfu root < "mysql_secure_installation.sql" && log_success "MySQL securely configured."

# Execute mysql_db_config_file.sql
mysql -sfu root -p"$MYSQL_ROOT_PASSWORD" < "mysql_db_config_file.sql" && log_success "MySQL user and database configured."
```

Figure 9: Master Script for setting up MariaDB.

We will then provision the EC2 Instance and see if our scripts work as expected.



☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
☑	WP_Instance	i-0b22720bfccbe2034	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +	us-east-1a	ec2-23-20-244-76.com...	23.20.244.76	–

i-0b22720bfccbe2034 (WP_Instance)

▼ Instance summary Info

Instance ID
i-0b22720bfccbe2034 (WP_Instance)

IPv6 address
–

Hostname type
IP name: ip-10-0-1-96.ec2.internal

Answer private resource DNS name
–

Auto-assigned IP address
23.20.244.76 [Public IP]

Public IPv4 address
23.20.244.76 | open address

Instance state
✓ Running

Private IP DNS name (IPv4 only)
ip-10-0-1-96.ec2.internal

Instance type
t2.micro

VPC ID
vpc-02029dd6696ebd028 (assignment2)

Private IPv4 addresses
10.0.1.96

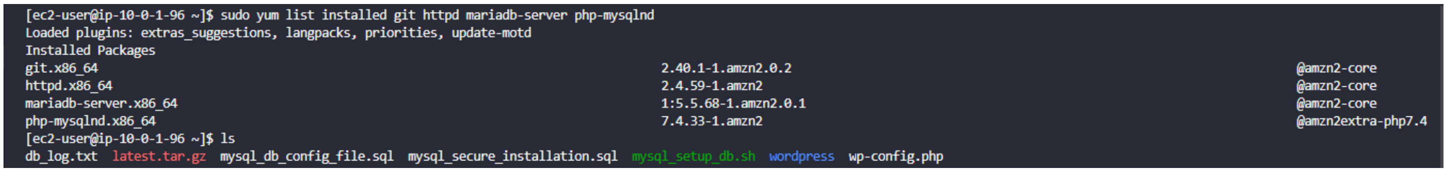
Public IPv4 DNS
ec2-23-20-244-76.compute-1.amazonaws.com | open address

Elastic IP addresses
–

AWS Compute Optimizer finding
⊗
User: arn:aws:iam::936650385822:user/cloud_user is not authorized to perform: compute-optimizer:GetEnrollmentStatus on resource: * with an explicit deny in a service control policy

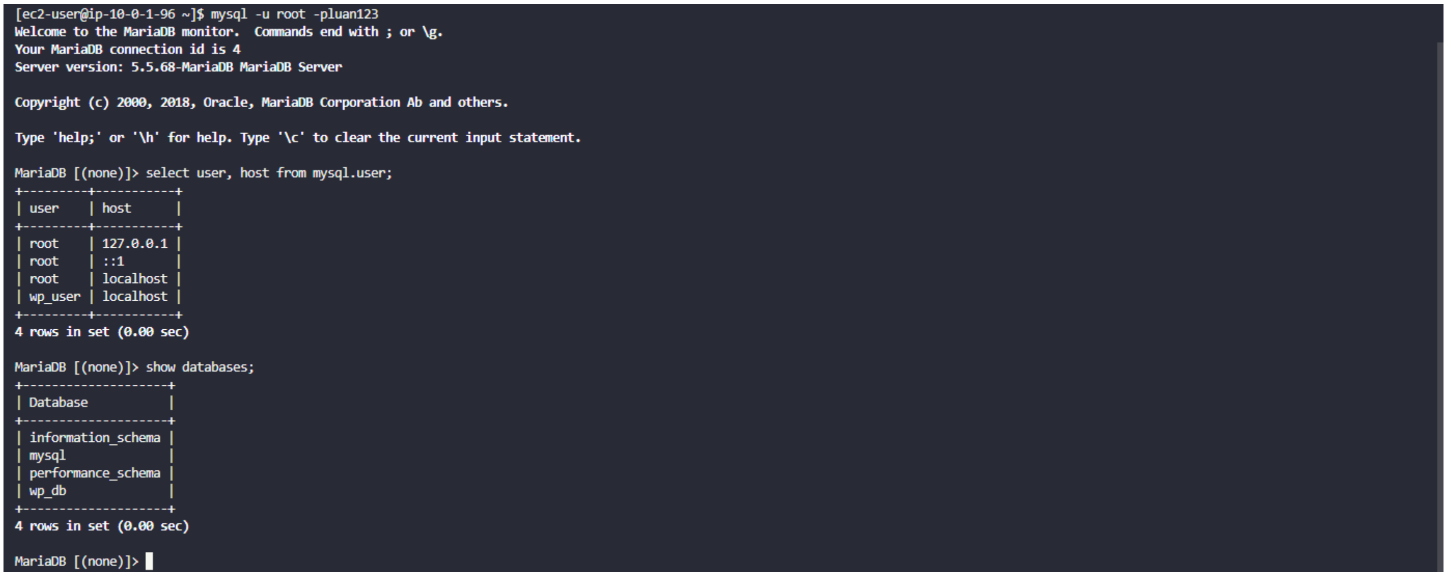
Figure 10: EC2 Instance successfully provisioned.

We can further verify our installation process by SSH into the instance and check if our services are working as expected.



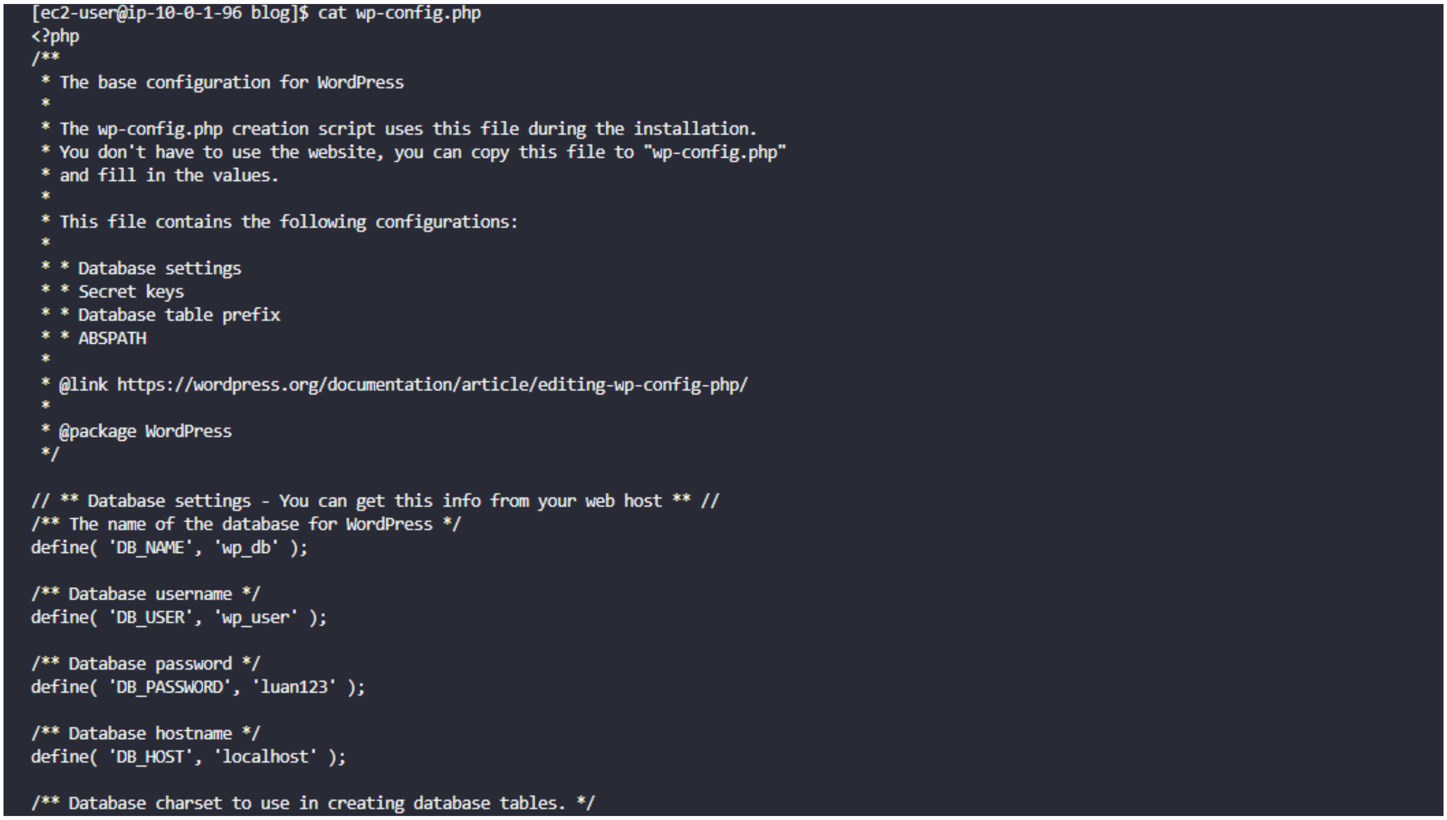
```
[ec2-user@ip-10-0-1-96 ~]$ sudo yum list installed git httpd mariadb-server php-mysqlnd
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Installed Packages
git.x86_64                                          2.40.1-1.amzn2.0.2                                          @amzn2-core
httpd.x86_64                                        2.4.59-1.amzn2                                              @amzn2-core
mariadb-server.x86_64                               1:5.5.68-1.amzn2.0.1                                        @amzn2-core
php-mysqlnd.x86_64                                  7.4.33-1.amzn2                                              @amzn2extra-php7.4
[ec2-user@ip-10-0-1-96 ~]$ ls
db_log.txt  latest.tar.gz  mysql_db_config_file.sql  mysql_secure_installation.sql  mysql_setup_db.sh  wordpress  wp-config.php
```

Figure 11: Services and Scripts



```
[ec2-user@ip-10-0-1-96 ~]$ mysql -u root -pluan123
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 5.5.68-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> select user, host from mysql.user;
+---------+-----------+
| user    | host      |
+---------+-----------+
| root    | 127.0.0.1 |
| root    | ::1       |
| root    | localhost |
| wp_user | localhost |
+---------+-----------+
4 rows in set (0.00 sec)

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| wp_db              |
+--------------------+
4 rows in set (0.00 sec)

MariaDB [(none)]>
```

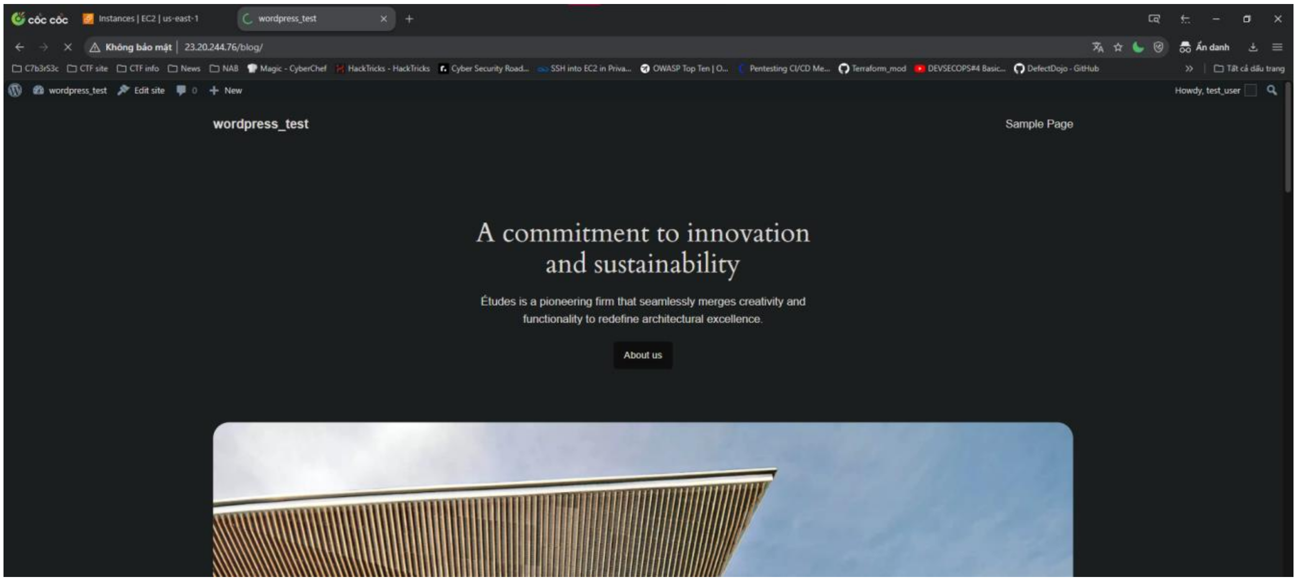
Figure 12: MariaDB's user and database successfully created.



```
[ec2-user@ip-10-0-1-96 blog]$ cat wp-config.php
<?php
/**
 * The base configuration for WordPress
 *
 * The wp-config.php creation script uses this file during the installation.
 * You don't have to use the website, you can copy this file to "wp-config.php"
 * and fill in the values.
 *
 * This file contains the following configurations:
 *
 * * Database settings
 * * Secret keys
 * * Database table prefix
 * * ABSPATH
 *
 * @link https://wordpress.org/documentation/article/editing-wp-config-php/
 *
 * @package WordPress
 */

// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wp_db' );

/** Database username */
define( 'DB_USER', 'wp_user' );

/** Database password */
define( 'DB_PASSWORD', 'luan123' );

/** Database hostname */
define( 'DB_HOST', 'localhost' );

/** Database charset to use in creating database tables. */
```

Figure 13: WordPress successfully downloaded and configured.

After verifying our user data, we can browse our services using the public IP to test its functionalities.



A commitment to innovation and sustainability

Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

Figure 14: WordPress successfully installed.

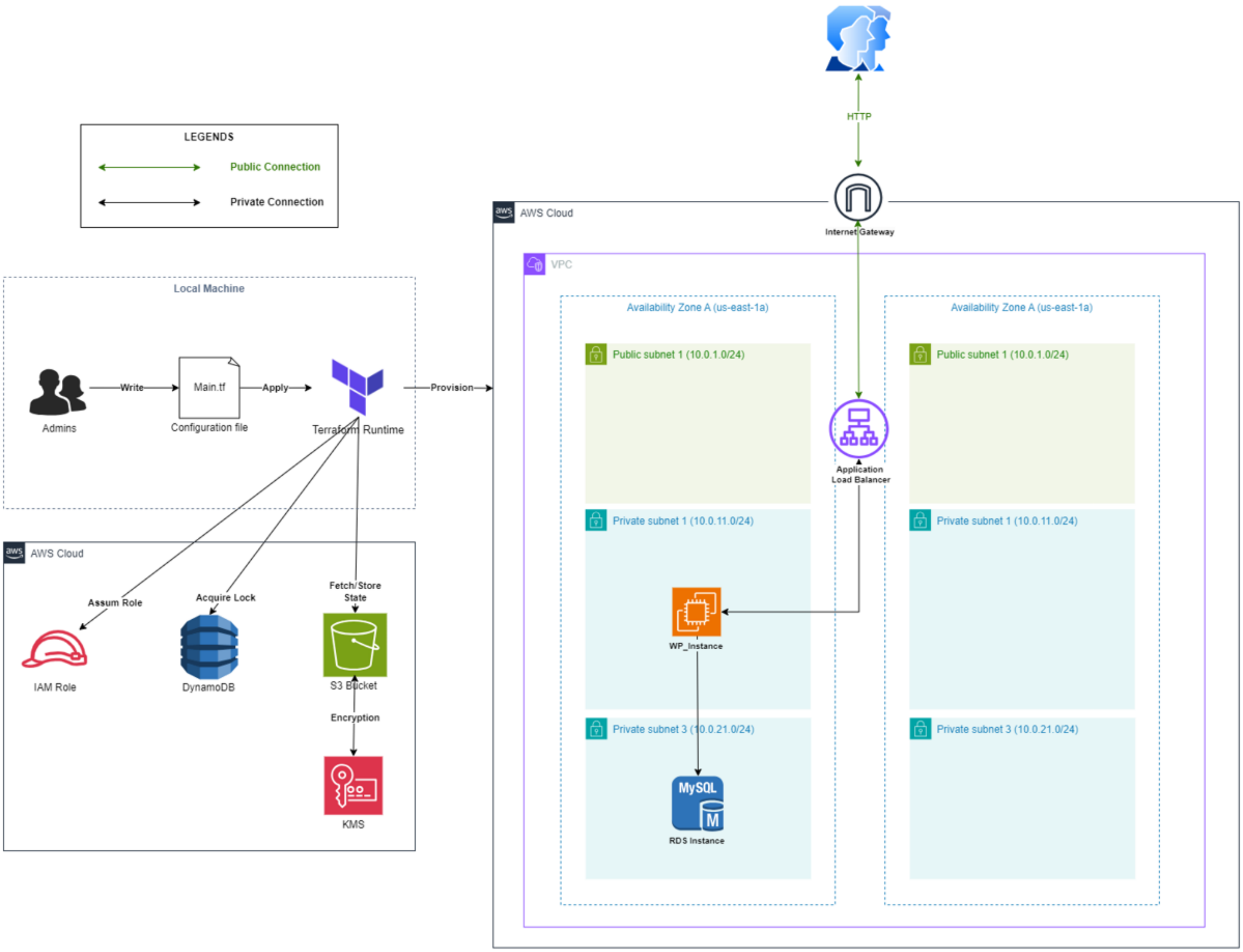
Figure 15 shows that our instance is successfully installed with WordPress, and it is working as expected. This will also conclude Task 2.1P of this assignment.

Task 2.2C: By following AWS documentation on the links provided complete the following tasks

2.2A. Create an ELB and Re-install WordPress using an external database

For this task, we will create an Elastic Load Balancer (ELB), specifically an Application Load Balancer, and relocate our WordPress instance to a private subnet. The ELB will be configured as internet-facing, thereby connecting our private subnet to the internet. Additionally, we will decouple our instance by migrating the database to a separate private subnet using AWS Relational Database Service (RDS), while continuing to utilize MariaDB as our database service.

Furthermore, to promote automation, we will employ Terraform, a widely used and highly effective Infrastructure as Code (IAC) tool, to provision our services on AWS. Therefore, our diagram can be revised as follows:



LEGENDS
Public Connection
Private Connection
Local Machine
Admins
Write
Main.tf
Configuration file
Apply
Terraform Runtime
Provision
AWS Cloud
Assum Role
Acquire Lock
Fetch/Store State
IAM Role
DynamoDB
S3 Bucket
Encryption
KMS
HTTP
Internet Gateway
AWS Cloud
VPC
Availability Zone A (us-east-1a)
Availability Zone A (us-east-1a)
Public subnet 1 (10.0.1.0/24)
Public subnet 1 (10.0.1.0/24)
Application Load Balancer
Private subnet 1 (10.0.11.0/24)
Private subnet 1 (10.0.11.0/24)
WP_Instance
Private subnet 3 (10.0.21.0/24)
Private subnet 3 (10.0.21.0/24)
MySQL
M
RDS Instance

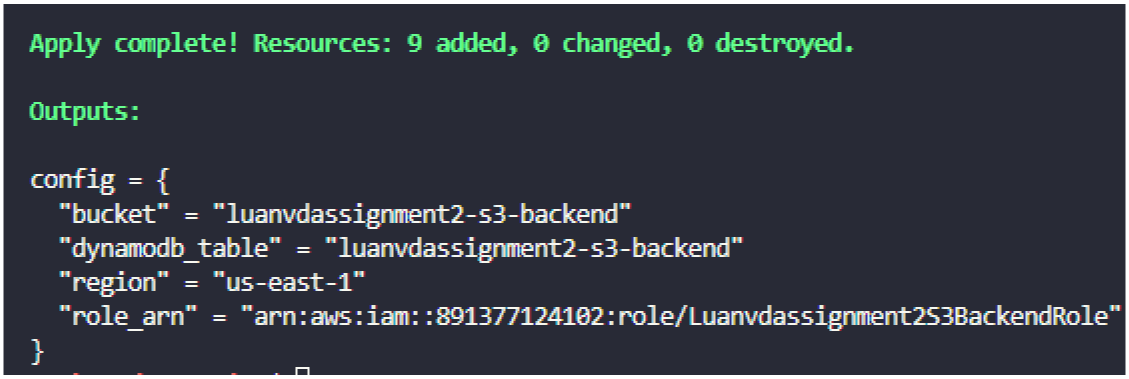
Figure 15: WordPress Instance provisioned using Terraform diagram.

We will divide this task into 3 parts:
+ **The creation of backend for Terraform state on Amazon S3.** This backend will also support state locking and consistency checking via DynamoDB. Once this is completed our terraform state file will be stored on Amazon S3, facilitating future teamwork and consistency.
+ **The configuration steps using Terraform on the local machine**, configuration files will then be used to provision required services on AWS Cloud.
+ **The provisioning of required services on AWS Cloud,** there will be 4 main services that need to be provisioned: AWS VPC, AWS EC2 Instances, AWS RDS, and an Application Load Balancer (ALB).

The creation of backend for Terraform state on Amazon S3.

This configuration is an extra step for ensuring a centralized state management, collaboration, and security of access control. Whenever other administrators wish to change the provisioned infrastructure on AWS, the lock on DynamoDB will be checked to ensure consistency and will only be granted if no one else is modifying the state file. After acquiring the lock, the administrator can modify the state file store on AWS S3 bucket and make changes to the infrastructure on AWS Cloud.

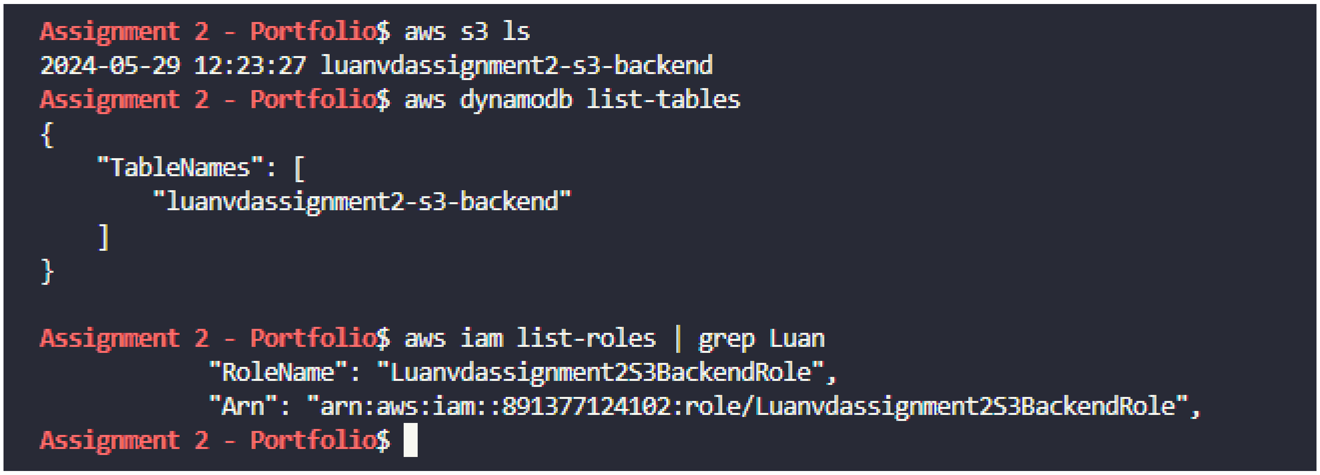
For this task we will use Terraform to create an S3 bucket, a DynamoDB table, and an IAM policy for our AWS user to assume the role and make modifications to these services.



