Assignment 2- Creating secure, scalable and highly available Architecture

Introduction
The proposed architecture was designed to deploy Café site in two AWS Availability Zone from a Region, mounting at least 1 EC2 instances per AZ but subject to scale if needed up to 4 instances for Web App Server and Web Api EC2 instances to assure high availability. It was also deployed and RDS MariaDB database and created a Read Only replica in the same AZ for API read only access. VPC CIRD used is 10.0.0.0/16 and 4 subnets (2 publics and 2 privates). A bastion host was created to access EC2 instances using private key.

Solution Description: Web Application

Steps for WebAppServer EC2 instance formation:

1. Create a S3 storage and upload the zip file, give public access to download from EC2 User Data code.
2. Create a launch template Form EC2 Launch templates with these features:

- Name: WebAppServer-template
- **AMI**: Amazon Linux 2023 AMI 2023.4.20240513.0 x86_64 HVM kernel-6.1
- **Instance type**: *t2.micro*
- Create a new key pair and downloaded (to connect to Instances via ssh).
- Check on Autoscaling Guidance
- **Key pair (login)**: Uses a *new key pair*
- **Security groups**: WebAppSer-NSG
- **Resource tags**:
  - **Key**: Name
  - **Value**: WebAppSer
  - **Resource types**: *Instances*

In user data must add the following code to install apache webserver, php and mariadb client.

```
#!/bin/bash
yum update -y
yum install -y httpd php

sudo dnf update -y
sudo dnf install -y mariadb105-server php-mysqli

# Download ZIP files from my S3 bucket
wget https://a2chebucket.s3.amazonaws.com/cafe.zip -O /var/www/html/cafe.zip
```

```
cd /var/www/html
unzip -o cafe.zip

# Start and enable MariaDB service
sudo systemctl start mariadb
sudo systemctl enable mariadb

#Check and start apache
chkconfig httpd on
service httpd start
```

Steps for an WebAppServer Intances Autoscaling Group creation:

1. Create a new Auto Scaling Group that meets the following criteria:

    - **Launch template**: Uses the launch template that you created in the previous task
    - **VPC**: CafeVPC
    - **Subnets**: Uses PrivSubAZ1 and PrivSubAZ2
    - Skips *all* the advanced options
    - Enable group metrics collection within CloudWatch
    - Has a **Group size** configured as:
        - **Desired capacity**: 2
        - **Minimum capacity**: 2
        - **Maximum capacity**: 4
    - Enables the **Target tracking scaling policy** configured as:
        - **Metric type**: *Average CPU utilization*
        - **Target Value**: 50.0 CPU capacity

Steps for Creating a load balancer-manually

1. Create an HTTP Application Load Balancer that meets the following criteria:
    - **VPC**: Café-VPC
    - **Subnets**: Pub_Sub_AZ1 and Pub_Sub_AZ2
    - Skips the HTTPS security configuration settings
    - **Security group**: Creates ALB-NSG that allows HTTP traffic from any IPV4 source (internet)
    - **Target group**: Cafe-ALB-TG
    - Skips registering targets

Note: *Wait* until the load balancer is active.

2. Modify the Auto Scaling group that you created in the previous task by adding this new load balancer.

Subnet: Priv_Sub_AZ1 and Priv_Sub_AZ2

Security Group: WebAppSer-NSG, it will allow inbound traffic from any in the VPC via port 80.

Solution Description: API End Points

1. Create a launch template (similar to WebAppSer-template) but adding this user data code (the idea is to run app.py as a deamon when it starts the instance)

```
# To use on an AWS Linux instance as USER DATA

#!/bin/bash
# Install Python 3 and pip
sudo yum install -y python3-pip unzip
# Install Flask, requests, and MySQL connector
pip3 install flask requests mysql-connector-python
# Create the directory for the Flask app
mkdir -p /home/ec2-user/flask_app
cd /home/ec2-user/flask_app
# Download the zip file containing the app.py
wget https://a2chebucket.s3.amazonaws.com/cafe_api.zip -O cafe_api.zip
# Unzip the downloaded file
unzip cafe_api.zip
# Create a systemd service file for the Flask app
cat <<EOF > /etc/systemd/system/flaskapp.service
[Unit]
Description=Flask Application
[Service]
ExecStart=/usr/bin/python3 /home/ec2-user/flask_app/cafe_api/app.py
WorkingDirectory=/home/ec2-user/flask_app/
User=ec2-user
Group=ec2-user
Restart=always
Environment="PATH=/usr/bin"

[Install]
WantedBy=multi-user.target
EOF

# Reload systemd, enable and start the Flask service
sudo systemctl daemon-reload
```

