Assignment 2 ACIT 4640

This script is creating a VPC with public and private subnets, an internet gateway, a route table, two security groups, an EC2 instance, and an RDS database.

First we need to create a VPC with CIDR block 10.0.0.0/16 and assign a name to it. Then, it creates a public subnet with CIDR block 10.0.1.0/24 and assigns it to the VPC. Two private subnets are also created with CIDR blocks 10.0.2.0/24 and 10.0.3.0/24 and assigned to different availability zones.

```
# Create vpc
vpc_id=$(aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications
'ResourceType=vpc,Tags=[{Key=Name,Value=assign2_VPC}]' --query 'Vpc.VpcId'
--output text)
echo "Created VPC: $vpc_id"

# Create public subnet
public_subnet_id=$(aws ec2 create-subnet --vpc-id $vpc_id --cidr-block
10.0.1.0/24 --availability-zone us-west-2a --tag-specifications
'ResourceType=subnet,Tags=[{Key=Name,Value=assign2_public_Subnet}]' --query
'Subnet.SubnetId' --output text)
echo "Created public subnet: $public_subnet_id"

# Create private subnets
private_subnet_1_id=$(aws ec2 create-subnet --vpc-id $vpc_id --cidr-block
10.0.2.0/24 --availability-zone us-west-2b --tag-specifications
'ResourceType=subnet,Tags=[{Key=Name,Value=assign2_private_Subnet1}]'
--query 'Subnet.SubnetId' --output text)
echo "Created private subnet 1: $private_subnet_1_id"
private_subnet_2_id=$(aws ec2 create-subnet --vpc-id $vpc_id --cidr-block
10.0.3.0/24 --availability-zone us-west-2c --tag-specifications
'ResourceType=subnet,Tags=[{Key=Name,Value=assign2_private_Subnet2}]'
--query 'Subnet.SubnetId' --output text)
echo "Created private subnet 2: $private_subnet_2_id"
```

An internet gateway is created and attached to the VPC, and a route table is created and associated with the public subnet. A route is added to the route table to allow traffic to flow to the internet through the internet gateway.

```
# Create internet gateway
gateway_id=$(aws ec2 create-internet-gateway --tag-specifications
'ResourceType=internet-gateway,Tags=[{Key=Name,Value=igw}]' --query
'InternetGateway.InternetGatewayId' --output text)
aws ec2 attach-internet-gateway --vpc-id $vpc_id --internet-gateway-id
$gateway_id
echo "Created internet gateway: $gateway_id"

# Create route table
route_table_id=$(aws ec2 create-route-table --vpc-id $vpc_id
--tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=rt}]'
--query 'RouteTable.RouteTableId' --output text)
aws ec2 associate-route-table --route-table-id $route_table_id --subnet-id
$public_subnet_id
aws ec2 create-route --route-table-id $route_table_id
--destination-cidr-block 0.0.0.0/0 --gateway-id $gateway_id
echo "Created route table: $route_table_id"
```

Two security groups are created: one for the EC2 instance and one for the RDS database. The security group for the EC2 instance allows incoming SSH and HTTP traffic from anywhere. The security group for the RDS database allows incoming MySQL traffic from the VPC CIDR block and from itself.

```
# Create security groups
ec2_sg_id=$(aws ec2 create-security-group --group-name assign2-ec2-sg
--description "Security group for EC2 instance" --vpc-id $vpc_id
--tag-specifications
'ResourceType=security-group,Tags=[{Key=Name,Value=assign2-ec2-sg}]'
--query 'GroupId' --output text)
rds_sg_id=$(aws ec2 create-security-group --group-name assign2-rds-sg
--description "Security group for RDS database" --vpc-id $vpc_id
--tag-specifications
'ResourceType=security-group,Tags=[{Key=Name,Value=assign2-rds-sg}]'
--query 'GroupId' --output text)
echo "Created security groups: ec2-sg ($ec2_sg_id), rds-sg ($rds_sg_id)"

# SG for EC2 instance
```

```
aws ec2 authorize-security-group-ingress --group-id $ec2_sg_id --protocol
tcp --port 22 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress --group-id $ec2_sg_id --protocol
tcp --port 80 --cidr 0.0.0.0/0
```

An SSH key pair is also created and stored in a local file for use when launching the EC2 instance.

```
# SSH key
ssh_key=bookstack

echo "Creating SSH key..."
aws ec2 create-key-pair --key-name $ssh_key --query 'KeyMaterial' --output
text > $ssh_key.pem
sudo chmod 400 $ssh_key.pem
```