```
Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Outputs:

config = {
  "bucket" = "luanvdassignment2-s3-backend"
  "dynamodb_table" = "luanvdassignment2-s3-backend"
  "region" = "us-east-1"
  "role_arn" = "arn:aws:iam::891377124102:role/Luanvdassignment2S3BackendRole"
}
```

Figure 16: The backend successfully created.

We can then use AWS CLI to verify the creation of our backend as follows:



```
Assignment 2 - Portfolio$ aws s3 ls
2024-05-29 12:23:27 luanvdassignment2-s3-backend
Assignment 2 - Portfolio$ aws dynamodb list-tables
{
    "TableNames": [
        "luanvdassignment2-s3-backend"
    ]
}

Assignment 2 - Portfolio$ aws iam list-roles | grep Luan
            "RoleName": "Luanvdassignment2S3BackendRole",
            "Arn": "arn:aws:iam::891377124102:role/Luanvdassignment2S3BackendRole",
Assignment 2 - Portfolio$ 
```

Figure 17: Backend creation verified.

With the backend successfully created, we can move on to write Terraform codes for provisioning the necessary AWS services.

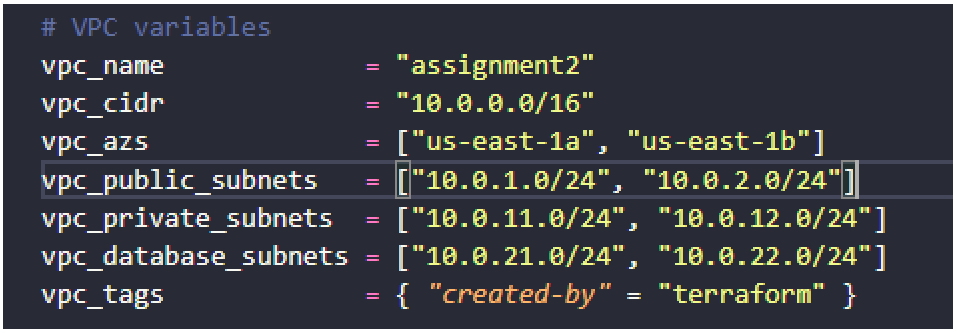
The configuration steps using Terraform on the local machine to provision AWS services.

Our WordPress Instance requires 4 main components that need to be configured on AWS Cloud, including:
+ **AWS VPC**: we will provision our VPC with 2 Availability Zones for our Load Balancer to work correctly, inside each Availability Zones there will be 3 subnets, 1 public subnet for internet-facing connection, and 2 private subnets for our instances and database.
+ **AWS EC2**: Our EC2 Instance would not be so different from its counterpart in task 2.1P. However, as now we are using an external database, extra configuration will need to be executed.
+ **AWS RDS**: We aim to decouple our EC2 Instance by moving MariaDB within the WordPress Instance to an external database, this is done by using an RDS Instance on a private subnet.
+ **AWS Application Load Balancer**: Currently, we will assign a load balancer onto public subnet 1 and public subnet 2, however we will only register it with one EC2 Instance to test if it works as expected, later on in this assignment, we will do additional configuration to ensure high availability and scaling for our system.

For this task we will use Terraform to provision an Application Load Balancer, an EC2 Instance, an RDS Instance, and a VPC for our infrastructure. The full code for this section can be found in **GitHub Repository.**

The provisioning of required services on AWS Cloud

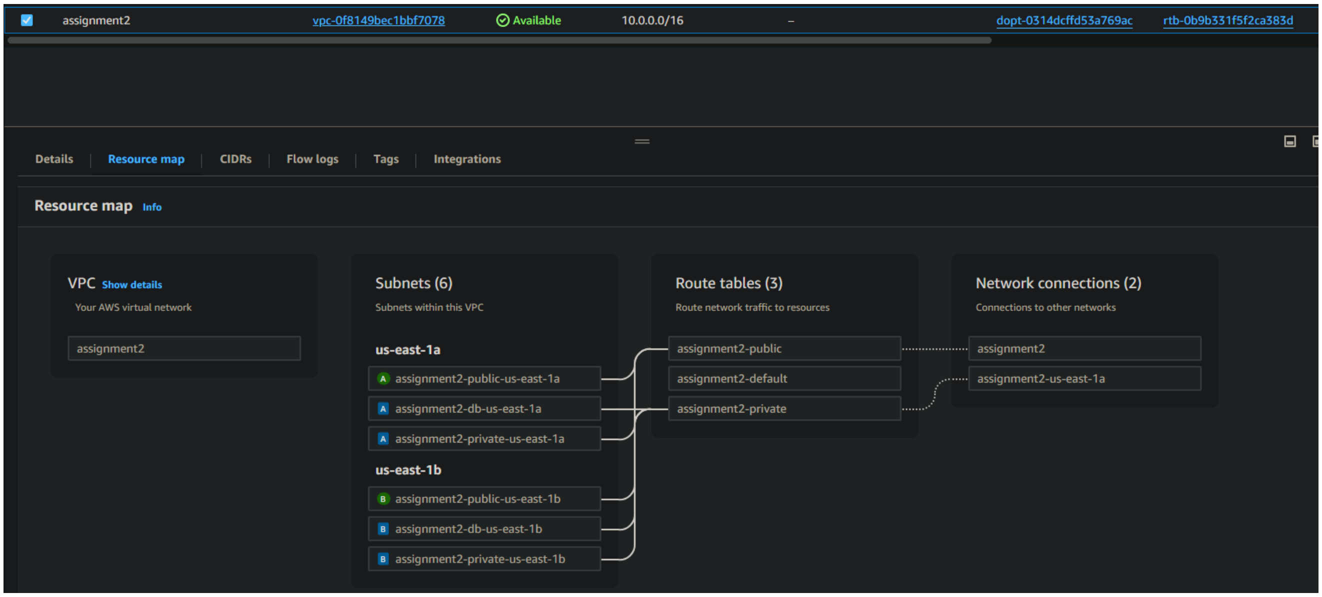
AWS VPC:
Our VPC variables are as follows:



```
# VPC variables
vpc_name             = "assignment2"
vpc_cidr             = "10.0.0.0/16"
vpc_azs              = ["us-east-1a", "us-east-1b"]
vpc_public_subnets   = ["10.0.1.0/24", "10.0.2.0/24"]
vpc_private_subnets  = ["10.0.11.0/24", "10.0.12.0/24"]
vpc_database_subnets = ["10.0.21.0/24", "10.0.22.0/24"]
vpc_tags             = { "created-by" = "terraform" }
```

Figure 18: VPC creation

We can use AWS GUI to confirm the completeness of this task as follows:



☑ assignment2 vpc-0f8149bec1bbf7078 Available 10.0.0.0/16 – dopt-0314dcffd53a769ac rtb-0b9b331f5f2ca383d

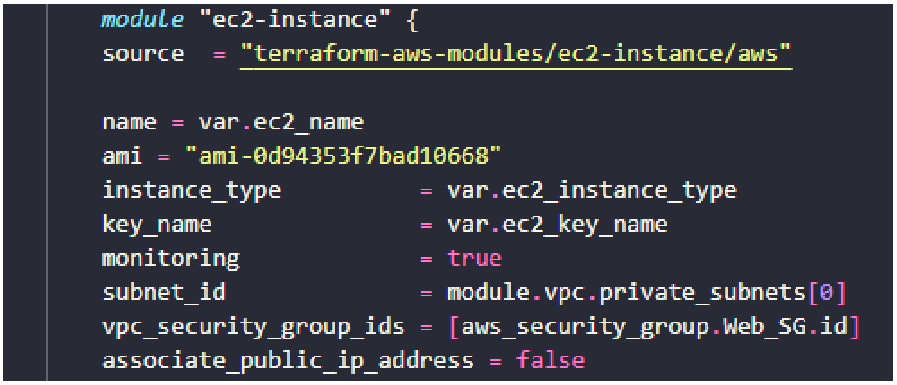
Figure 19: AWS VPC on AWS GUI

AWS EC2:

The EC2 Instance for this task is not so different from the Instance in Task 2.1P, however, there are some modifications:

+ EC2's subnet: we will move our instance to a private subnet and attached it to the target group of the ALB.
+ EC2's security group: we will only allow HTTP traffic to the Instance from the ALB.
+ EC2's user data: as we have removed the MariaDB on the Instance, we will make some modification onto the user data scripts.

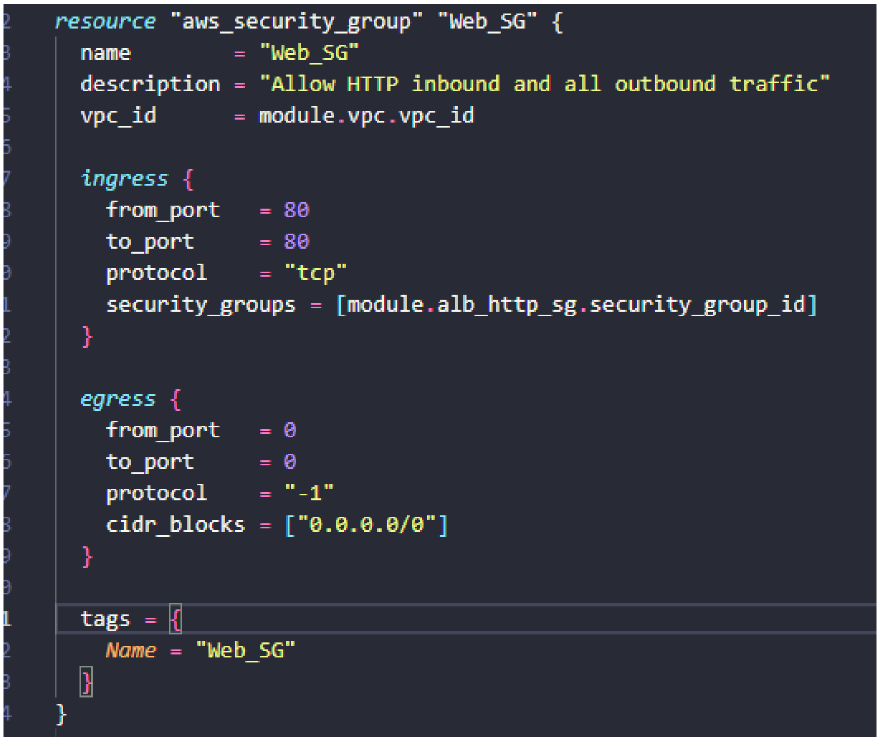
First, we will move our instance to a private subnet and disable its public IP address.



```
module "ec2-instance" {
source  = "terraform-aws-modules/ec2-instance/aws"

name = var.ec2_name
ami = "ami-0d94353f7bad10668"
instance_type           = var.ec2_instance_type
key_name                = var.ec2_key_name
monitoring              = true
subnet_id               = module.vpc.private_subnets[0]
vpc_security_group_ids = [aws_security_group.Web_SG.id]
associate_public_ip_address = false
```

Figure 20: Subnet modification

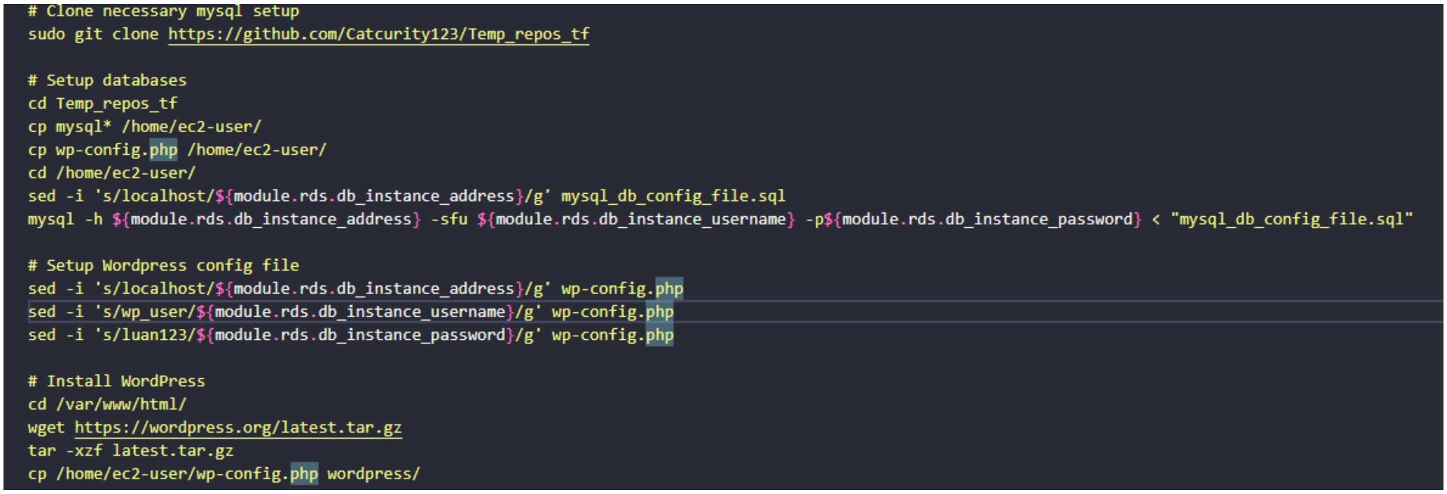
Then we will make some changes to the EC2's security group.



```
resource "aws_security_group" "Web_SG" {
  name        = "Web_SG"
  description = "Allow HTTP inbound and all outbound traffic"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    security_groups = [module.alb_http_sg.security_group_id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "Web_SG"
  }
}
```

Figure 21: Security group modification

Finally, we will need to make some changes to the EC2's user data:

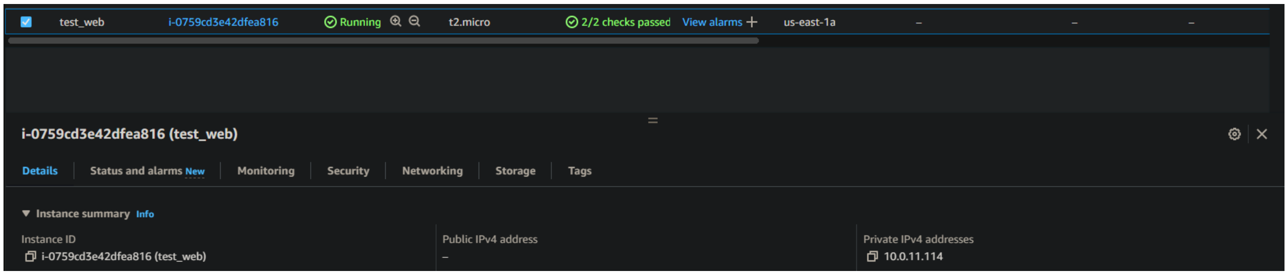


```
# Clone necessary mysql setup
sudo git clone https://github.com/Catcurity123/Temp_repos_tf

# Setup databases
cd Temp_repos_tf
cp mysql* /home/ec2-user/
cp wp-config.php /home/ec2-user/
cd /home/ec2-user/
sed -i 's/localhost/${module.rds.db_instance_address}/g' mysql_db_config_file.sql
mysql -h ${module.rds.db_instance_address} -sfu ${module.rds.db_instance_username} -p${module.rds.db_instance_password} < "mysql_db_config_file.sql"

# Setup Wordpress config file
sed -i 's/localhost/${module.rds.db_instance_address}/g' wp-config.php
sed -i 's/wp_user/${module.rds.db_instance_username}/g' wp-config.php
sed -i 's/luan123/${module.rds.db_instance_password}/g' wp-config.php

# Install WordPress
cd /var/www/html/
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
cp /home/ec2-user/wp-config.php wordpress/
```

Figure 22: User data scripts modification

As now we are using an external database, configuration files for database and WordPress on the Instance also need to be modified, namely we will change the "localhost" to the RDS endpoint of our new RDS Instance. This is done automatically using Terraform variable syntax and Linux command "sed – stream editor". Additionally, we will also need to change the username and password of our WordPress configuration files to reflect new requirements. We can take a look at our provisioned EC2 Instance to see if our scripts work as expected.

We can see that our EC2 Instance is deployed on private subnet.



☑	test_web	i-0759cd3e42dfea816	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	–	–	–

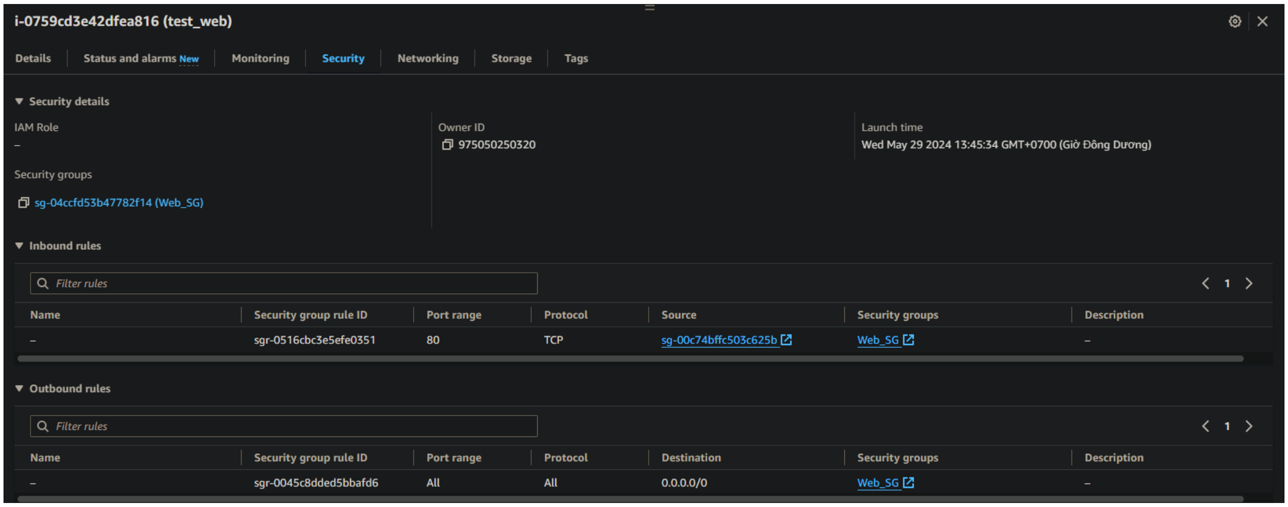
i-0759cd3e42dfea816 (test_web)

▼ Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0759cd3e42dfea816 (test_web)	–	10.0.11.114

Figure 23: EC2's subnet.

Our security group only accepts connections from the Application Load Balancer security group.



i-0759cd3e42dfea816 (test_web)

▼ Security details

IAM Role: –

Owner ID: 975050250320

Launch time: Wed May 29 2024 13:45:34 GMT+0700 (Giờ Đông Dương)

Security groups: sg-04ccfd53b47782f14 (Web_SG)

▼ Inbound rules

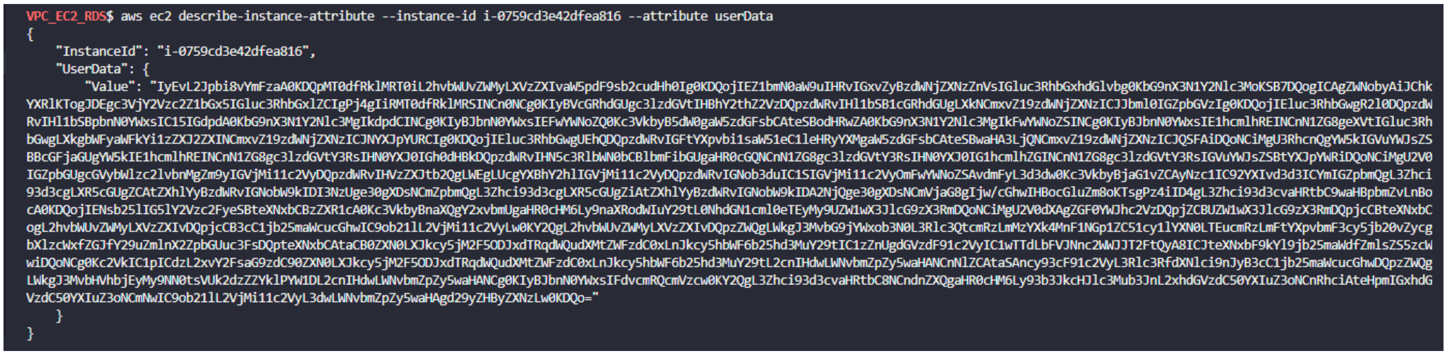
Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
–	sgr-0516cbc3e5efe0351	80	TCP	sg-00c74bffc503c625b	Web_SG	–

▼ Outbound rules

Name	Security group rule ID	Port range	Protocol	Destination	Security groups	Description
–	sgr-0045c8dded5bbafd6	All	All	0.0.0.0/0	Web_SG	–

Figure 24: EC2's Security Group.

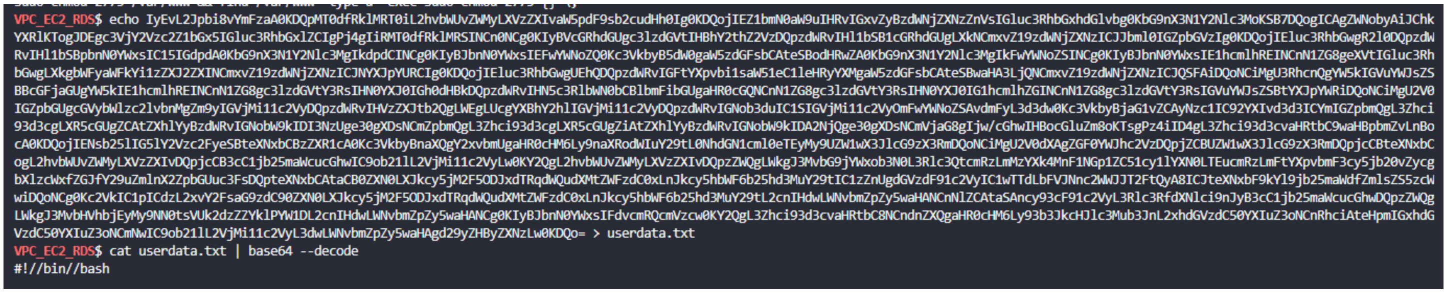
Finally, we will check our new user data, by using AWS CLI:



```
VPC_EC2_RDS$ aws ec2 describe-instance-attribute --instance-id i-0759cd3e42dfea816 --attribute userData
{
    "InstanceId": "i-0759cd3e42dfea816",
    "UserData": {
        "Value": "IyEvL2Jpbi8vYmFzaA0KDQpMT0dfRklMRT0iL2hvbWUvZWMyLXVzZXIvaW5pdF9sb2cudHh0Ig0KDQojIEZ1bmN0aW9uIHRvIGxvZyBzdWNjZXNzZnVsIGluc3RhbGxhdGlvbg0KbG9nX3N1Y2Nlc3MoKSB7DQogICAgZWNobyAiJChk
YXRlKTogJDEgc3VjY2Vzc2Z1bGx5IGluc3RhbGxlZCIgPj4gIiRMT0dfRklMRSINCn0NCg0KIyBVcGRhdGUgc3lzdGVtIHBhY2thZ2VzDQpzdWRvIHl1bSB1cGRhdGUgLXkNCmxvZ19zdWNjZXNzICJJbml0IGZpbGVzIg0KDQojIEluc3RhbGwgR2l0DQpzdW
RvIHl1bSBpbnN0YWxsIC15IGdpdA0KbG9nX3N1Y2Nlc3MgIkdpdCINCg0KIyBJbnN0YWxsIEFwYWNoZQ0Kc3VkbyB5dW0gaW5zdGFsbCAteSBodHRwZA0KbG9nX3N1Y2Nlc3MgIkFwYWNoZSINCg0KIyBJbnN0YWxsIE1hcmlhREINCnN1ZG8geXVtIGluc3Rh
bGwgLXkgbWFyaWFkYi1zZXJ2ZXINCmxvZ19zdWNjZXNzICJNYXJpYURCIg0KDQojIEluc3RhbGwgUEhQDQpzdWRvIGFtYXpvbi1saW51eC1leHRyYXMgaW5zdGFsbCAteSBwaHA3LjQNCmxvZ19zdWNjZXNzICJQSFAiDQoNCiMgU3RhcnQgYW5kIGVuYWJsZS
BBcGFjaGUgYW5kIE1hcmlhREINCnN1ZG8gc3lzdGVtY3RsIHN0YXJ0IGh0dHBkDQpzdWRvIHN5c3RlbWN0bCBlbmFibGUgaHR0cGQNCnN1ZG8gc3lzdGVtY3RsIHN0YXJ0IG1hcmlhZGINCnN1ZG8gc3lzdGVtY3RsIGVuYWJsZSBtYXJpYWRiDQoNCiMgU2V0
IGZpbGUgcGVybWlzc2lvbnMgZm9yIGVjMi11c2VyDQpzdWRvIHVzZXJtb2QgLWEgLUcgYXBhY2hlIGVjMi11c2VyDQpzdWRvIGNob3duIC1SIGVjMi11c2VyOmFwYWNoZSAvdmFyL3d3dw0Kc3VkbyBjaG1vZCAyNzc1IC92YXIvd3d3ICYmIGZpbmQgL3Zhci
93d3cgLXR5cGUgZCAtZXhlYyBzdWRvIGNobW9kIDI3NzUge30gXDsNCmZpbmQgL3Zhci93d3cgLXR5cGUgZiAtZXhlYyBzdWRvIGNobW9kIDA2NjQge30gXDsNCmVjaG8gIjw/cGhwIHBocGluZm8oKTsgPz4iID4gL3Zhci93d3cvaHRtbC9waHBpbmZvLnBo
cA0KDQojIENsb25lIG5lY2Vzc2FyeSBteXNxbCBzZXR1cA0Kc3VkbyBnaXQgY2xvbmUgaHR0cHM6Ly9naXRodWIuY29tL0NhdGN1cml0eTEyMy9UZW1wX3JlcG9zX3RmDQoNCiMgU2V0dXAgZGF0YWJhc2VzDQpjZCBUZW1wX3JlcG9zX3RmDQpjcCBteXNxbC
ogL2hvbWUvZWMyLXVzZXIvDQpjcCB3cC1jb25maWcucGhwIC9ob21lL2VjMi11c2VyLw0KY2QgL2hvbWUvZWMyLXVzZXIvDQpzZWQgLWkgJ3MvbG9jYWxob3N0L3Rlc3QtcmRzLmMzYXk4MnF1NGp1ZC51cy1lYXN0LTEucmRzLmFtYXpvbmF3cy5jb20vZycg
bXlzcWxfZGJfY29uZmlnX2ZpbGUuc3FsDQpteXNxbCAtaCB0ZXN0LXJkcy5jM2F5ODJxdTRqdWQudXMtZWFzdC0xLnJkcy5hbWF6b25hd3MuY29tIC1zZnUgdGVzdF91c2VyIC1wTTdLbFVJNnc2WWJJT2FtQyA8ICJteXNxbF9kYl9jb25maWdfZmlsZS5zcW
wiDQoNCg0Kc2VkIC1pICdzL2xvY2FsaG9zdC90ZXN0LXJkcy5jM2F5ODJxdTRqdWQudXMtZWFzdC0xLnJkcy5hbWF6b25hd3MuY29tL2cnIHdwLWNvbmZpZy5waHANCnNlZCAtaSAncy93cF91c2VyL3Rlc3RfdXNlci9nJyB3cC1jb25maWcucGhwDQpzZWQg
LWkgJ3MvbHVhbjEyMy9NN0tsVUk2dzZZYklPYW1DL2cnIHdwLWNvbmZpZy5waHANCg0KIyBJbnN0YWxsIFdvcmRQcmVzcw0KY2QgL3Zhci93d3cvaHRtbC8NCndnZXQgaHR0cHM6Ly93b3JkcHJlc3Mub3JnL2xhdGVzdC50YXIuZ3oNCnRhciAteHpmIGxhdG
VzdC50YXIuZ3oNCmNwIC9ob21lL2VjMi11c2VyL3dwLWNvbmZpZy5waHAgd29yZHByZXNzLw0KDQo="
    }
}
```

Figure 25: EC2's user data

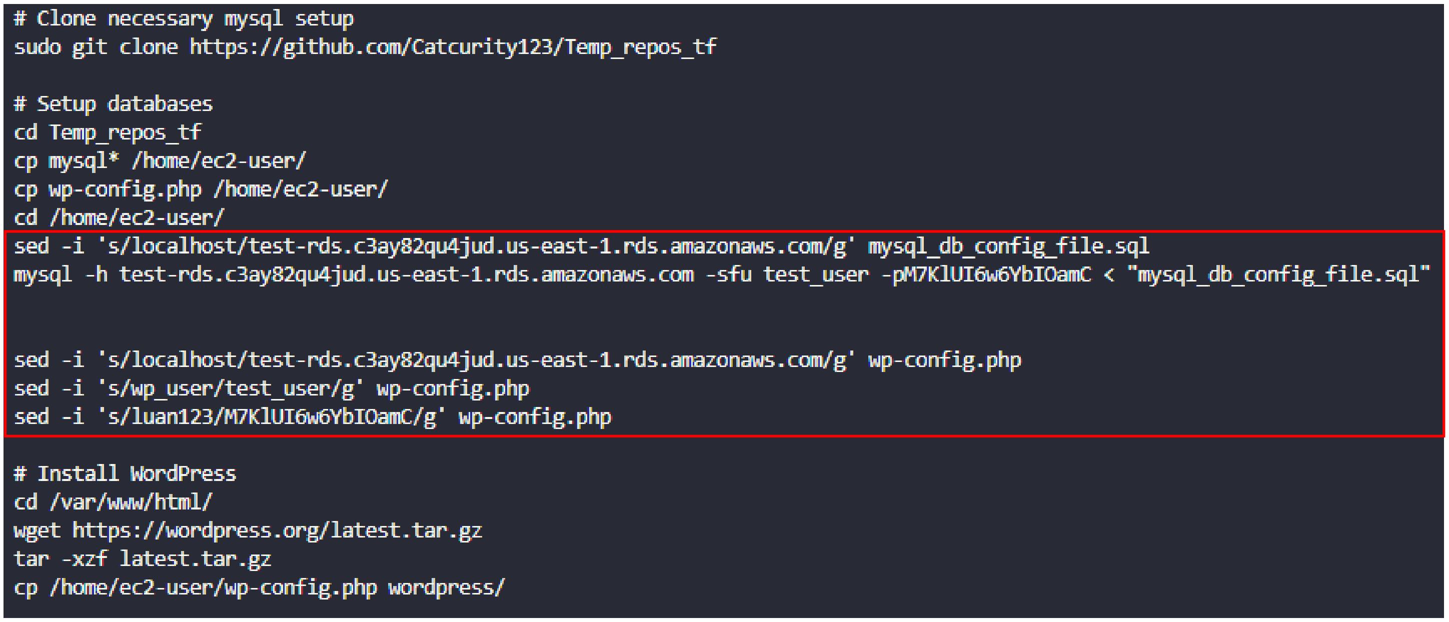
And we can decode it using Linux base64 command:



```
sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
VPC_EC2_RDS$ echo IyEvL2Jpbi8vYmFzaA0KDQpMT0dfRklMRT0iL2hvbWUvZWMyLXVzZXIvaW5pdF9sb2cudHh0Ig0KDQojIEZ1bmN0aW9uIHRvIGxvZyBzdWNjZXNzZnVsIGluc3RhbGxhdGlvbg0KbG9nX3N1Y2Nlc3MoKSB7DQogICAgZWNobyAiJChk
YXRlKTogJDEgc3VjY2Vzc2Z1bGx5IGluc3RhbGxlZCIgPj4gIiRMT0dfRklMRSINCn0NCg0KIyBVcGRhdGUgc3lzdGVtIHBhY2thZ2VzDQpzdWRvIHl1bSB1cGRhdGUgLXkNCmxvZ19zdWNjZXNzICJJbml0IGZpbGVzIg0KDQojIEluc3RhbGwgR2l0DQpzdW
RvIHl1bSBpbnN0YWxsIC15IGdpdA0KbG9nX3N1Y2Nlc3MgIkdpdCINCg0KIyBJbnN0YWxsIEFwYWNoZQ0Kc3VkbyB5dW0gaW5zdGFsbCAteSBodHRwZA0KbG9nX3N1Y2Nlc3MgIkFwYWNoZSINCg0KIyBJbnN0YWxsIE1hcmlhREINCnN1ZG8geXVtIGluc3Rh
bGwgLXkgbWFyaWFkYi1zZXJ2ZXINCmxvZ19zdWNjZXNzICJNYXJpYURCIg0KDQojIEluc3RhbGwgUEhQDQpzdWRvIGFtYXpvbi1saW51eC1leHRyYXMgaW5zdGFsbCAteSBwaHA3LjQNCmxvZ19zdWNjZXNzICJQSFAiDQoNCiMgU3RhcnQgYW5kIGVuYWJsZS
BBcGFjaGUgYW5kIE1hcmlhREINCnN1ZG8gc3lzdGVtY3RsIHN0YXJ0IGh0dHBkDQpzdWRvIHN5c3RlbWN0bCBlbmFibGUgaHR0cGQNCnN1ZG8gc3lzdGVtY3RsIHN0YXJ0IG1hcmlhZGINCnN1ZG8gc3lzdGVtY3RsIGVuYWJsZSBtYXJpYWRiDQoNCiMgU2V0
IGZpbGUgcGVybWlzc2lvbnMgZm9yIGVjMi11c2VyDQpzdWRvIHVzZXJtb2QgLWEgLUcgYXBhY2hlIGVjMi11c2VyDQpzdWRvIGNob3duIC1SIGVjMi11c2VyOmFwYWNoZSAvdmFyL3d3dw0Kc3VkbyBjaG1vZCAyNzc1IC92YXIvd3d3ICYmIGZpbmQgL3Zhci
93d3cgLXR5cGUgZCAtZXhlYyBzdWRvIGNobW9kIDI3NzUge30gXDsNCmZpbmQgL3Zhci93d3cgLXR5cGUgZiAtZXhlYyBzdWRvIGNobW9kIDA2NjQge30gXDsNCmVjaG8gIjw/cGhwIHBocGluZm8oKTsgPz4iID4gL3Zhci93d3cvaHRtbC9waHBpbmZvLnBo
cA0KDQojIENsb25lIG5lY2Vzc2FyeSBteXNxbCBzZXR1cA0Kc3VkbyBnaXQgY2xvbmUgaHR0cHM6Ly9naXRodWIuY29tL0NhdGN1cml0eTEyMy9UZW1wX3JlcG9zX3RmDQoNCiMgU2V0dXAgZGF0YWJhc2VzDQpjZCBUZW1wX3JlcG9zX3RmDQpjcCBteXNxbC
ogL2hvbWUvZWMyLXVzZXIvDQpjcCB3cC1jb25maWcucGhwIC9ob21lL2VjMi11c2VyLw0KY2QgL2hvbWUvZWMyLXVzZXIvDQpzZWQgLWkgJ3MvbG9jYWxob3N0L3Rlc3QtcmRzLmMzYXk4MnF1NGp1ZC51cy1lYXN0LTEucmRzLmFtYXpvbmF3cy5jb20vZycg
bXlzcWxfZGJfY29uZmlnX2ZpbGUuc3FsDQpteXNxbCAtaCB0ZXN0LXJkcy5jM2F5ODJxdTRqdWQudXMtZWFzdC0xLnJkcy5hbWF6b25hd3MuY29tIC1zZnUgdGVzdF91c2VyIC1wTTdLbFVJNnc2WWJJT2FtQyA8ICJteXNxbF9kYl9jb25maWdfZmlsZS5zcW
wiDQoNCg0Kc2VkIC1pICdzL2xvY2FsaG9zdC90ZXN0LXJkcy5jM2F5ODJxdTRqdWQudXMtZWFzdC0xLnJkcy5hbWF6b25hd3MuY29tL2cnIHdwLWNvbmZpZy5waHANCnNlZCAtaSAncy93cF91c2VyL3Rlc3RfdXNlci9nJyB3cC1jb25maWcucGhwDQpzZWQg
LWkgJ3MvbHVhbjEyMy9NN0tsVUk2dzZZYklPYW1DL2cnIHdwLWNvbmZpZy5waHANCg0KIyBJbnN0YWxsIFdvcmRQcmVzcw0KY2QgL3Zhci93d3cvaHRtbC8NCndnZXQgaHR0cHM6Ly93b3JkcHJlc3Mub3JnL2xhdGVzdC50YXIuZ3oNCnRhciAteHpmIGxhdG
VzdC50YXIuZ3oNCmNwIC9ob21lL2VjMi11c2VyL3dwLWNvbmZpZy5waHAgd29yZHByZXNzLw0KDQo= > userdata.txt
VPC_EC2_RDS$ cat userdata.txt | base64 --decode
#!//bin//bash
```

Figure 26: Decoded the base64 user data.

We can see that our script works as expected, as the RDS endpoint and RDS credentials are pasted onto WordPress configuration files.

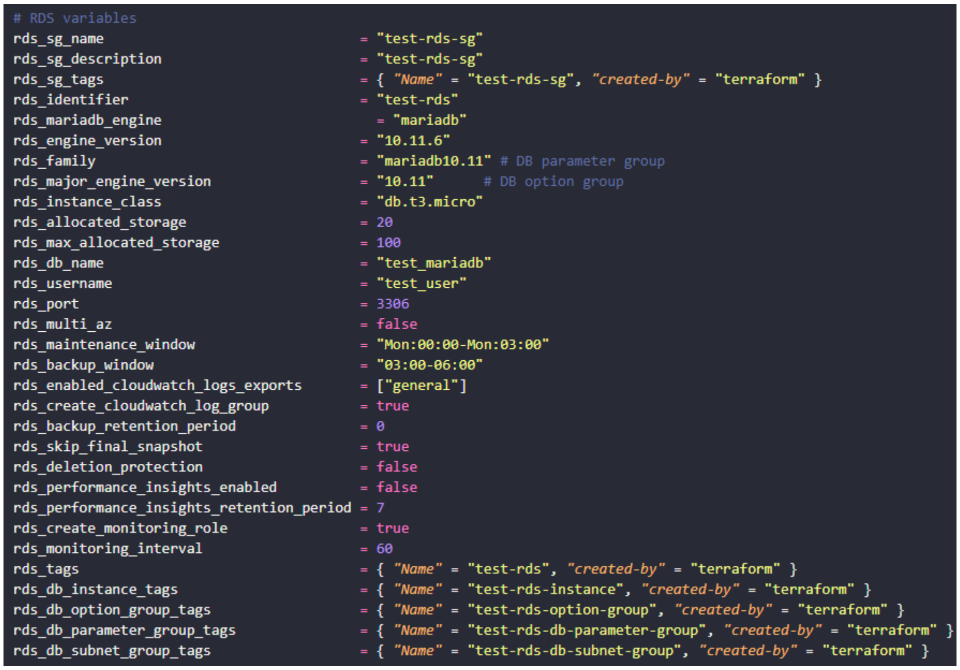


```
# Clone necessary mysql setup
sudo git clone https://github.com/Catcurity123/Temp_repos_tf

# Setup databases
cd Temp_repos_tf
cp mysql* /home/ec2-user/
cp wp-config.php /home/ec2-user/
cd /home/ec2-user/
sed -i 's/localhost/test-rds.c3ay82qu4jud.us-east-1.rds.amazonaws.com/g' mysql_db_config_file.sql
mysql -h test-rds.c3ay82qu4jud.us-east-1.rds.amazonaws.com -sfu test_user -pM7KlUI6w6YbIOamC < "mysql_db_config_file.sql"


sed -i 's/localhost/test-rds.c3ay82qu4jud.us-east-1.rds.amazonaws.com/g' wp-config.php
sed -i 's/wp_user/test_user/g' wp-config.php
sed -i 's/luan123/M7KlUI6w6YbIOamC/g' wp-config.php

# Install WordPress
cd /var/www/html/
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
cp /home/ec2-user/wp-config.php wordpress/
```

Figure 27: Modified user data

AWS RDS:

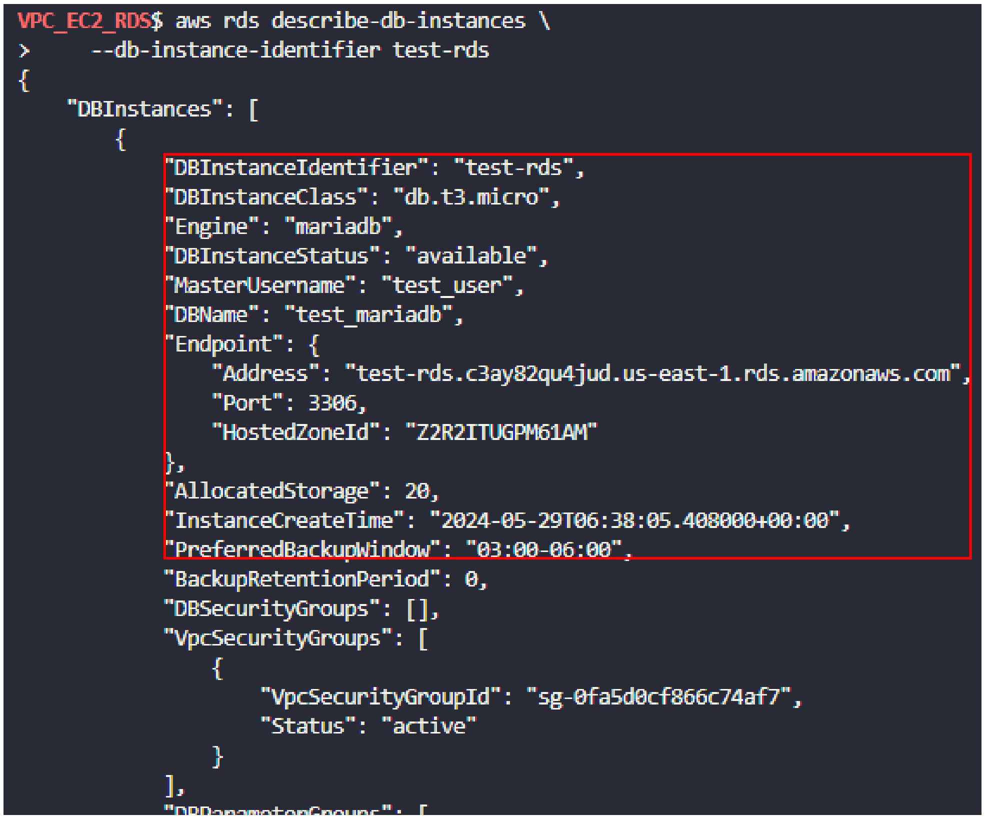
Our AWS RDS variables are as follows:



```
# RDS variables
rds_sg_name                               = "test-rds-sg"
rds_sg_description                        = "test-rds-sg"
rds_sg_tags                               = { "Name" = "test-rds-sg", "created-by" = "terraform" }
rds_identifier                            = "test-rds"
rds_mariadb_engine                          = "mariadb"
rds_engine_version                        = "10.11.6"
rds_family                                = "mariadb10.11" # DB parameter group
rds_major_engine_version                  = "10.11"       # DB option group
rds_instance_class                        = "db.t3.micro"
rds_allocated_storage                     = 20
rds_max_allocated_storage                 = 100
rds_db_name                               = "test_mariadb"
rds_username                              = "test_user"
rds_port                                  = 3306
rds_multi_az                              = false
rds_maintenance_window                    = "Mon:00:00-Mon:03:00"
rds_backup_window                         = "03:00-06:00"
rds_enabled_cloudwatch_logs_exports       = ["general"]
rds_create_cloudwatch_log_group           = true
rds_backup_retention_period               = 0
rds_skip_final_snapshot                   = true
rds_deletion_protection                   = false
rds_performance_insights_enabled          = false
rds_performance_insights_retention_period = 7
rds_create_monitoring_role                = true
rds_monitoring_interval                   = 60
rds_tags                                  = { "Name" = "test-rds", "created-by" = "terraform" }
rds_db_instance_tags                      = { "Name" = "test-rds-instance", "created-by" = "terraform" }
rds_db_option_group_tags                  = { "Name" = "test-rds-option-group", "created-by" = "terraform" }
rds_db_parameter_group_tags               = { "Name" = "test-rds-db-parameter-group", "created-by" = "terraform" }
rds_db_subnet_group_tags                  = { "Name" = "test-rds-db-subnet-group", "created-by" = "terraform" }
```

Figure 28: AWS RDS variables

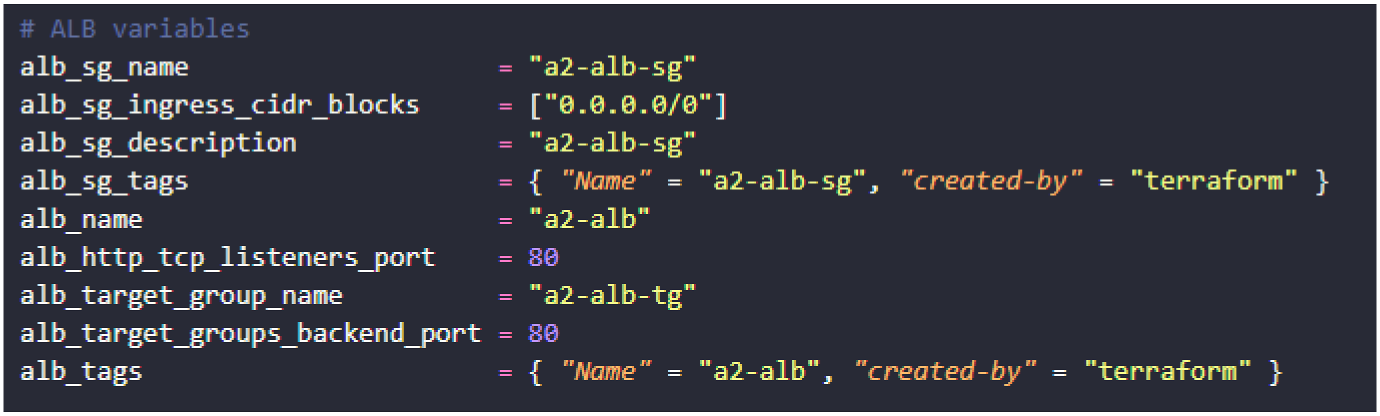
We, again, use AWS CLI to verify our installation.



```
VPC_EC2_RDS$ aws rds describe-db-instances \
>     --db-instance-identifier test-rds
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "test-rds",
            "DBInstanceClass": "db.t3.micro",
            "Engine": "mariadb",
            "DBInstanceStatus": "available",
            "MasterUsername": "test_user",
            "DBName": "test_mariadb",
            "Endpoint": {
                "Address": "test-rds.c3ay82qu4jud.us-east-1.rds.amazonaws.com",
                "Port": 3306,
                "HostedZoneId": "Z2R2ITUGPM61AM"
            },
            "AllocatedStorage": 20,
            "InstanceCreateTime": "2024-05-29T06:38:05.408000+00:00",
            "PreferredBackupWindow": "03:00-06:00",
            "BackupRetentionPeriod": 0,
            "DBSecurityGroups": [],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-0fa5d0cf866c74af7",
                    "Status": "active"
                }
            ],
            "DBParameterGroups": [
```

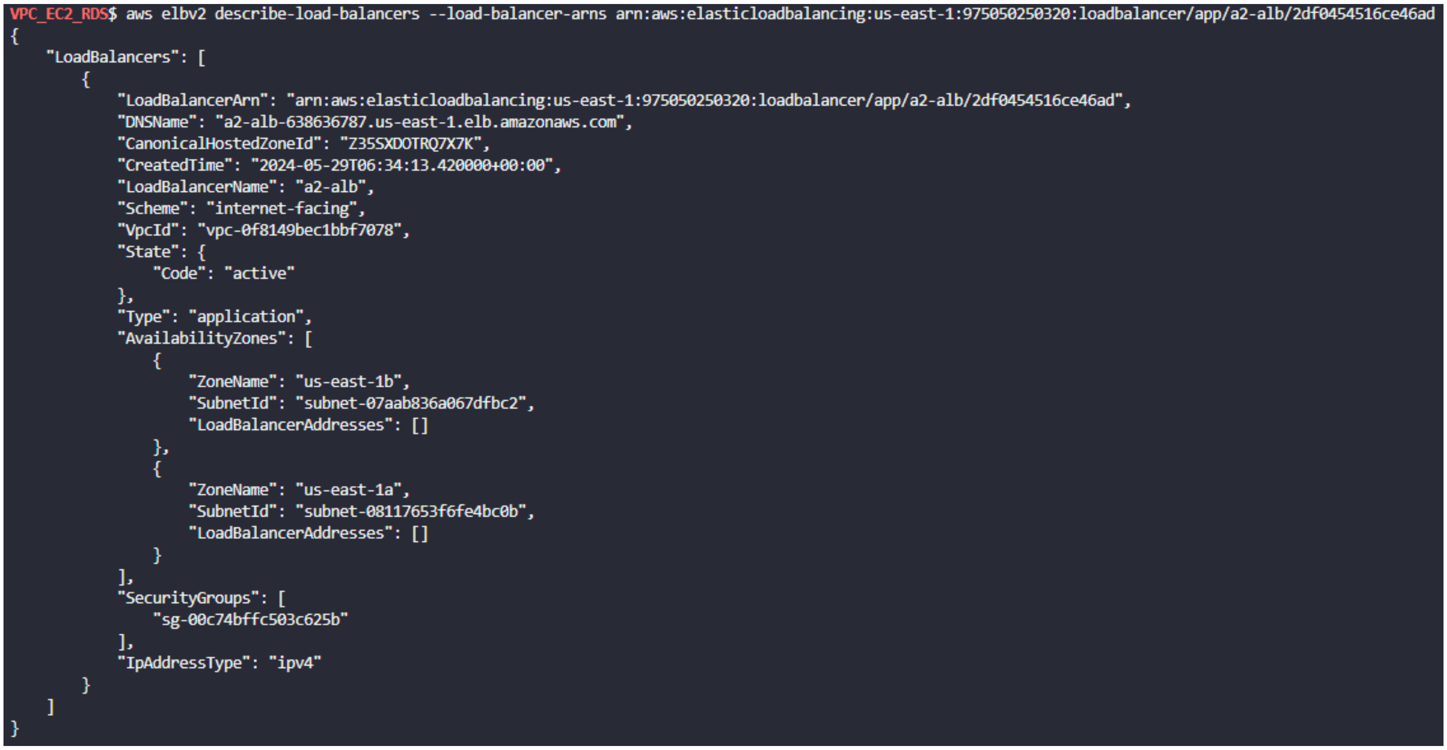
Figure 29: AWS RDS on AWS CLI

AWS Application Load Balancer:
Our AWS Application Load Balancer are as follows:



```
# ALB variables
alb_sg_name                     = "a2-alb-sg"
alb_sg_ingress_cidr_blocks      = ["0.0.0.0/0"]
alb_sg_description              = "a2-alb-sg"
alb_sg_tags                     = { "Name" = "a2-alb-sg", "created-by" = "terraform" }
alb_name                        = "a2-alb"
alb_http_tcp_listeners_port     = 80
alb_target_group_name           = "a2-alb-tg"
alb_target_groups_backend_port  = 80
alb_tags                        = { "Name" = "a2-alb", "created-by" = "terraform" }
```

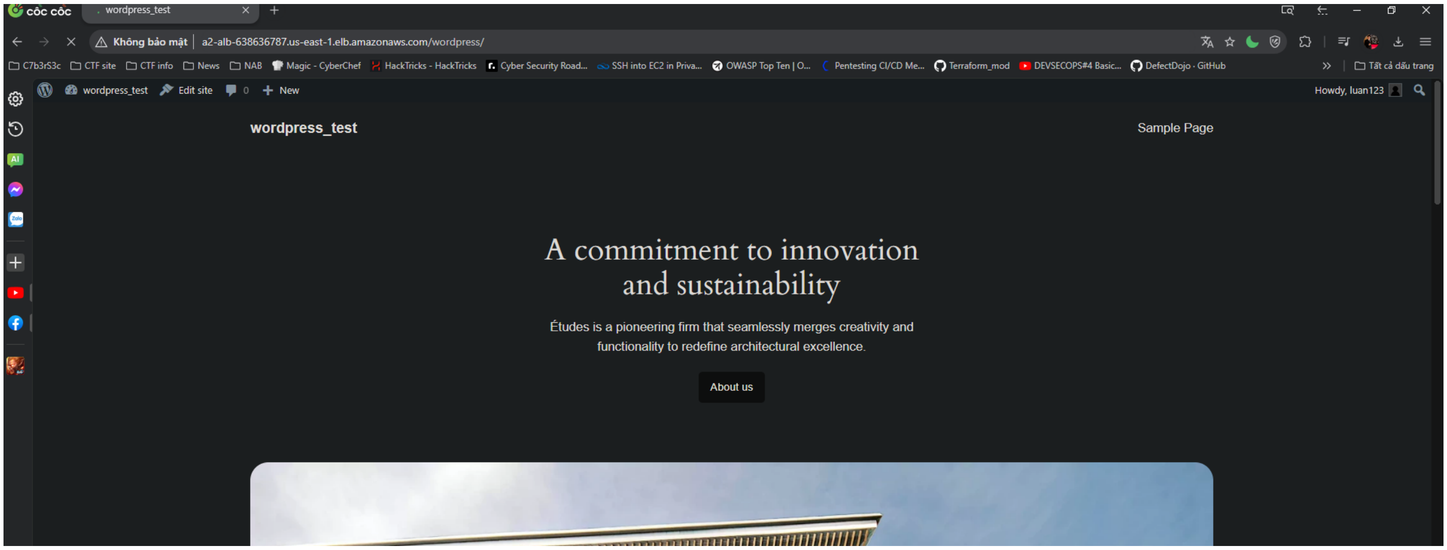
Figure 30: AWS ALB variables



```
VPC_EC2_RDS$ aws elbv2 describe-load-balancers --load-balancer-arns arn:aws:elasticloadbalancing:us-east-1:975050250320:loadbalancer/app/a2-alb/2df0454516ce46ad
{
    "LoadBalancers": [
        {
            "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:975050250320:loadbalancer/app/a2-alb/2df0454516ce46ad",
            "DNSName": "a2-alb-638636787.us-east-1.elb.amazonaws.com",
            "CanonicalHostedZoneId": "Z35SXDOTRQ7X7K",
            "CreatedTime": "2024-05-29T06:34:13.420000+00:00",
            "LoadBalancerName": "a2-alb",
            "Scheme": "internet-facing",
            "VpcId": "vpc-0f8149bec1bbf7078",
            "State": {
                "Code": "active"
            },
            "Type": "application",
            "AvailabilityZones": [
                {
                    "ZoneName": "us-east-1b",
                    "SubnetId": "subnet-07aab836a067dfbc2",
                    "LoadBalancerAddresses": []
                },
                {
                    "ZoneName": "us-east-1a",
                    "SubnetId": "subnet-08117653f6fe4bc0b",
                    "LoadBalancerAddresses": []
                }
            ],
            "SecurityGroups": [
                "sg-00c74bffc503c625b"
            ],
            "IpAddressType": "ipv4"
        }
    ]
}
```

Figure 31: AWS ALB on AWS CLI

With all the configurations done, we can use the ALB’s DNS to see if our website works as expected.



Không bảo mật | a2-alb-638636787.us-east-1.elb.amazonaws.com/wordpress/

wordpress_test

Sample Page

A commitment to innovation and sustainability

Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

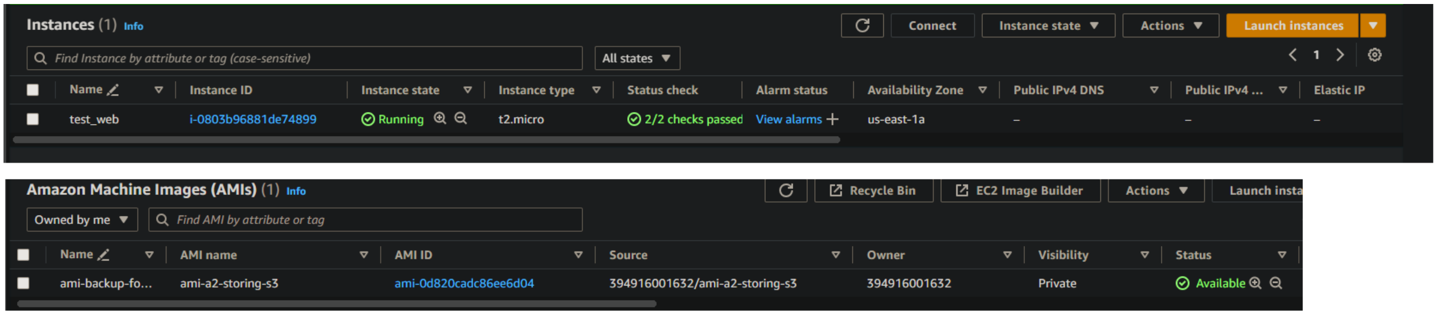
About us

Figure 32: WordPress Instance on an external database and load balancer.

Our WordPress Instance works perfectly with the Application Load Balancer, this will also conclude this task. We will move on to create an AMI and test store and restore ability on Amazon S3 bucket.

2.2B. Backup instance to S3 and create an instance from S3 backup.

First we will create an Amazon Machine Image (AMI) of our running instance.



Instances (1) Info

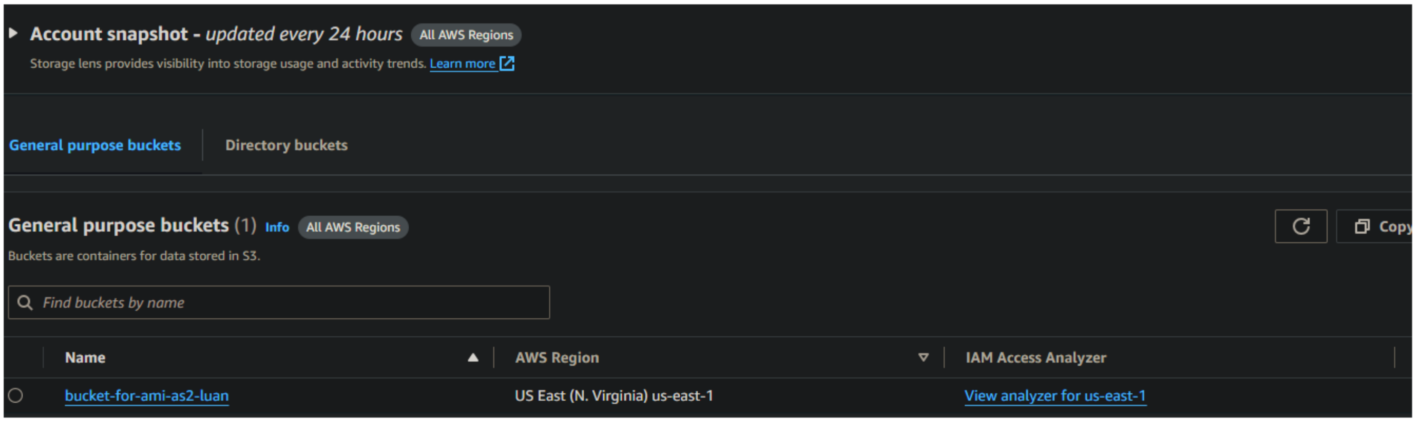
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
test_web	i-0803b96881de74899	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	–	–	–

Amazon Machine Images (AMIs) (1) Info

Name	AMI name	AMI ID	Source	Owner	Visibility	Status
ami-backup-fo...	ami-a2-storing-s3	ami-0d820cadc86ee6d04	394916001632/ami-a2-storing-s3	394916001632	Private	Available

Figure 33: AMI of the running instance

Next we will provision an S3 bucket to store our AMI.



Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets | Directory buckets

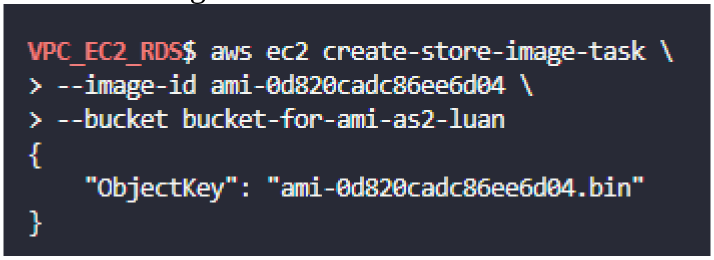
General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer
bucket-for-ami-as2-luan	US East (N. Virginia) us-east-1	View analyzer for us-east-1

Figure 34: S3 bucket for AMI

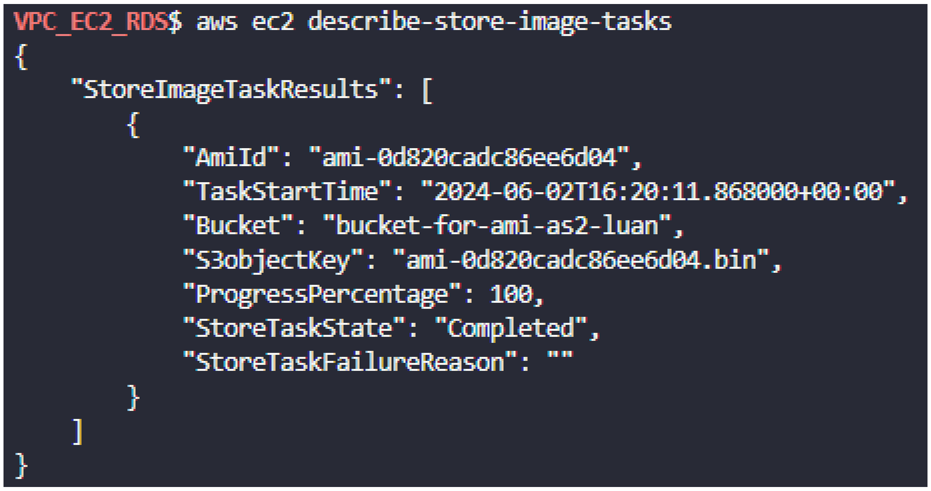
Next we can use AWS CLI to make an image task in our S3 Bucket.