sudo systemctl enable flaskapp
sudo systemctl start flaskapp

For TAG instance

- Key: Name
- Value: WebApiSer

2. Create a new Auto Scaling Group that meets the following criteria:

    - **Launch template**: Uses the launch template WebApiSer-Template that you created in the previous task
    - **VPC**: CafeVPC
    - **Subnets**: Uses PrivSubAZ1 and PrivSubAZ2
    - Skips *all* the advanced options
    - Has a **Group size** configured as:
        - **Desired capacity**: 2
        - **Minimum capacity**: 2
        - **Maximum capacity**: 4
    - Enables the **Target tracking scaling policy** configured as:
        - **Metric type**: *Average CPU utilization*
        - **Target Value**: 50.0 CPU capacity

3. Create the target Group named WebApi-TG.
4. Add listener to the ALB that was already created and select the WebApi-TG listening to port 5000.
5. Associate Autoscaling group with Web Application load balancer already created.
6. Create a Bastion Host via EC2 instance with IP access in Pub_Sub_AZ1, Bastion-NSG to allow inbound ssh port 22 traffic from

Solution Description: Database Component

The report should contain enough details of the solution architecture related to the Database component. This should include but is not limited to VPC and subnets, RDS instances, and security groups.

Create an RDS instance.

1. Create an RDS instance that complies with these specifications.
    - **Engine type**: *MariaDB*
    - **Templates**: *Free Tier*
    - **DB instance identifier**:cafe-db

- **Username**: admin
- **Password**: Re:Start!9
- **DB Instance Class**: *db.t3.micro*
- **Storage type**: *General Purpose (SSD)*
- **Allocated storage**: 20 GiB
- Do *not* create a standby instance
- Place it in the **CafeVPC**
    - **Subnet Group**: default, where the database is *not* publicly accessible.
- Choose existing **VPC security group** named DB-NSG and unselect the default security group.
- **Availability Zone**: Choose the first Availability Zone in the list. For example, if the Region is *us-east-1*, choose **us-east-1a**.
- **Database port**: Keep the default TCP port of *3306*.
- Enhanced monitoring must be clear.

2. Once created RDS, was temporary allowed remote connection to execute café_db with its respective tables and data using MySQL WorkBench (r2schools, 2023).
3. Form RDS management console it was created a Read only replica instance named café_dbR and the EndPoint was included in db config of cafe_api host parameter.

Acknowledge

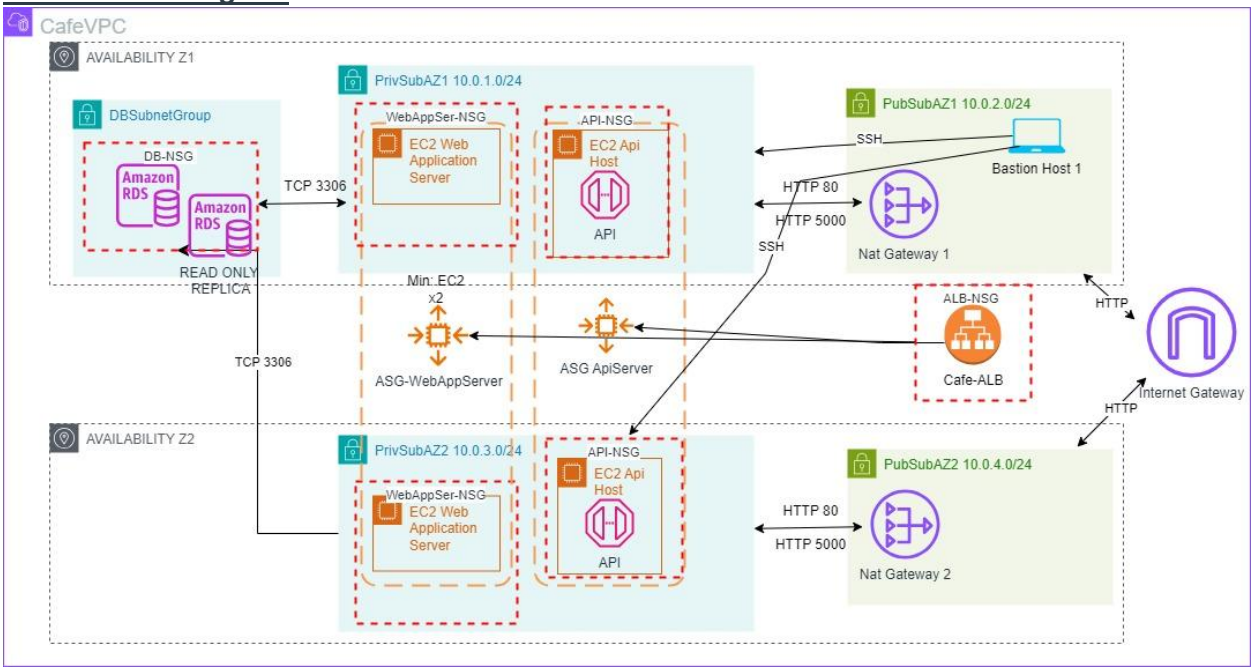
- r2schools. (2023, August 18). *41. MySQL DBA: How to create and connect to MariaDB RDS instance on AWS*. YouTube. https://www.youtube.com/watch?v=PY5otQe1mEs