The script then creates the EC2 instance using the specified AMI, instance type, key pair, subnet, security group, and tags. It also assigns a public IP address to the instance.

```
# Create EC2 instance
echo "Creating EC2 instance..."
instance_id=$(aws ec2 run-instances \
        --image-id ami-0735c191cf914754d  \
        --instance-type t2.micro \
        --key-name $ssh_key \
        --subnet-id $public_subnet_id \
        --security-group-ids $ec2_sg_id \
        --associate-public-ip-address \
        --tag-specifications
'ResourceType=instance,Tags=[{Key=Name,Value=bookstack_instances}]' \
        --query 'Instances[0].InstanceId' --output text)
echo "Created EC2 instance: $instance_id"
```

Finally, a subnet group is created for the RDS database, and the RDS database is created using the subnet group, security group, and specified parameters.

```
# Create subnet group for RDS database
subnet_group_name="rds-subnet-group"
aws rds create-db-subnet-group --db-subnet-group-name $subnet_group_name
--db-subnet-group-description "Subnet group for RDS database" --subnet-ids
$private_subnet_1_id $private_subnet_2_id
echo "Created subnet group for RDS database: $subnet_group_name"

# Create RDS database
echo "Creating RDS database..."
aws rds create-db-instance \
--db-instance-identifier my-rds-instance \
--db-instance-class db.t2.micro \
--engine mysql \
--allocated-storage 20 \
--db-name bookstack \
--master-username bookstack \
--master-user-password password \
--vpc-security-group-ids $rds_sg_id \
--db-subnet-group-name $subnet_group_name
echo "Created RDS database: my-rds-db"

rds_endpoint=$(aws rds describe-db-instances --db-instance-identifier
my-rds-instance --query 'DBInstances[0].Endpoint.Address' --output text)
echo "RDS endpoint: rds_endpoint"
```

Then we want to make sure to describe our infrastructure. At the bottom of the script we can make a function to describe the resources.

```
function describe_resources() {
  read -p "Do you want to describe AWS resources? (y/n) " answer
  case $answer in
    [yY])
      aws ec2 describe-vpcs --filters "Name=tag:Name,Values=assign2_VPC"
      aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id"
"Name=tag:Name,Values=assign2_public_Subnet"
      aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id"
"Name=tag:Name,Values=assign2_private_Subnet1"
      aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id"
"Name=tag:Name,Values=assign2_private_Subnet2"
      aws ec2 describe-internet-gateways --filters
```

```
"Name=attachment.vpc-id,Values=$vpc_id"
      aws ec2 describe-route-tables --filters "Name=vpc-id,Values=$vpc_id"
      aws ec2 describe-security-groups --filters
"Name=vpc-id,Values=$vpc_id" "Name=tag:Name,Values=assign2-ec2-sg"
      aws ec2 describe-security-groups --filters
"Name=vpc-id,Values=$vpc_id" "Name=tag:Name,Values=assign2-rds-sg"
      aws ec2 describe-instances --filters "Name=vpc-id,Values=$vpc_id"
"Name=instance-state-name,Values=running"
      aws rds describe-db-instances --filters
"Name=db-instance-id,Values=my-rds-instance"
      ;;
    [nN])
      echo "Exiting without describing resources."
      ;;
    *)
      echo "Invalid input. Exiting without describing resources."
      ;;
  esac
}

describe_resources
```

After we finish creating the infrastructure script we can begin creating the bookstack script. First we need to create a few variables to pass into commands later on in the script. We also have to make sure the user runs the script as root.

```
# Gets public ip of ec2 instance
ip_a=$(curl http://169.254.169.254/latest/meta-data/public-ipv4)

# BookStack installation Directory
BOOKSTACK_DIR="/var/www/bookstack"

SCRIPT_USER="${SUDO_USER:-$USER}"

#make sure to run as root
if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root"
   exit 1
fi
```

Now we can create the functions to automatically install BookStack. First we need to install a set of packages that are required by BookStack to run. Next we need to clone the BookStack repository from GitHub. Finally it installs Composer, a PHP package manager. It first downloads the Composer installer and its SHA-384 checksum, and verifies the downloaded file's checksum.

```
#install packages
function run_package_installs() {
    apt-get update
    apt-get install -y git nginx curl unzip mysql-client php8.1-fpm
php8.1-mysql php8.1-mbstring php8.1-gd php8.1-curl php8.1-ldap
php8.1-mbstring php8.1-xml
}

#download bookstack
function run_bookstack_download() {
  cd /var/www || exit
  git clone https://github.com/BookStackApp/BookStack.git --branch release
--single-branch bookstack
}

#install composer
function run_install_composer() {
  EXPECTED_CHECKSUM="$(php -r
'copy("https://composer.github.io/installer.sig", "php://stdout");')"
  php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
  ACTUAL_CHECKSUM="$(php -r "echo hash_file('sha384',
'composer-setup.php');")"

  if [ "$EXPECTED_CHECKSUM" != "$ACTUAL_CHECKSUM" ]
  then
      >&2 echo 'ERROR: Invalid composer installer checksum'
      rm composer-setup.php
      exit 1
  fi

  php composer-setup.php --quiet
  rm composer-setup.php

  # Move composer to global installation
  mv composer.phar /usr/local/bin/composer
}

#install bookstack composer dependencies
```

```
function run_install_bookstack_composer_deps() {
  cd "$BOOKSTACK_DIR" || exit
  export COMPOSER_ALLOW_SUPERUSER=1
  php /usr/local/bin/composer install --no-dev --no-plugins
}
```