```
VPC_EC2_RDS$ aws ec2 create-store-image-task \
> --image-id ami-0d820cadc86ee6d04 \
> --bucket bucket-for-ami-as2-luan
{
    "ObjectKey": "ami-0d820cadc86ee6d04.bin"
}
```

Figure 35: Create image task for AMI

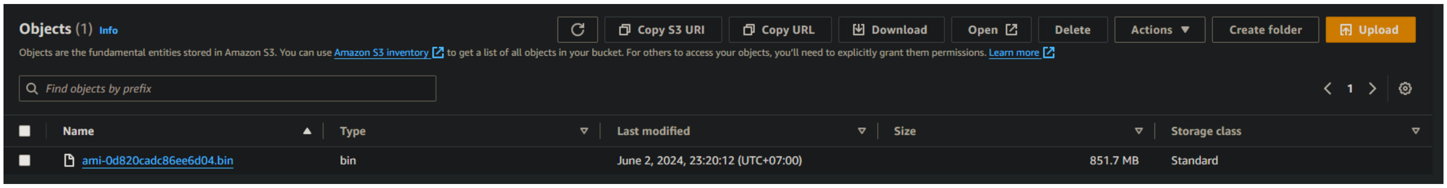
We can verify whether our task is executed correctly or note as follows:



```
VPC_EC2_RDS$ aws ec2 describe-store-image-tasks
{
    "StoreImageTaskResults": [
        {
            "AmiId": "ami-0d820cadc86ee6d04",
            "TaskStartTime": "2024-06-02T16:20:11.868000+00:00",
            "Bucket": "bucket-for-ami-as2-luan",
            "S3objectKey": "ami-0d820cadc86ee6d04.bin",
            "ProgressPercentage": 100,
            "StoreTaskState": "Completed",
            "StoreTaskFailureReason": ""
        }
    ]
}
```

Figure 36: Verify AMI stored in S3

We can also see our AMI created in S3 bucket.



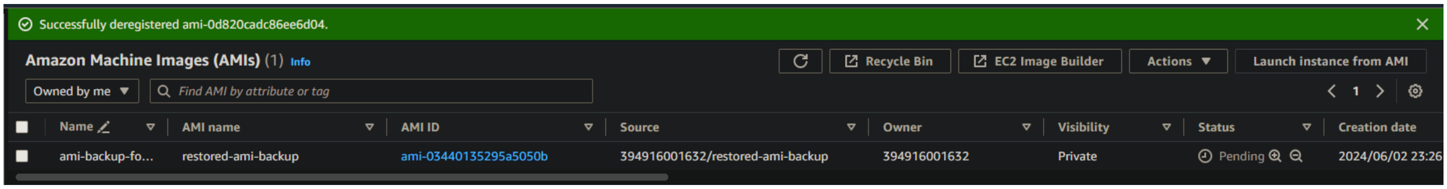
Objects (1) Info

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

□	Name ▲	Type	Last modified	Size	Storage class
□	ami-0d820cadc86ee6d04.bin	bin	June 2, 2024, 23:20:12 (UTC+07:00)	851.7 MB	Standard

Figure 37: Verify again on AWS GUI

This is done for storing backup instane to S3, now we will delete our instance and the previously created AMI and restore it from the AMI stored in S3 Bucket.

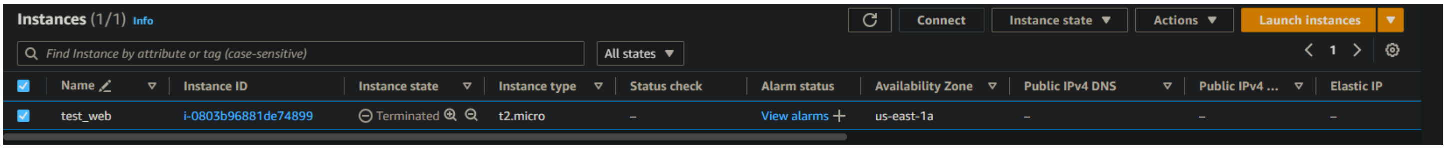


Successfully deregistered ami-0d820cadc86ee6d04.

Amazon Machine Images (AMIs) (1) Info

□	Name	AMI name	AMI ID	Source	Owner	Visibility	Status	Creation date
□	ami-backup-fo...	restored-ami-backup	ami-03440135295a5050b	394916001632/restored-ami-backup	394916001632	Private	Pending	2024/06/02 23:26

Figure 38: Delete created AMI

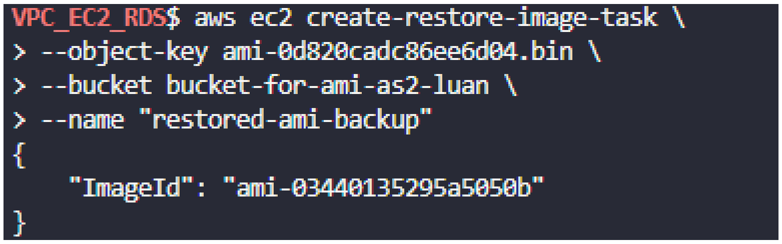


Instances (1/1) Info

☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
☑	test_web	i-0803b96881de74899	Terminated	t2.micro	–	View alarms +	us-east-1a	–	–	–

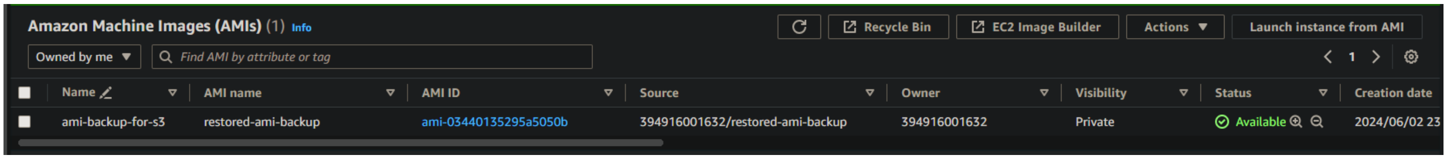
Figure 39: Terminate EC2 Instance

Now, again, we will use AWS CLI to restore our AMI.



```
VPC_EC2_RDS$ aws ec2 create-restore-image-task \
> --object-key ami-0d820cadc86ee6d04.bin \
> --bucket bucket-for-ami-as2-luan \
> --name "restored-ami-backup"
{
    "ImageId": "ami-03440135295a5050b"
}
```

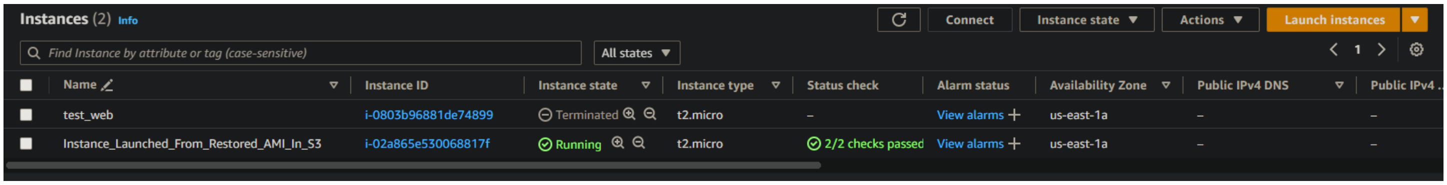
Figure 40: AMI Successfully restored.



□	Name	AMI name	AMI ID	Source	Owner	Visibility	Status	Creation date
□	ami-backup-for-s3	restored-ami-backup	ami-03440135295a5050b	394916001632/restored-ami-backup	394916001632	Private	✓ Available	2024/06/02 23

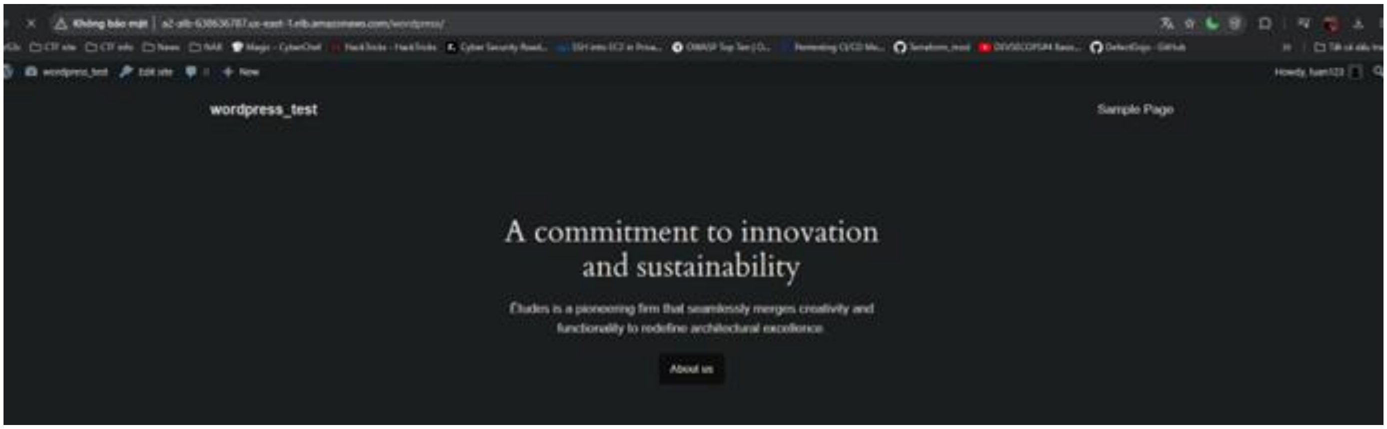
Figure 41: AMI successfully restored on AWS GUI.

We have successfully restored out AMI from S3, now we will relaunch our EC2 Instance to see if it is still working as expected.



□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
□	test_web	i-0803b96881de74899	Terminated	t2.micro	–	View alarms +	us-east-1a	–	–
□	Instance_Launched_From_Restored_AMI_In_S3	i-02a865e530068817f	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +	us-east-1a	–	–

Figure 42: Instance successfully relaunched.



A commitment to innovation and sustainability

Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

Figure 43: Instance functionality successfully restored

We can now be sure that our website is restored with sufficient functionality using the AMI stored in S3 Bucket. This will also conclude this task. We will move on to add more services for our system in Task 2.3D

Task 2.3D: By following AWS documentation on the links provided complete the following tasks

2.3 Create a Launch Template, Create an ASG, and configure ASG to launch instance.

We will also use Terraform for this task; our diagram above can be redrawn as follows:

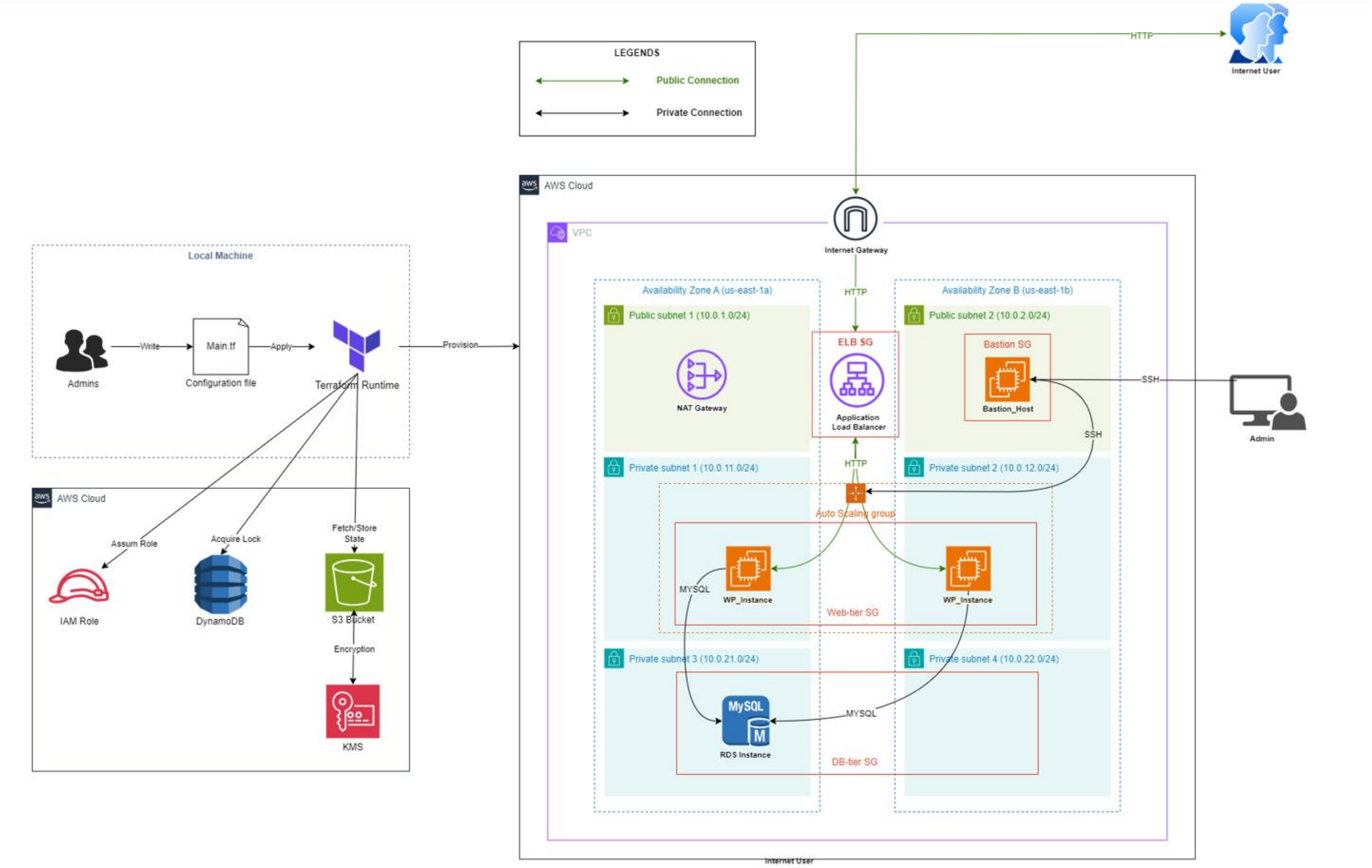


Figure 44: Final diagram for WordPress Instance

Building on the foundation of Task 2.2C, this project will streamline the provisioning of EC2 instances by leveraging **Launch Templates** and **Auto Scaling Groups**, thereby automating the process. The use of Launch Templates will establish a framework for the necessary instance settings, allowing for more efficient and consistent instance provisioning through Auto Scaling Groups. This approach ensures that AWS will automatically manage the scaling and deployment of EC2 instances as required.

To enhance security, we will implement comprehensive security measures, including the addition of **Security Groups** to all services. This will help to control traffic flow and protect our infrastructure. Additionally, all functional services will be migrated to a private subnet to further safeguard against unauthorized access.

Furthermore, we will introduce a **Bastion Host**. This host will serve as a secure entry point for administrators needing to access the private subnet for configuration and management purposes. This will enable secure, controlled access to our system's backend, maintaining the integrity and security of our network.

All of the infrastructure provisioning above will be conducted using Terraform.

This task will be divided into 4 main objectives:

+ **The configuration of Launch Template and Auto Scaling Group for WordPress Instance:** this task involves creating a Launch Template with a specified image ID and instance type. This template will be used in conjunction with our Auto Scaling Group to enable system scaling. The Auto Scaling Group will maintain a minimum of two instances at all times and can scale up to four instances in response to increased user usage. The scaling group will be located in a private subnet, and connections to our instances will be routed through the Application Load Balancer created in Task 2.2C.

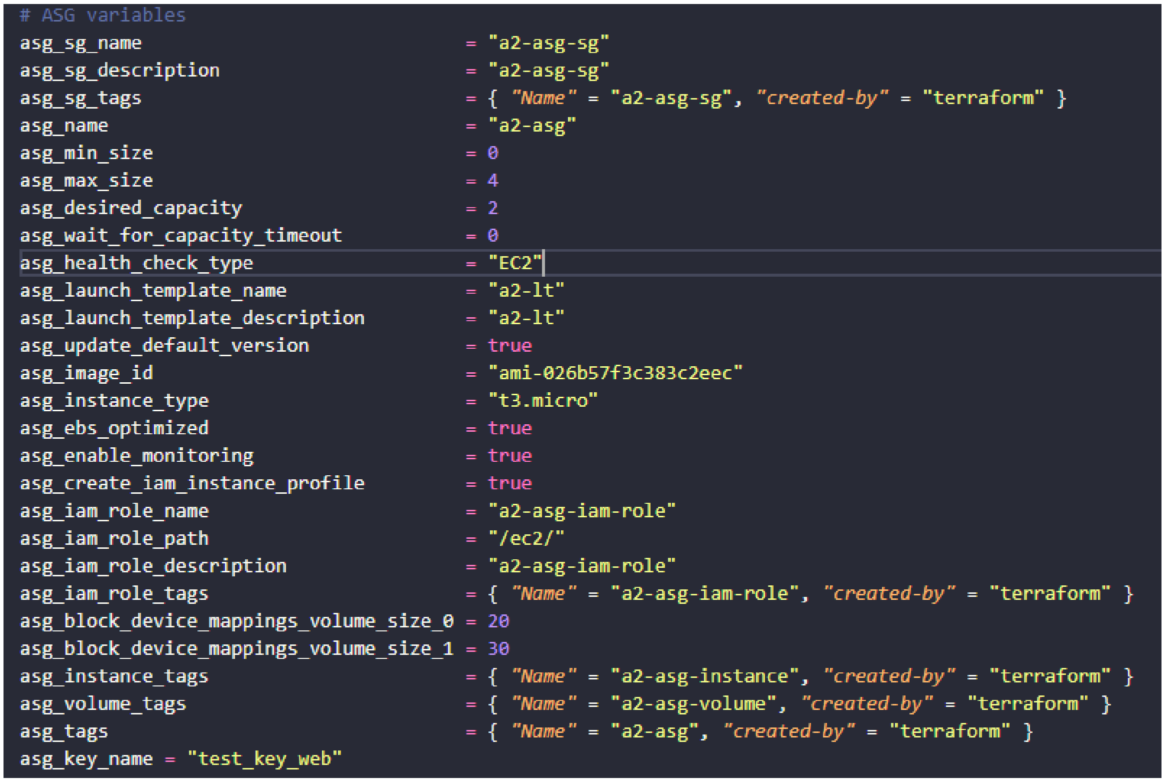
+ **The creation of a bastion host for secure configurational entry point**: we will deploy a Bastion Host in our public subnet, this instance will only accept SSH connection using key pair created in task 2.1P. Connection to private subnet instances can only be established through this instance.

+ **Apply additional security measures for our services**: for the final touch we will apply appropriate security groups for our services to ensure no malicious entry can penetrate our system.

+ **Scaling and functionality testing**: upon the completion of all the above tasks, we will proceed to test our configuration to see if it works as expected.

The configuration of Launch Template and Auto Scaling Group for WordPress Instance

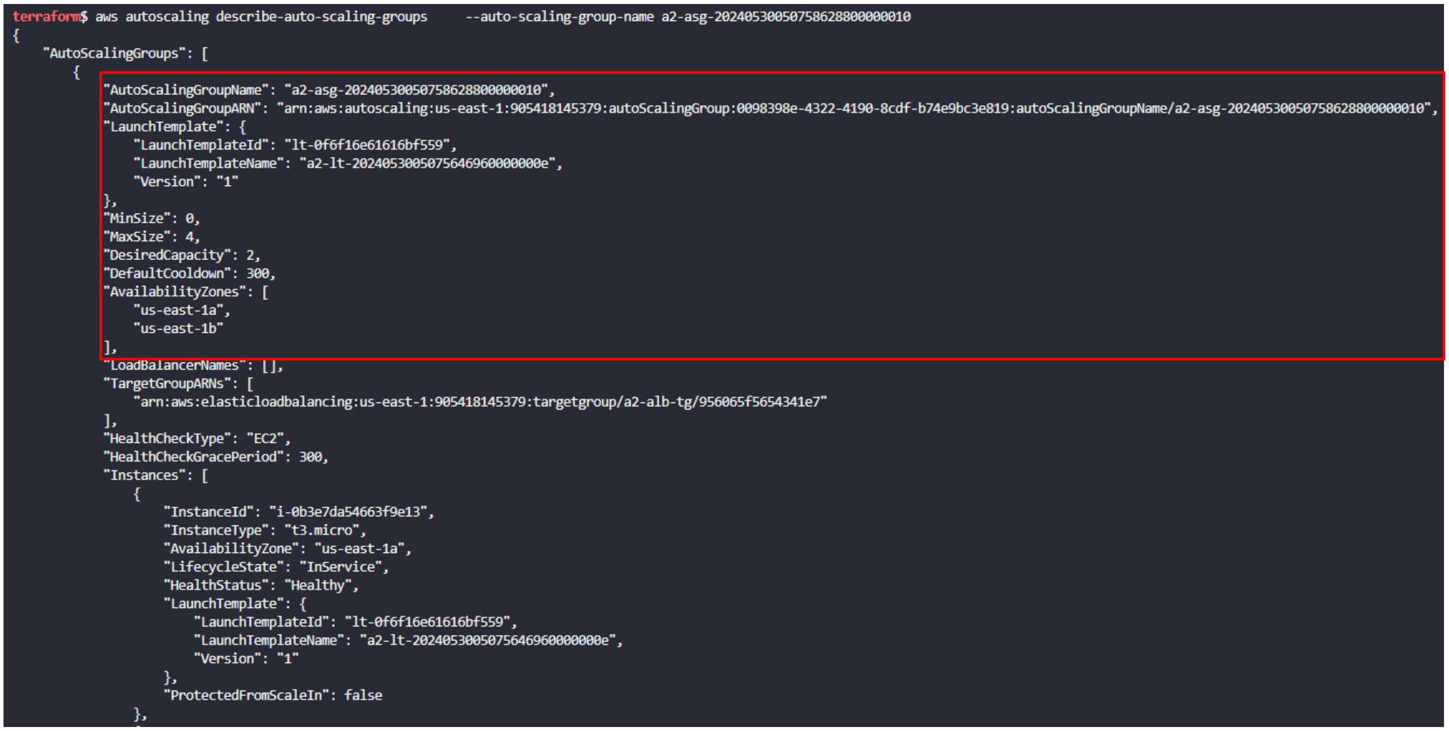
As our functionality for each EC2 Instance remains the same as it is in Task 2.2C, our user data would not need any additional configuration, however, as we will no longer manually deploy each EC2 Instance, we will have to specify a Launch Template for our Auto Scaling Group.