- It was used AI ChatGPT for better understanding of code. Any code created by ChatGPT was used without doing several changes.

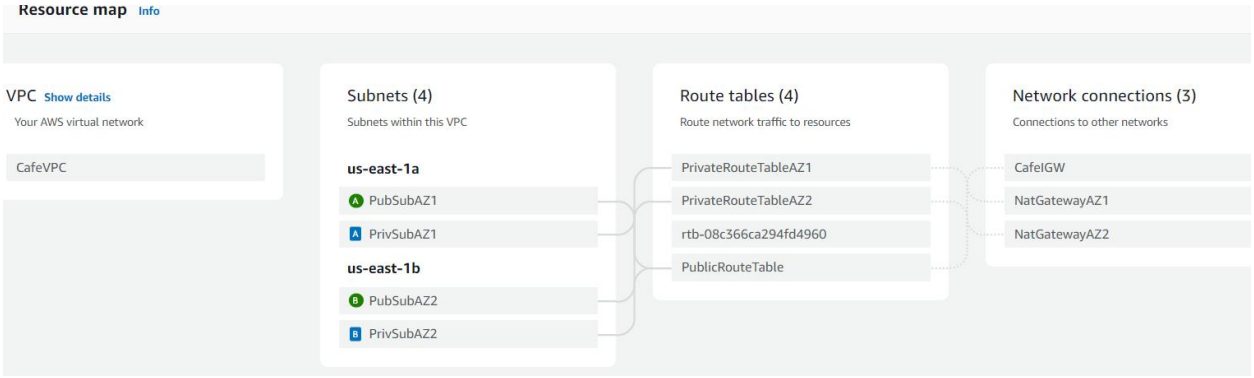
  VPC yaml creation prompting in ChatGPT

  create an AWS yaml template to create a VPC with CIDR 10.0.0.0/16, a private subnet with CIRD 10.0.1.0/24 called Priv_Sub_AZ1 in current availability zone connected to a Public Subnet with CIRD 10.0.2.0/24 called Pub_Sub_AZ1 with a Nat gateway connected to an Internet gateway service, a private subnet with CIRD 10.0.3.0/24 called Priv_Sub_AZ2 in another availability zone that I will edit later connected to a public subnet in the same second availability zone with CIRD 10.0.4.0/24 called Pub_Sub_AZ2 with a Nat gateway connected to the same Internet Gateway service indicated before