We need to change the default configuration settings in the .env file that comes with BookStack. Specifically, the script sets the database host to an Amazon RDS instance, sets the application URL to the IP address of the server, sets the database name to "bookstack", sets the database username to "bookstack", and sets the database password to "password". It also generates an application key for security. Then we run the database migrations required by BookStack using the php artisan migrate command. After that it needs to set the file permissions for the BookStack application files.
Note: Make sure to run the following command from AWS cli to get your rds endpoint.

```
aws rds describe-db-instances --db-instance-identifier my-rds-instance
--query 'DBInstances[0].Endpoint.Address' --output text
```

```
function run_update_bookstack_env() {
    cd "$BOOKSTACK_DIR" || exit
    cp .env.example .env
    sed -i 's/DB_HOST=localhost/DB_HOST=<your-rds-endpoint>/g' .env
    sed -i.bak "s@APP_URL=.*\$@APP_URL=http://$ip_a/@" .env
    sed -i.bak 's/DB_DATABASE=.*$/DB_DATABASE=bookstack/' .env
    sed -i.bak 's/DB_USERNAME=.*$/DB_USERNAME=bookstack/' .env
    sed -i.bak "s/DB_PASSWORD=.*\$/DB_PASSWORD=password/" .env

    php artisan key:generate --no-interaction --force
}

function run_bookstack_database_migrations() {
  cd "$BOOKSTACK_DIR" || exit
  php artisan migrate --no-interaction --force
}

function run_set_application_file_permissions() {
  cd "$BOOKSTACK_DIR" || exit
  chown -R "$SCRIPT_USER":www-data ./
  chmod -R 755 ./
  chmod -R 775 bootstrap/cache public/uploads storage
  chmod 740 .env
```

```
  # Tell git to ignore permission changes
  git config core.fileMode false
}
```

Now we can configure Nginx. We need to create a new Nginx virtual host configuration file for BookStack in the /etc/nginx/sites-available/ directory and link it to the sites-enabled directory to enable it.

```
function run_configure_nginx {
    cat >/etc/nginx/sites-available/bookstack <<EOL
    server {
      listen 80;
      listen [::]:80;

      server_name http://$ip_a/;

      root /var/www/bookstack/public;
      index index.php index.html;

      location / {
        try_files \$uri \$uri/ /index.php?\$query_string;
      }

      location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.1-fpm.sock;
      }
    }
EOL

    ln -s /etc/nginx/sites-available/bookstack /etc/nginx/sites-enabled/
    rm /etc/nginx/sites-enabled/default

    systemctl restart nginx
}
```

After this we can run all the functions in order.

```
echo "Installing required system packages (This may take several minutes)"
run_package_installs
echo "Downloading BookStack to $BOOKSTACK_DIR"
run_bookstack_download
```

```
echo "Installing Composer (PHP dependency manager)"
run_install_composer
echo "Installing PHP dependencies using composer"
run_install_bookstack_composer_deps
echo "Creating and populating BookStack .env file"
run_update_bookstack_env
echo "Running initial BookStack database migrations"
run_bookstack_database_migrations
echo "Setting BookStack file & folder permissions"
run_set_application_file_permissions
echo "Configuring nginx server"
run_configure_nginx


echo "BookStack has been installed and configured."
echo "Setup finished, your BookStack instance should now be installed!"
echo "Default login email: admin@admin.com"
echo "Default login password: password"
echo "You can access it at http://$ip_a"
```

After the BookStack script is finished we can run the infrastructure script. Once all the infrastructure is created we need to copy our BookStack script from our computer to the newly created instance. We can do this using sftp.

```
sftp -i bookstack.pem ubuntu@<ec2_public_ip>
put bookstack_installation_script
```

Then we can finally ssh into our ec2 instance and run the bookstack installation script. Before running the script make sure to give it execute permissions and then run it as root.

```
ssh -i bookstack,pem ubuntu@<ec2_public_ip>
chmod +x bookstack_installation_script
sudo ./bookstack_installation_script
```