```
# ASG variables
asg_sg_name                             = "a2-asg-sg"
asg_sg_description                      = "a2-asg-sg"
asg_sg_tags                             = { "Name" = "a2-asg-sg", "created-by" = "terraform" }
asg_name                                = "a2-asg"
asg_min_size                            = 0
asg_max_size                            = 4
asg_desired_capacity                    = 2
asg_wait_for_capacity_timeout           = 0
asg_health_check_type                   = "EC2"
asg_launch_template_name                = "a2-lt"
asg_launch_template_description         = "a2-lt"
asg_update_default_version              = true
asg_image_id                            = "ami-026b57f3c383c2eec"
asg_instance_type                       = "t3.micro"
asg_ebs_optimized                       = true
asg_enable_monitoring                   = true
asg_create_iam_instance_profile         = true
asg_iam_role_name                       = "a2-asg-iam-role"
asg_iam_role_path                       = "/ec2/"
asg_iam_role_description                = "a2-asg-iam-role"
asg_iam_role_tags                       = { "Name" = "a2-asg-iam-role", "created-by" = "terraform" }
asg_block_device_mappings_volume_size_0 = 20
asg_block_device_mappings_volume_size_1 = 30
asg_instance_tags                       = { "Name" = "a2-asg-instance", "created-by" = "terraform" }
asg_volume_tags                         = { "Name" = "a2-asg-volume", "created-by" = "terraform" }
asg_tags                                = { "Name" = "a2-asg", "created-by" = "terraform" }
asg_key_name = "test_key_web"
```

Figure 45: Auto Scaling Group and Launch Template variables.

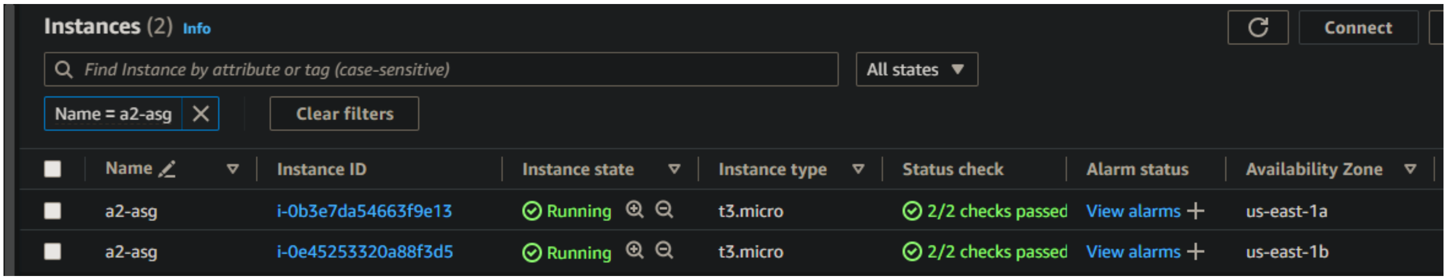
We can then verify this configuration on AWS CLI as follows:



```
terraform$ aws autoscaling describe-auto-scaling-groups     --auto-scaling-group-name a2-asg-20240530050758628800000010
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "a2-asg-20240530050758628800000010",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-east-1:905418145379:autoScalingGroup:0098398e-4322-4190-8cdf-b74e9bc3e819:autoScalingGroupName/a2-asg-20240530050758628800000010",
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-0f6f16e61616bf559",
                "LaunchTemplateName": "a2-lt-2024053005075646960000000e",
                "Version": "1"
            },
            "MinSize": 0,
            "MaxSize": 4,
            "DesiredCapacity": 2,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "us-east-1a",
                "us-east-1b"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [
                "arn:aws:elasticloadbalancing:us-east-1:905418145379:targetgroup/a2-alb-tg/956065f5654341e7"
            ],
            "HealthCheckType": "EC2",
            "HealthCheckGracePeriod": 300,
            "Instances": [
                {
                    "InstanceId": "i-0b3e7da54663f9e13",
                    "InstanceType": "t3.micro",
                    "AvailabilityZone": "us-east-1a",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchTemplate": {
                        "LaunchTemplateId": "lt-0f6f16e61616bf559",
                        "LaunchTemplateName": "a2-lt-2024053005075646960000000e",
                        "Version": "1"
                    },
                    "ProtectedFromScaleIn": false
                },
```

Figure 46: AWS CLI output for auto scaling group and launch template.

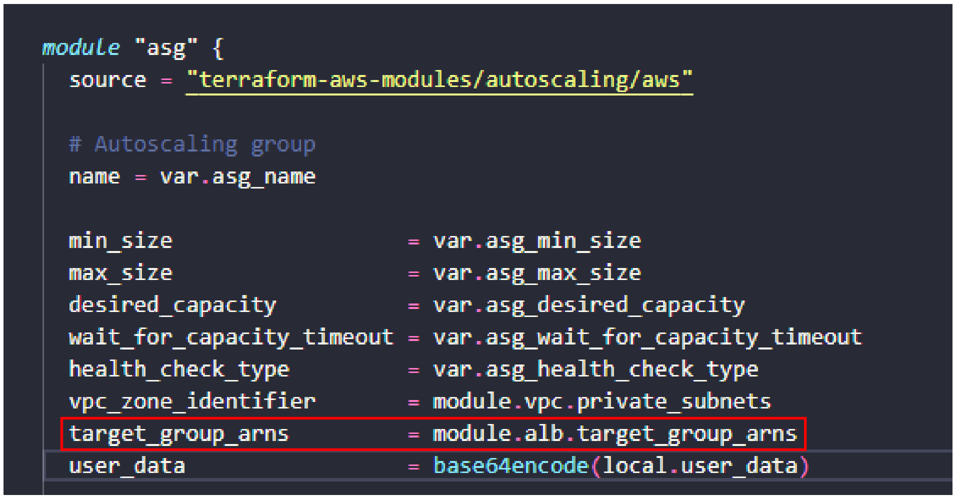
From the AWS CLI's output, we can see that our configuration is working as expected. We have deployed an auto scaling group using our launch template, it has also launched 2 EC2 WordPress Instances in 2 separate availability zones for us.



☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
☐	a2-asg	i-0b3e7da54663f9e13	Running	t3.micro	2/2 checks passed	View alarms +	us-east-1a
☐	a2-asg	i-0e45253320a88f3d5	Running	t3.micro	2/2 checks passed	View alarms +	us-east-1b

Figure 47: Automatically launched instances by auto scaling group.

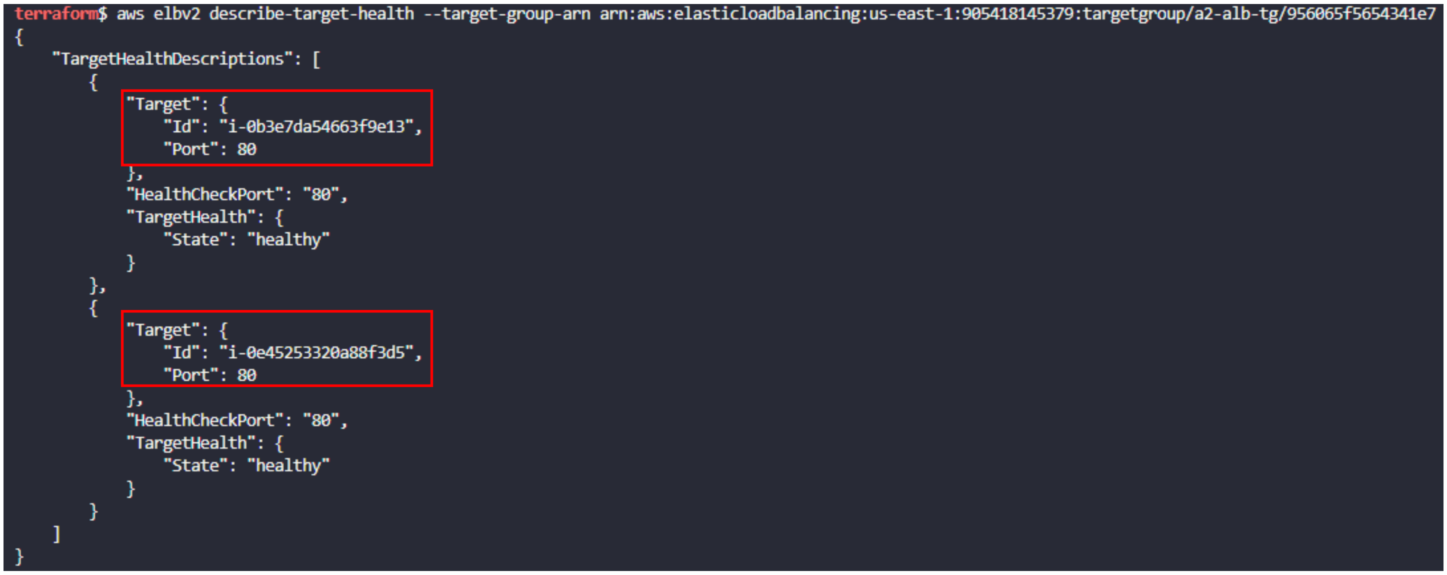
Finally for these instances to work correctly for internet users, we need to register it as target groups for our Application Load Balancer.



```
module "asg" {
  source = "terraform-aws-modules/autoscaling/aws"

  # Autoscaling group
  name = var.asg_name

  min_size                  = var.asg_min_size
  max_size                  = var.asg_max_size
  desired_capacity          = var.asg_desired_capacity
  wait_for_capacity_timeout = var.asg_wait_for_capacity_timeout
  health_check_type         = var.asg_health_check_type
  vpc_zone_identifier       = module.vpc.private_subnets
  target_group_arns         = module.alb.target_group_arns
  user_data                 = base64encode(local.user_data)
```

Figure 48: Target Group allocation

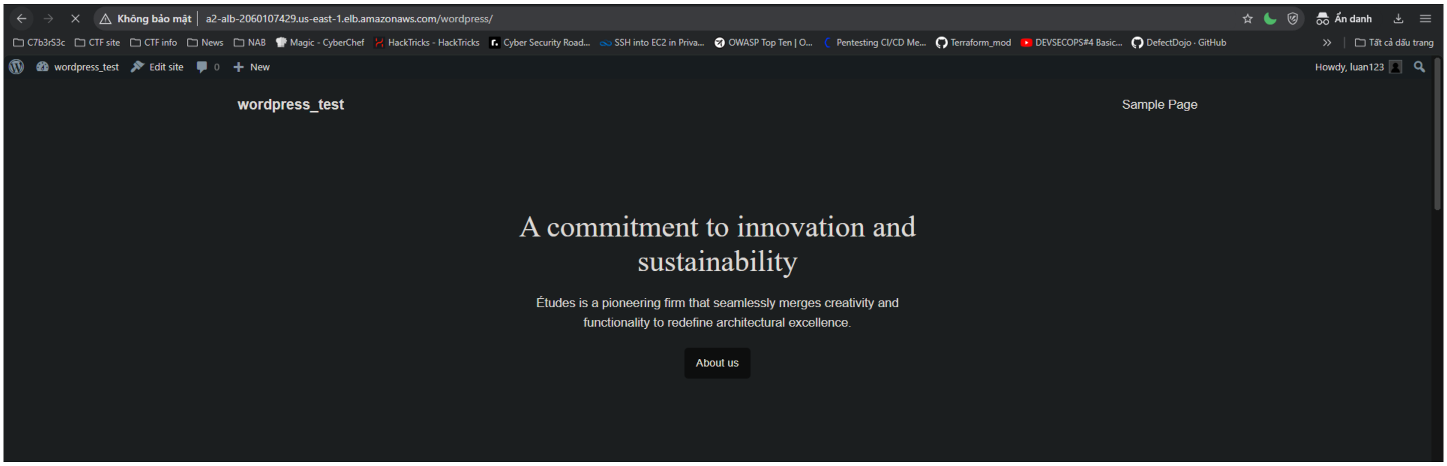
The above command means that our newly created Auto Scaling Group will be the target group of our Application Load Balancer, to verify this configuration, we can use AWS CLI as follows:



```
terraform$ aws elbv2 describe-target-health --target-group-arn arn:aws:elasticloadbalancing:us-east-1:905418145379:targetgroup/a2-alb-tg/956065f5654341e7
{
    "TargetHealthDescriptions": [
        {
            "Target": {
                "Id": "i-0b3e7da54663f9e13",
                "Port": 80
            },
            "HealthCheckPort": "80",
            "TargetHealth": {
                "State": "healthy"
            }
        },
        {
            "Target": {
                "Id": "i-0e45253320a88f3d5",
                "Port": 80
            },
            "HealthCheckPort": "80",
            "TargetHealth": {
                "State": "healthy"
            }
        }
    ]
}
```

Figure 49: Target Group verification on AWS CLI

We can see that our instances created by auto scaling group are successfully registered for the application load balancer, with that in mind we can test our WordPress site using the ALB DNS.



A commitment to innovation and sustainability

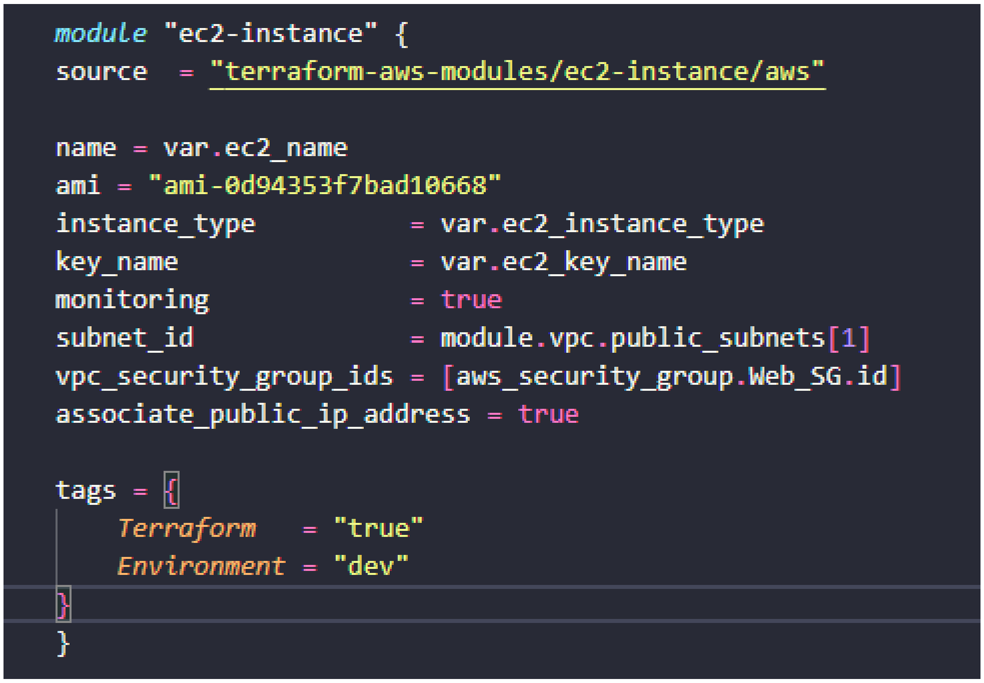
Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

Figure 50: WordPress Instance using Auto Scaling Group

We can now be sure that our WordPress instance on the newly modified infrastructure works as expected. We will move on to deploy a Bastion Host and make some security groups for our system.

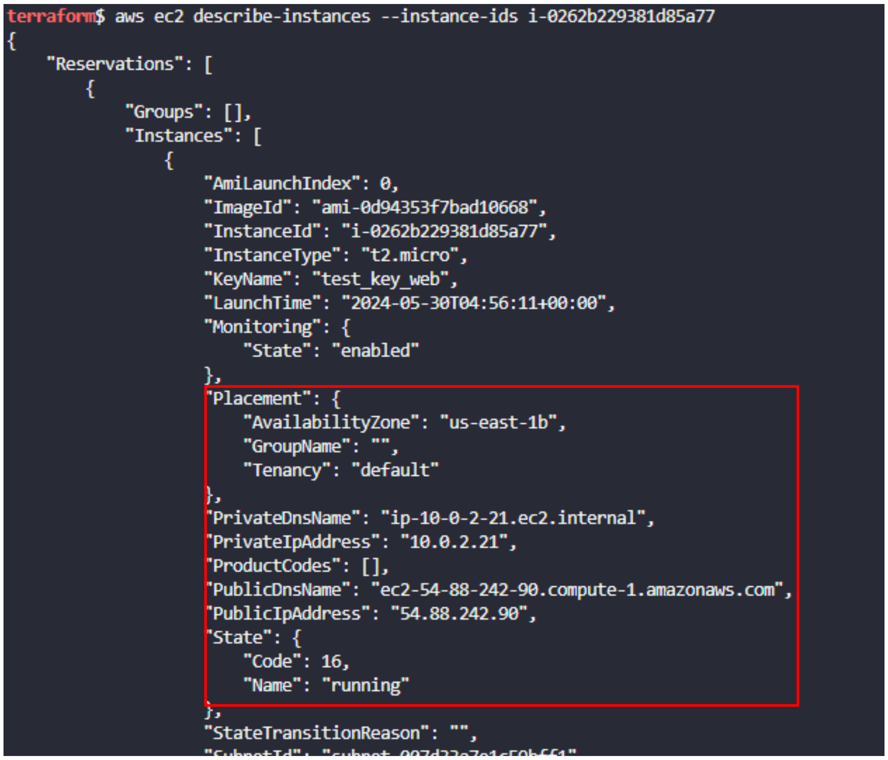
The creation of a bastion host for secure configurational entry point

For this task, we can utilize the Terraform code from task 2.2C to deploy our Bastion Host, the only difference would be this instance must be deployed on a public subnet.



```
module "ec2-instance" {
source  = "terraform-aws-modules/ec2-instance/aws"

name = var.ec2_name
ami = "ami-0d94353f7bad10668"
instance_type          = var.ec2_instance_type
key_name               = var.ec2_key_name
monitoring             = true
subnet_id              = module.vpc.public_subnets[1]
vpc_security_group_ids = [aws_security_group.Web_SG.id]
associate_public_ip_address = true

tags = {
    Terraform   = "true"
    Environment = "dev"
}
}
```

Figure 51: Code to deploy Bastion Host



```
terraform$ aws ec2 describe-instances --instance-ids i-0262b229381d85a77
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0d94353f7bad10668",
                    "InstanceId": "i-0262b229381d85a77",
                    "InstanceType": "t2.micro",
                    "KeyName": "test_key_web",
                    "LaunchTime": "2024-05-30T04:56:11+00:00",
                    "Monitoring": {
                        "State": "enabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1b",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-10-0-2-21.ec2.internal",
                    "PrivateIpAddress": "10.0.2.21",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-54-88-242-90.compute-1.amazonaws.com",
                    "PublicIpAddress": "54.88.242.90",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    },
                    "StateTransitionReason": "",
                    "SubnetId": "subnet-007d32e7e1cf0bff61"
```

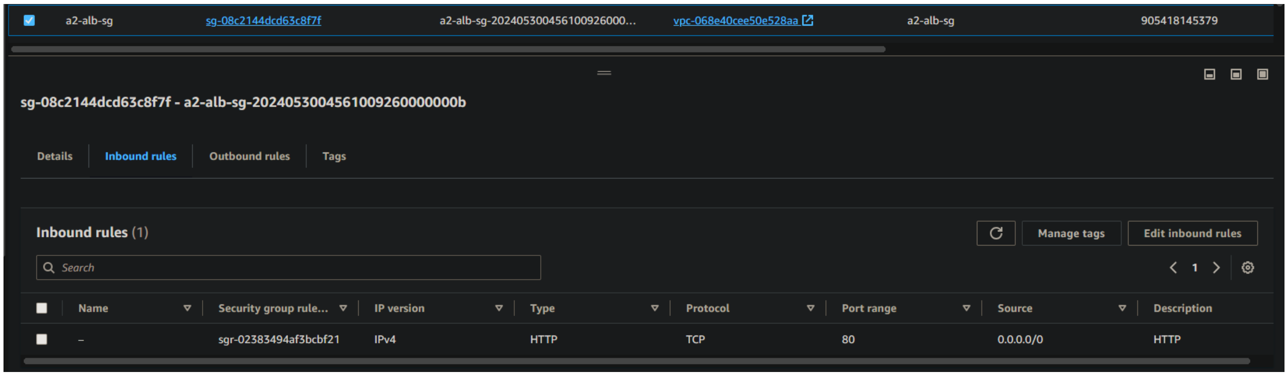
Figure 52: AWS CLI to verify the Bastion Host

Using the AWS CLI, we can verify that our Bastion Host is successfully deployed and running.

Apply additional security measures for our services.

Finally, we will apply appropriate security measures for our services, specifically we will create 4 Security Groups as follows:

+ **Application Load Balancer Security Group**: Our ALB will be internet-facing; therefore, it shall accept inbound connection from everywhere using port 80, its outbound connection can be to everywhere for the internet users.



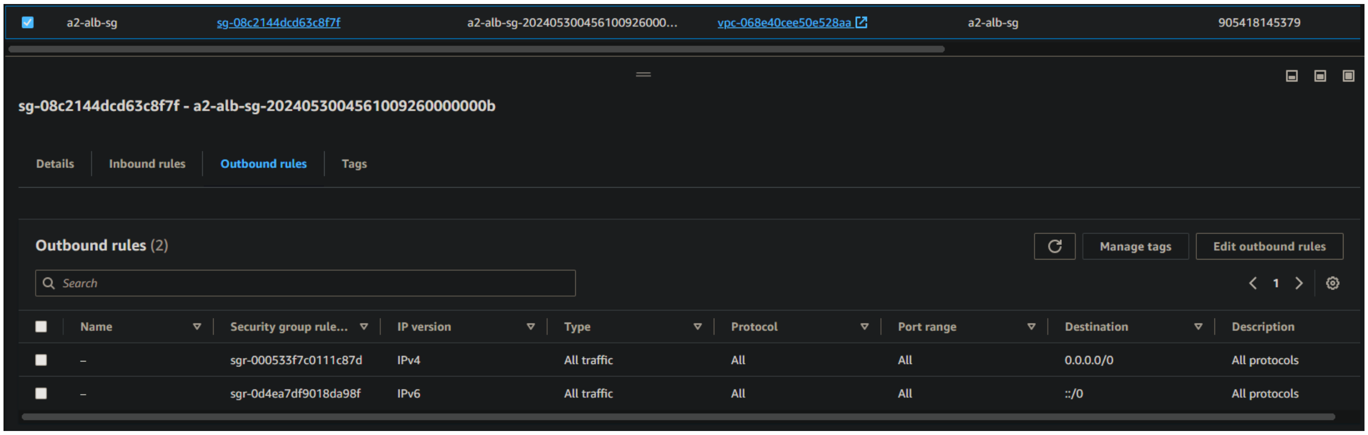
☑	a2-alb-sg	sg-08c2144dcd63c8f7f	a2-alb-sg-202405300456100926000...	vpc-068e40cee50e528aa	a2-alb-sg	905418145379

sg-08c2144dcd63c8f7f - a2-alb-sg-20240530045610092600000000b

Inbound rules (1)

□	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-02383494af3bcbf21	IPv4	HTTP	TCP	80	0.0.0.0/0	HTTP

Figure 53: ALB Inbound rules



☑	a2-alb-sg	sg-08c2144dcd63c8f7f	a2-alb-sg-202405300456100926000...	vpc-068e40cee50e528aa	a2-alb-sg	905418145379

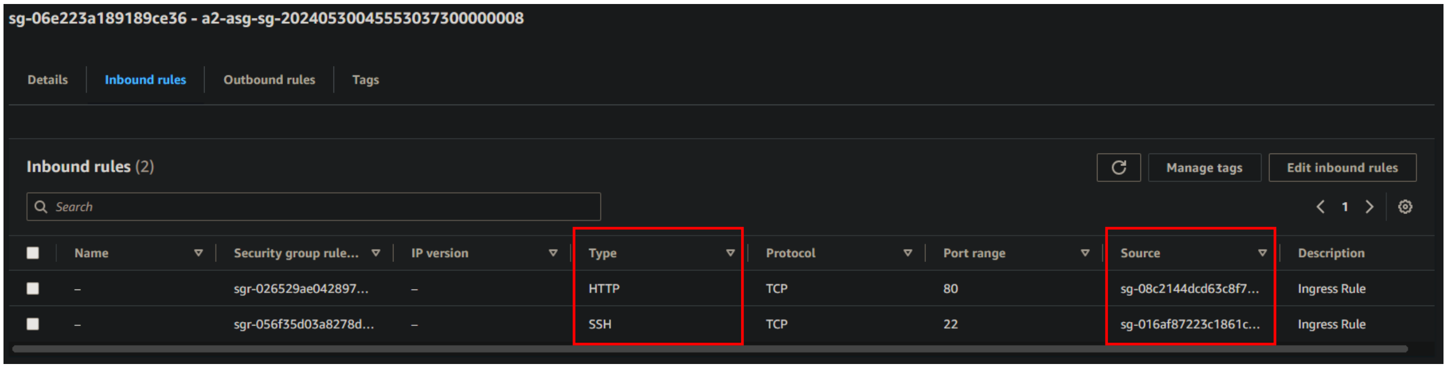
sg-08c2144dcd63c8f7f - a2-alb-sg-20240530045610092600000000b

Outbound rules (2)

□	Name	Security group rule...	IP version	Type	Protocol	Port range	Destination	Description
□	–	sgr-000533f7c0111c87d	IPv4	All traffic	All	All	0.0.0.0/0	All protocols
□	–	sgr-0d4ea7df9018da98f	IPv6	All traffic	All	All	::/0	All protocols

Figure 54: ALB outbounds rules

+ **Auto Scaling Group Security Group:** The Auto Scaling Group is located in private subnet, connection from the internet users will be directed to these instances by the Load Balancer; therefore, this security group should only accept inbound HTTP connection through port 80 from Application Load Balancer. Furthermore, it should accept SSH connection only from the Bastion Host as a secured entry for our system.

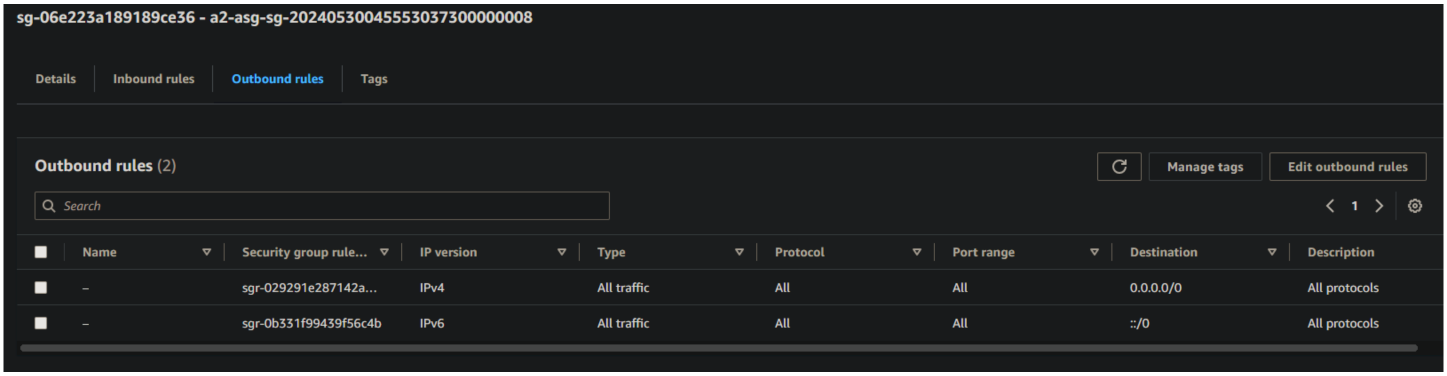


sg-06e223a189189ce36 - a2-asg-sg-20240530045553037300000008

Inbound rules (2)

□	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-026529ae042897...	–	HTTP	TCP	80	sg-08c2144dcd63c8f7...	Ingress Rule
□	–	sgr-056f35d03a8278d...	–	SSH	TCP	22	sg-016af87223c1861c...	Ingress Rule

Figure 55: Auto Scaling Group inbound rules.



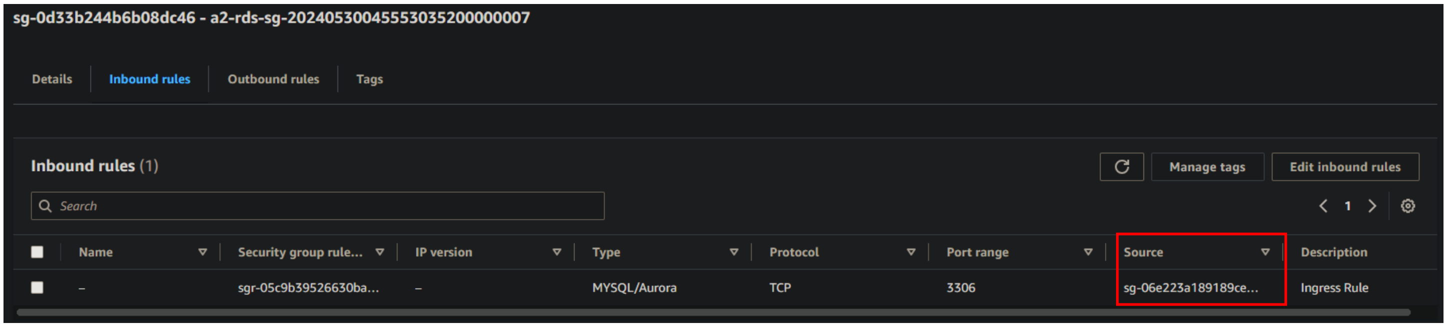
sg-06e223a189189ce36 - a2-asg-sg-20240530045553037300000008

Outbound rules (2)

Name	Security group rule...	IP version	Type	Protocol	Port range	Destination	Description
–	sgr-029291e287142a...	IPv4	All traffic	All	All	0.0.0.0/0	All protocols
–	sgr-0b331f99439f56c4b	IPv6	All traffic	All	All	::/0	All protocols

Figure 56: Auto Scaling Group outbound rules

+ **AWS RDS Security Group:** Our database should only accept inbound connection through the auto scaling group, there is no need for outbound rules for this entity.



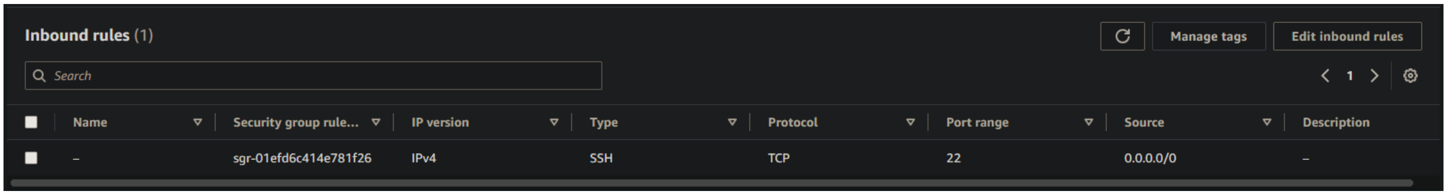
sg-0d33b244b6b08dc46 - a2-rds-sg-20240530045553035200000007

Inbound rules (1)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
–	sgr-05c9b39526630ba...	–	MYSQL/Aurora	TCP	3306	sg-06e223a189189ce...	Ingress Rule

Figure 57: AWS RDS inbound rules

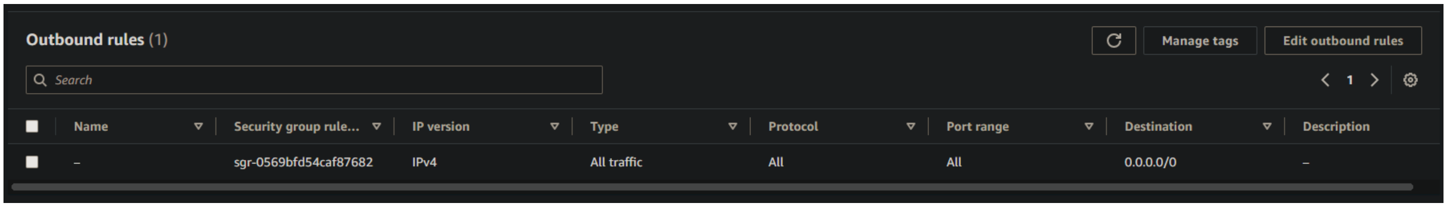
+ **Bastion Host Security Group:** Our Bastion Host can accept inbound SSH connection from everywhere, only verified credentials using the key we created in task 2.1P can be granted with access to our bastion host. Outbound connection can be anything.



Inbound rules (1)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
–	sgr-01efd6c414e781f26	IPv4	SSH	TCP	22	0.0.0.0/0	–

Figure 58: Bastion Host inbound rules



Outbound rules (1)

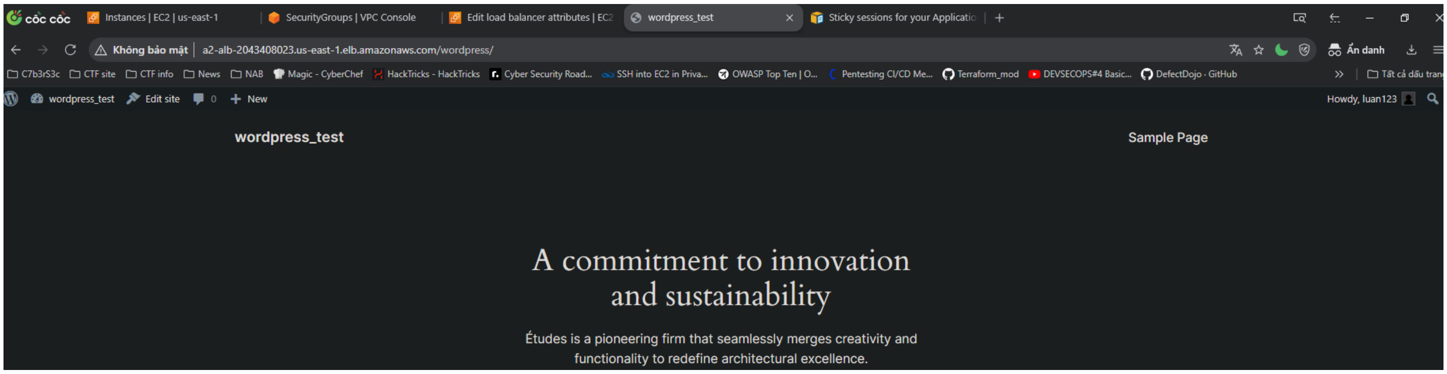
Name	Security group rule...	IP version	Type	Protocol	Port range	Destination	Description
–	sgr-0569bfd54caf87682	IPv4	All traffic	All	All	0.0.0.0/0	–

Figure 59: Bastion Host outbound rules.

This will conclude the security measures for our system, we can move on to perform various scaling testing to see if our system works as expected.

Scaling and functionality testing

After all the configuration, we can proceed to test if our website is still working as expected. We can use the ALB's DNS to access our WordPress Private Instance.



a2-alb-2043408023.us-east-1.elb.amazonaws.com/wordpress/

wordpress_test

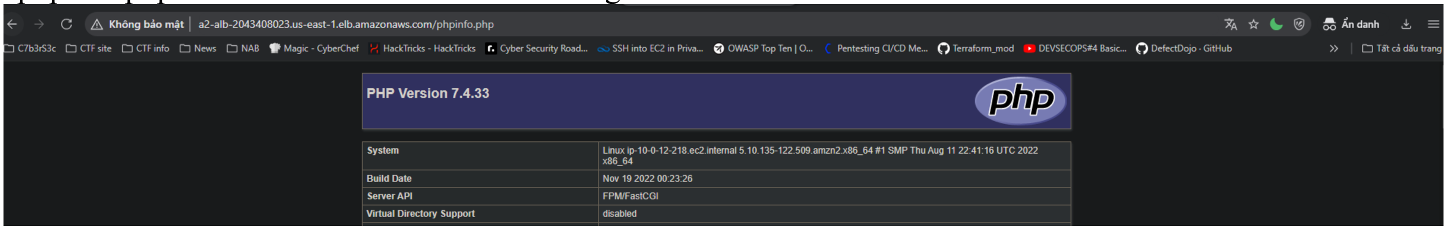
Sample Page

A commitment to innovation and sustainability

Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

Figure 60: Access to Website using ALB's DNS

It seems that our infrastructure and WordPress Instances are working as expected, we can proceed to use "phpinfo.php" file to see if our ALB is directing our connection to both of our instances.

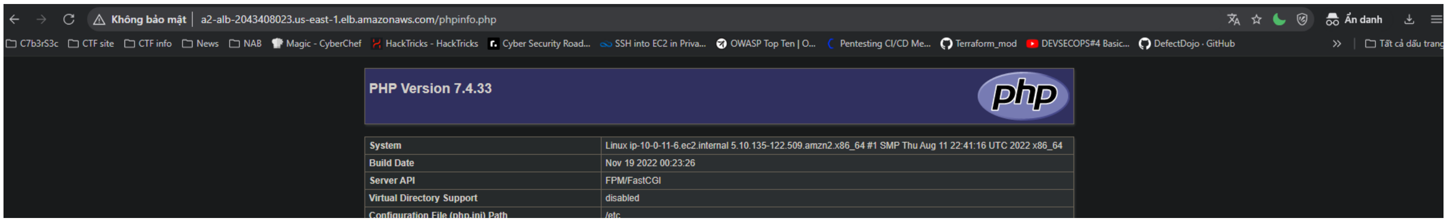


a2-alb-2043408023.us-east-1.elb.amazonaws.com/phpinfo.php

PHP Version 7.4.33

System	Linux ip-10-0-12-218.ec2.internal 5.10.135-122.509.amzn2.x86_64 #1 SMP Thu Aug 11 22:41:16 UTC 2022 x86_64
Build Date	Nov 19 2022 00:23:26
Server API	FPM/FastCGI
Virtual Directory Support	disabled

Figure : Load Balance to Private Instance 1



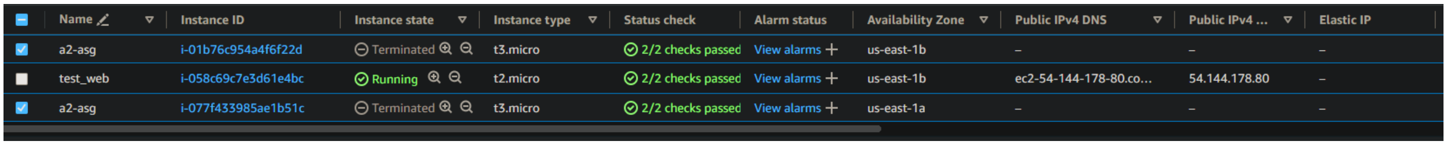
a2-alb-2043408023.us-east-1.elb.amazonaws.com/phpinfo.php

PHP Version 7.4.33

System	Linux ip-10-0-11-6.ec2.internal 5.10.135-122.509.amzn2.x86_64 #1 SMP Thu Aug 11 22:41:16 UTC 2022 x86_64
Build Date	Nov 19 2022 00:23:26
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc

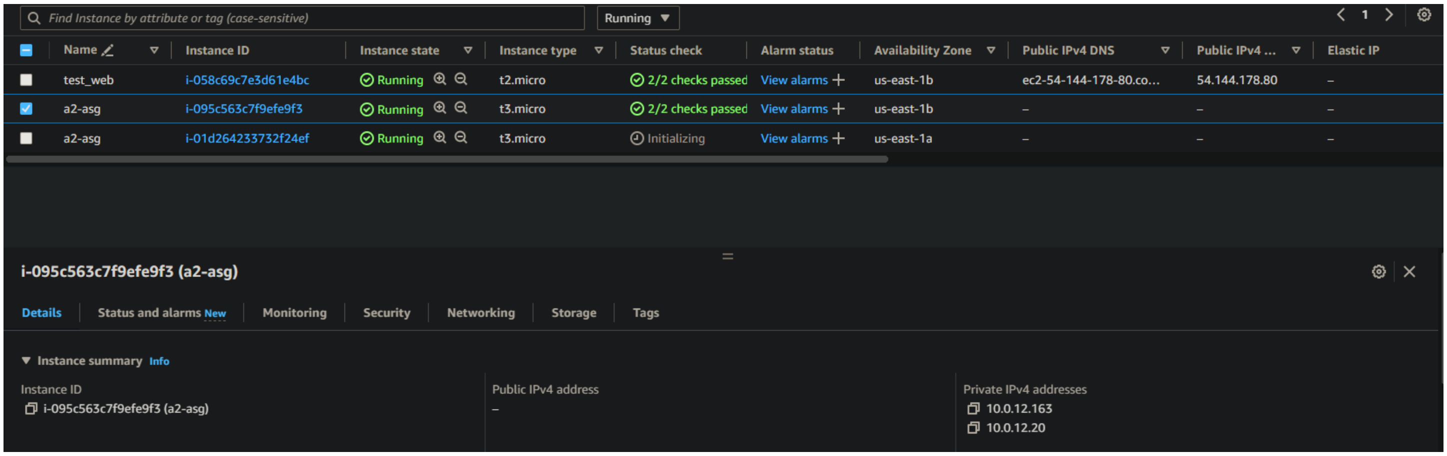
Figure 61: Load Balance to Private Instance 2

Finally, we can test if our scaling feature is working by terminating both instances to see if our auto scaling group can provision additional instances so that our system can still be functional.



☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
☑	a2-asg	i-01b76c954a4f6f22d	Terminated	t3.micro	2/2 checks passed	View alarms +	us-east-1b	–	–	–
☐	test_web	i-058c69c7e3d61e4bc	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-54-144-178-80.co...	54.144.178.80	–
☑	a2-asg	i-077f433985ae1b51c	Terminated	t3.micro	2/2 checks passed	View alarms +	us-east-1a	–	–	–

Figure 62: WordPress Instances terminated



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
test_web	i-058c69c7e3d61e4bc	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-54-144-178-80.co...	54.144.178.80	–
a2-asg	i-095c563c7f9efe9f3	Running	t3.micro	2/2 checks passed	View alarms +	us-east-1b	–	–	–
a2-asg	i-01d264233732f24ef	Running	t3.micro	Initializing	View alarms +	us-east-1a	–	–	–

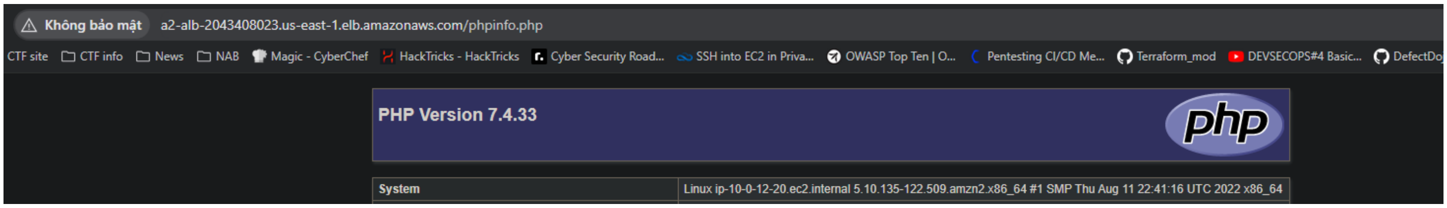
i-095c563c7f9efe9f3 (a2-asg)

Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-095c563c7f9efe9f3 (a2-asg)	–	10.0.12.163 10.0.12.20

Figure 63: 2 new instances are provisioned

After terminating, two new instances are immediately provisioned to ensure high availability of our system. We can browse using our ALB's DNS to see if our website is working as expected.

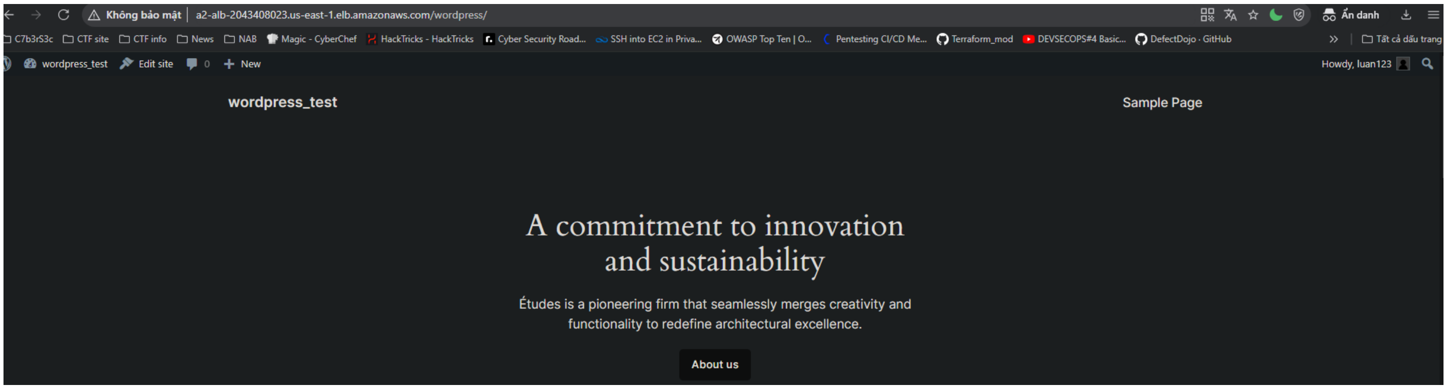


a2-alb-2043408023.us-east-1.elb.amazonaws.com/phpinfo.php

PHP Version 7.4.33

System	Linux ip-10-0-12-20.ec2.internal 5.10.135-122.509.amzn2.x86_64 #1 SMP Thu Aug 11 22:41:16 UTC 2022 x86_64

Figure 64: New instance works as expected



a2-alb-2043408023.us-east-1.elb.amazonaws.com/wordpress/

wordpress_test

A commitment to innovation and sustainability

Études is a pioneering firm that seamlessly merges creativity and functionality to redefine architectural excellence.

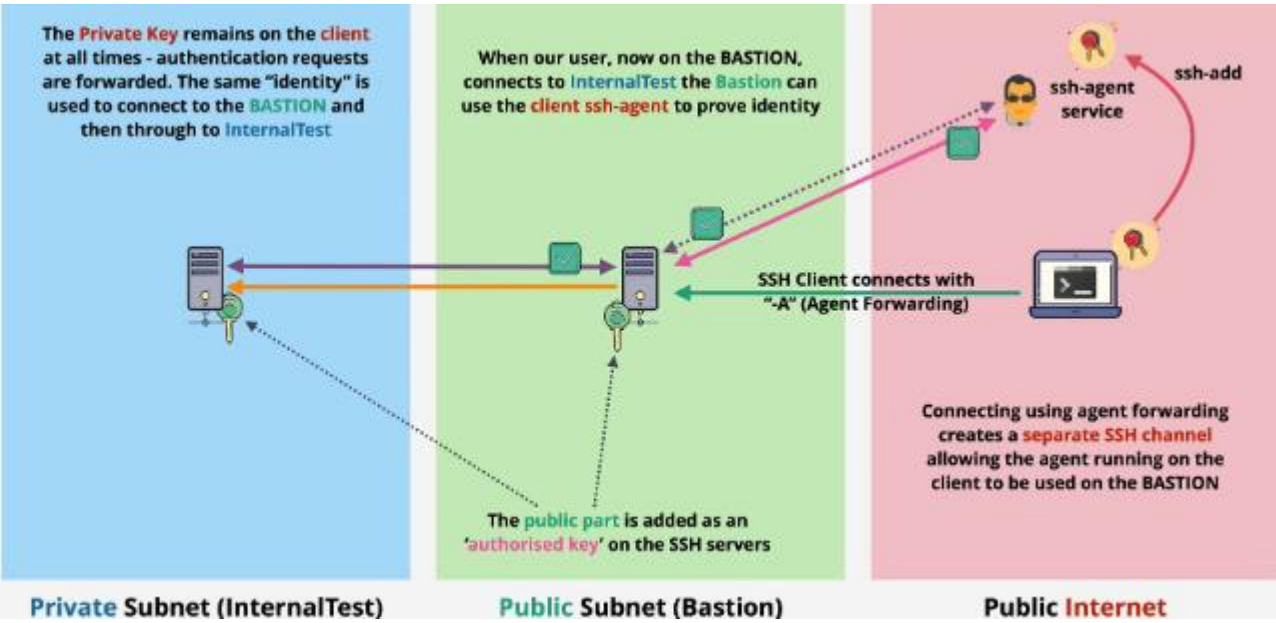
Figure 65: New instances ensure succifient scaling and functionality

Now we can be sure that our system is enabled with succifient auto scaling measures that will ensure high availability and functionalities according to requirements.

Task 2.4HD: Complete task 2.3 and set up and demonstrate SSH access from Windows or Mac.

2.4A. Required configuration to establish SSH connection to WordPress Instances

As our WordPress Instances is in the private subnet, we can not establish an SSH connection directly to them. In order to resolve this, I have provision a Bastion Host in Task 2.3 and now we can use ot as a secured entry point to our private subnet. To further promote the security aspect of our system, we will use a technique called **SSH Agent Forwarding** so as not to store our private key on the Bastion Host.



The Private Key remains on the client at all times - authentication requests are forwarded. The same "identity" is used to connect to the BASTION and then through to InternalTest

When our user, now on the BASTION, connects to InternalTest the Bastion can use the client ssh-agent to prove identity

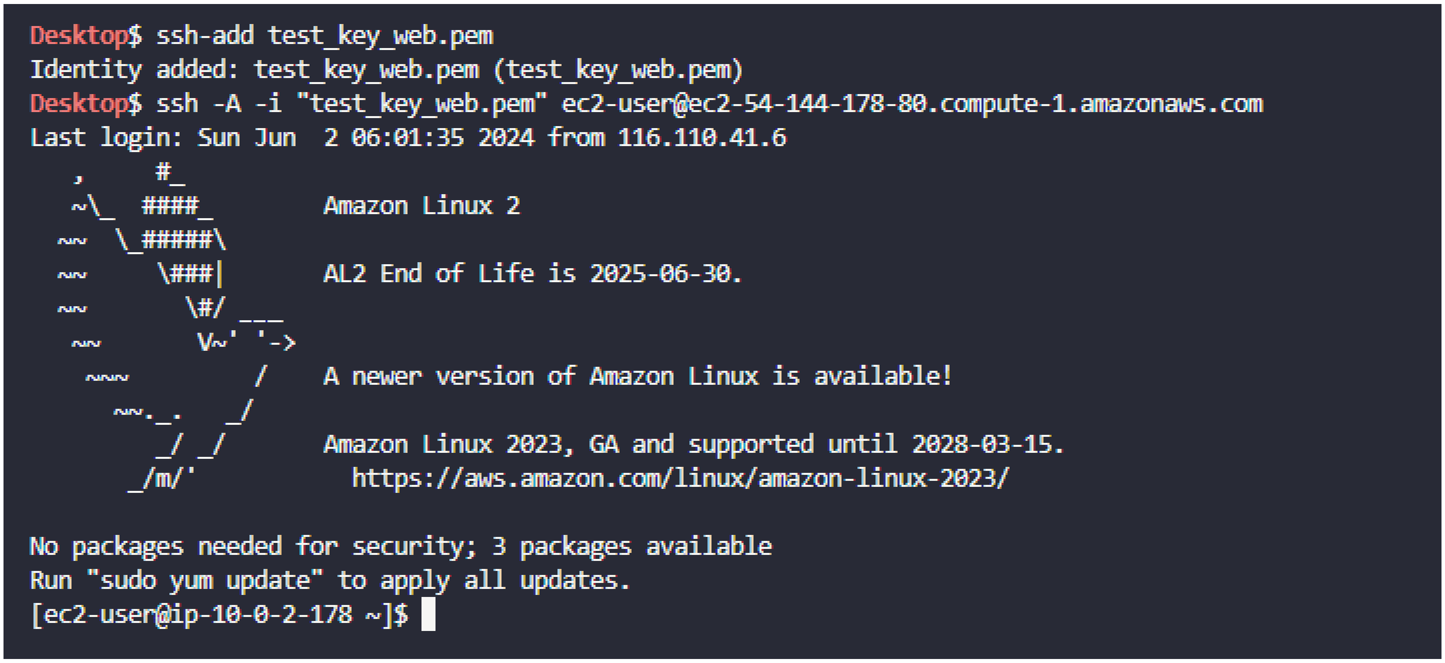
SSH Client connects with "-A" (Agent Forwarding)

Connecting using agent forwarding creates a separate SSH channel allowing the agent running on the client to be used on the BASTION

The public part is added as an 'authorised key' on the SSH servers

Figure 66: SSH Agent Forwarding

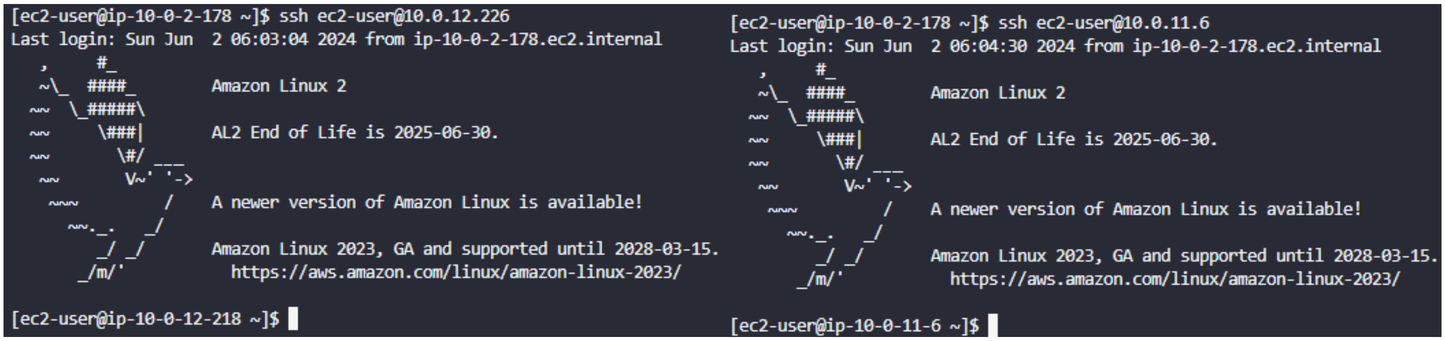
In this resolution, we first add the private key as an `identity` into `ssh-agent service`. Then we can connect to the Bastion Host with the `-A` tag to inform that whenever other hosts ask the Bastion to identify its identity, the Bastion Host can forward that request to the `ssh-agent service`



```
Desktop$ ssh-add test_key_web.pem
Identity added: test_key_web.pem (test_key_web.pem)
Desktop$ ssh -A -i "test_key_web.pem" ec2-user@ec2-54-144-178-80.compute-1.amazonaws.com
Last login: Sun Jun  2 06:01:35 2024 from 116.110.41.6
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

No packages needed for security; 3 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-2-178 ~]$ 
```

Figure 67: Establish connection to Bastion Host

After that we can normally establish an SSH connection to our private instances.

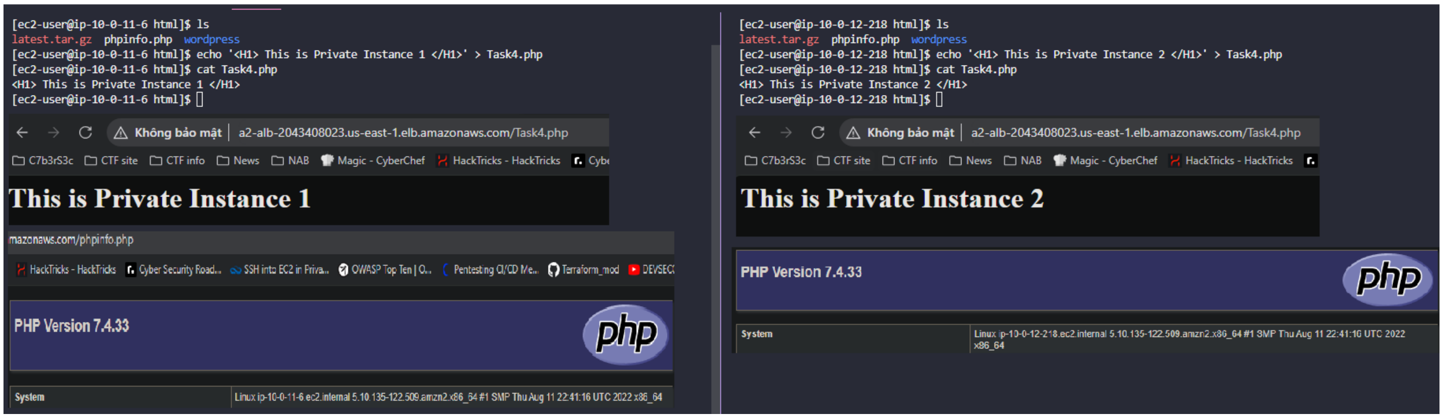


```
[ec2-user@ip-10-0-2-178 ~]$ ssh ec2-user@10.0.12.226
Last login: Sun Jun  2 06:03:04 2024 from ip-10-0-2-178.ec2.internal
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-12-218 ~]$ 
```

```
[ec2-user@ip-10-0-2-178 ~]$ ssh ec2-user@10.0.11.6
Last login: Sun Jun  2 06:04:30 2024 from ip-10-0-2-178.ec2.internal
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-11-6 ~]$ 
```

Figure 68: Establish connection to private instances from Bastion Host

After the connection to private instances is established, we can make some modifications to see if it would reflect on our website.



```
[ec2-user@ip-10-0-11-6 html]$ ls
latest.tar.gz  phpinfo.php  wordpress
[ec2-user@ip-10-0-11-6 html]$ echo '<H1> This is Private Instance 1 </H1>' > Task4.php
[ec2-user@ip-10-0-11-6 html]$ cat Task4.php
<H1> This is Private Instance 1 </H1>
[ec2-user@ip-10-0-11-6 html]$ 
```

Không bảo mật | a2-alb-2043408023.us-east-1.elb.amazonaws.com/Task4.php

C7b3rS3c CTF site CTF info News NAB Magic - CyberChef HackTricks - HackTricks Cybe

This is Private Instance 1

mazonaws.com/phpinfo.php

HackTricks - HackTricks Cyber Security Road... SSH into EC2 in Priva... OWASP Top Ten | O... Pentesting CI/CD Me... Terraform_mod DEVSEC

PHP Version 7.4.33

System | Linux ip-10-0-11-6.ec2.internal 5.10.135-122.509.amzn2.x86_64 #1 SMP Thu Aug 11 22:41:16 UTC 2022 x86_64

```
[ec2-user@ip-10-0-12-218 html]$ ls
latest.tar.gz  phpinfo.php  wordpress
[ec2-user@ip-10-0-12-218 html]$ echo '<H1> This is Private Instance 2 </H1>' > Task4.php
[ec2-user@ip-10-0-12-218 html]$ cat Task4.php
<H1> This is Private Instance 2 </H1>
[ec2-user@ip-10-0-12-218 html]$ 
```

Không bảo mật | a2-alb-2043408023.us-east-1.elb.amazonaws.com/Task4.php

C7b3rS3c CTF site CTF info News NAB Magic - CyberChef HackTricks - HackTricks

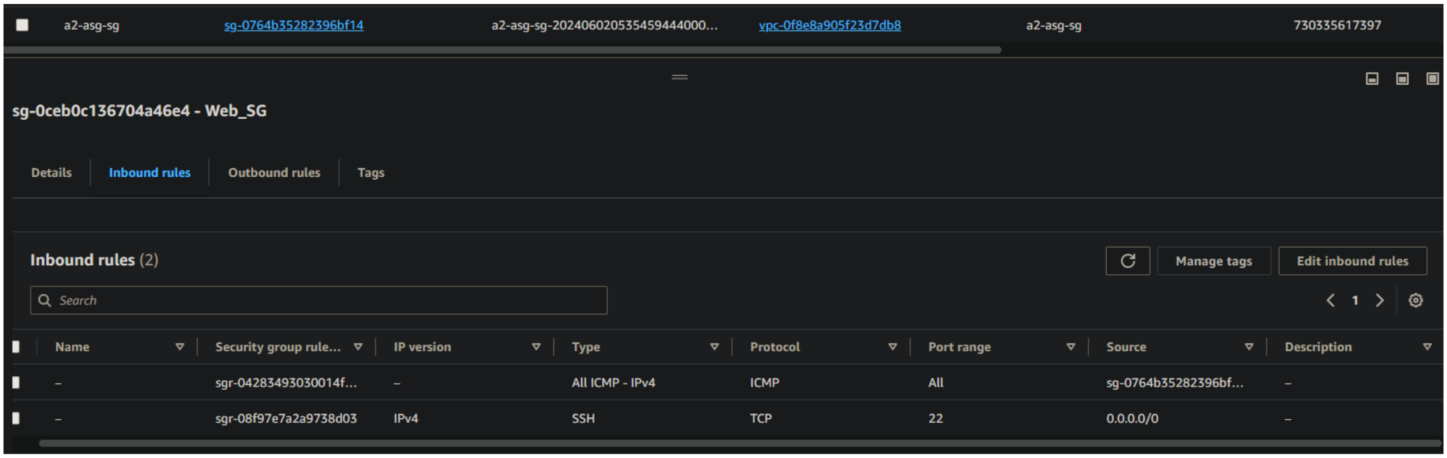
This is Private Instance 2

PHP Version 7.4.33

System | Linux ip-10-0-12-218.ec2.internal 5.10.135-122.509.amzn2.x86_64 #1 SMP Thu Aug 11 22:41:16 UTC 2022 x86_64

Figure 69: Modification reflected on the website

Additionally we can perform some command on the private instance, let modify the security group to allow ICMP from Bastion Host to Private Instances and vice versa to execute an ICMP command and see if our instances will allow it.



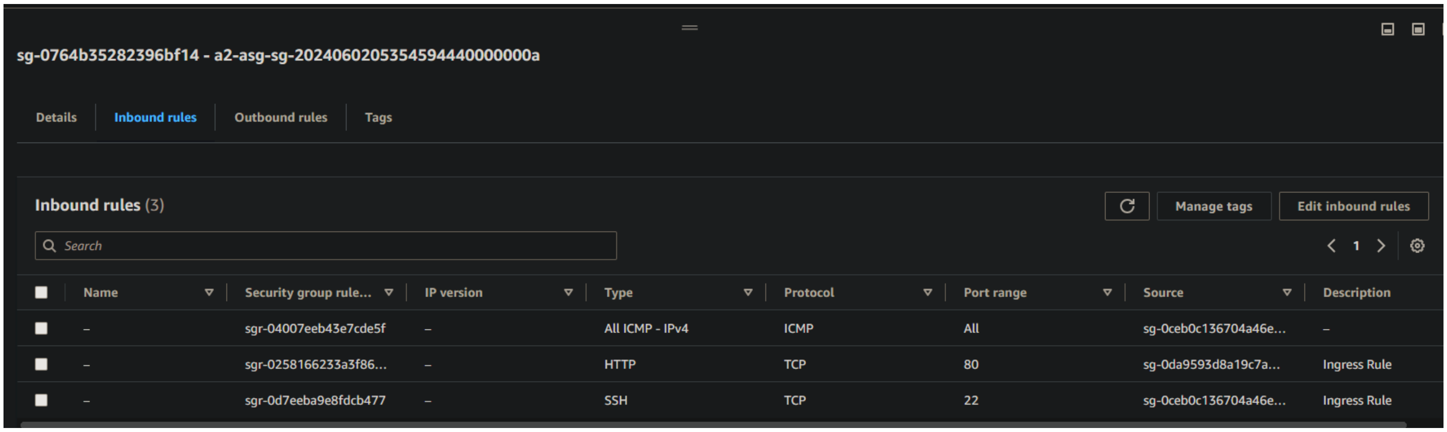
□ a2-asg-sg sg-0764b35282396bf14 a2-asg-sg-20240602053545944400... vpc-0f8e8a905f23d7db8 a2-asg-sg 730335617397

sg-0ceb0c136704a46e4 - Web_SG

Inbound rules (2)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
–	sgr-04283493030014f...	–	All ICMP - IPv4	ICMP	All	sg-0764b35282396bf...	–
–	sgr-08f97e7a2a9738d03	IPv4	SSH	TCP	22	0.0.0.0/0	–

Figure 70: Modified Security Groups for Bastion Host



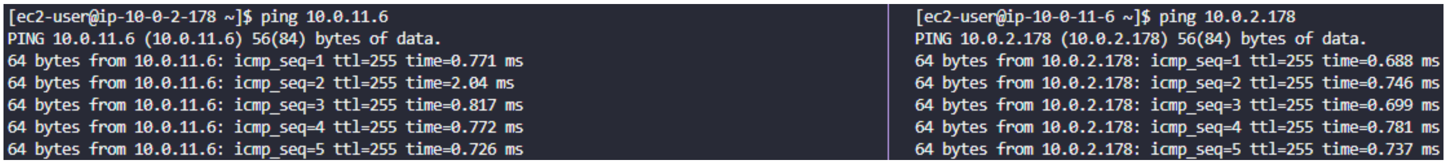
sg-0764b35282396bf14 - a2-asg-sg-20240602053545944400000000a

Inbound rules (3)

□	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-04007eeb43e7cde5f	–	All ICMP - IPv4	ICMP	All	sg-0ceb0c136704a46e...	–
□	–	sgr-0258166233a3f86...	–	HTTP	TCP	80	sg-0da9593d8a19c7a...	Ingress Rule
□	–	sgr-0d7eeba9e8fdcb477	–	SSH	TCP	22	sg-0ceb0c136704a46e...	Ingress Rule

Figure 71: Modified Security Groups for Private Instances

After that we can execute ping command from and to our private instances to see if this works.



```
[ec2-user@ip-10-0-2-178 ~]$ ping 10.0.11.6
PING 10.0.11.6 (10.0.11.6) 56(84) bytes of data.
64 bytes from 10.0.11.6: icmp_seq=1 ttl=255 time=0.771 ms
64 bytes from 10.0.11.6: icmp_seq=2 ttl=255 time=2.04 ms
64 bytes from 10.0.11.6: icmp_seq=3 ttl=255 time=0.817 ms
64 bytes from 10.0.11.6: icmp_seq=4 ttl=255 time=0.772 ms
64 bytes from 10.0.11.6: icmp_seq=5 ttl=255 time=0.726 ms
```

```
[ec2-user@ip-10-0-11-6 ~]$ ping 10.0.2.178
PING 10.0.2.178 (10.0.2.178) 56(84) bytes of data.
64 bytes from 10.0.2.178: icmp_seq=1 ttl=255 time=0.688 ms
64 bytes from 10.0.2.178: icmp_seq=2 ttl=255 time=0.746 ms
64 bytes from 10.0.2.178: icmp_seq=3 ttl=255 time=0.699 ms
64 bytes from 10.0.2.178: icmp_seq=4 ttl=255 time=0.781 ms
64 bytes from 10.0.2.178: icmp_seq=5 ttl=255 time=0.737 ms
```

Figure 72: Ping commands executed from and to Private instances

Modification made on private instances is reflected on the functionality of our website. We can now SSH onto our private instances and made modification whenever needed. This will also conclude Deployment Assignment 2.

Resources

The full code of terraform and all the scripts that I used in this assignment can be found via this link.

Should the lecturer wanted to test the functionality of my website, kindly inform me and I will relaunch my Terraform scripts for each of the task, it should takes only 10 minutes to provision and we can have live testing. I can not give the link as the resources are too expensive and this course does not provide me with any account or resources.