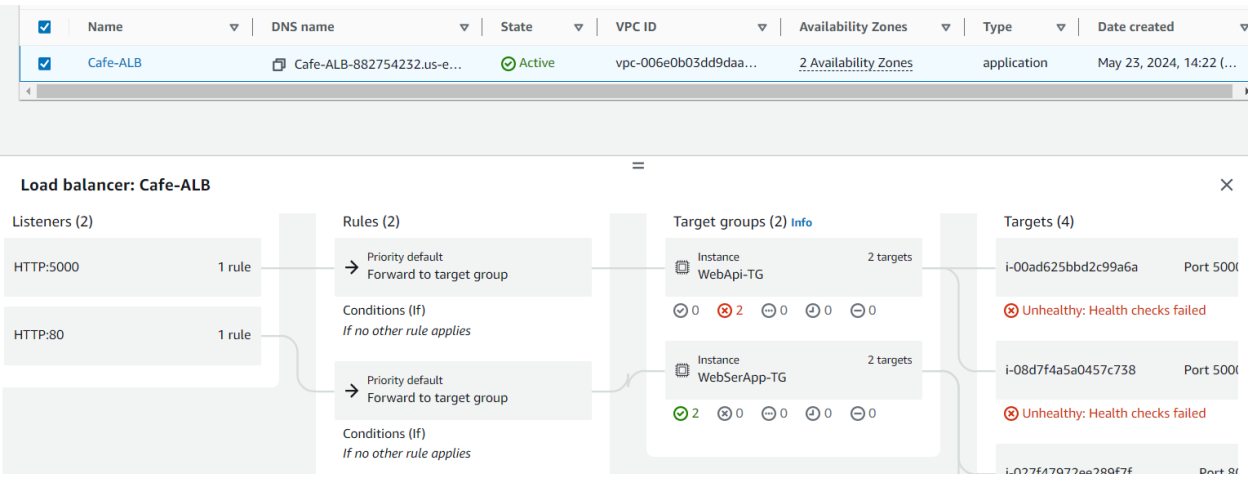
Architecture Diagram



- VPC and subnets

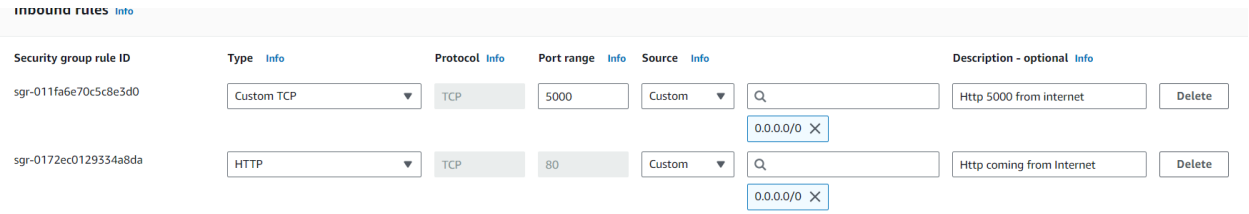


- Load balancer



-Security groups and their components

- The VPC named Café VPC is configured with the default ACL rules, also has a Network security Group VPC-NSG with an inbound rule that allows http port 80 to any IPV4 source, also an inbound rule that allows Custom TCP on port 5000 from Any IPV4 for traffic that comes from internet.
- The Network security group form Application load Balancer (Café-ALB) is named ALB-NSG and has as Inbound rules to allow traffic coming from Internet.



- API-NSG is a network security group for API EC2 instances, Inbound rules were created to limit the inbound Custom TCP port 5000 from ALB-NSG. Also traffic coming from Bastion Host via ssh (port 22) for configuration purposes. Inbound traffic coming from DB-NSG via port 3306 for database communication purposes.
- WebAppServ-NSG is a network security group created to limit traffic to EC2 Web App Server instances.
- DBSubnetGroup created when instance RDS was created, it groups the subnets of Café VPC
- DB-NSG is a network security group created to limit traffic to RDS database instances. Only allows Inbound traffic to 3306 coming from WebAppServ-NSG and API-NSG.

Resource Provision

NetworA2 stack must be configure first then create manually the resources.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: VPC with public and private subnets in two availability zones, with NAT gateways and an Internet gateway, including a database subnet with restricted access.

Resources:
 # Create VPC
 CafeVPC:
  Type: 'AWS::EC2::VPC'
  Properties:
   CidrBlock: '10.0.0.0/16'
   EnableDnsSupport: true
   EnableDnsHostnames: true
   Tags:
    - Key: Name
      Value: CafeVPC

 # Create Internet Gateway
 CafeIGW:
  Type: 'AWS::EC2::InternetGateway'
  Properties:
   Tags:
    - Key: Name
      Value: CafeIGW

 AttachGateway:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
   VpcId: !Ref CafeVPC
   InternetGatewayId: !Ref CafeIGW

 # Public Subnet in AZ1
 PubSubAZ1:
  Type: 'AWS::EC2::Subnet'
  Properties:
   VpcId: !Ref CafeVPC
   CidrBlock: '10.0.2.0/24'
   MapPublicIpOnLaunch: true
   AvailabilityZone: !Select [0, !GetAZs '']
   Tags:
    - Key: Name
      Value: PubSubAZ1

 # Private Subnet in AZ1
 PrivSubAZ1:
```

```
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref CafeVPC
    CidrBlock: '10.0.1.0/24'
    AvailabilityZone: !Select [0, !GetAZs '']
    Tags:
      - Key: Name
        Value: PrivSubAZ1

# Public Subnet in AZ2
PubSubAZ2:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref CafeVPC
    CidrBlock: '10.0.4.0/24'
    MapPublicIpOnLaunch: true
    AvailabilityZone: !Select [1, !GetAZs '']
    Tags:
      - Key: Name
        Value: PubSubAZ2

# Private Subnet in AZ2
PrivSubAZ2:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref CafeVPC
    CidrBlock: '10.0.3.0/24'
    AvailabilityZone: !Select [1, !GetAZs '']
    Tags:
      - Key: Name
        Value: PrivSubAZ2


# Public Route Table
PublicRouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref CafeVPC
    Tags:
      - Key: Name
        Value: PublicRouteTable

# Public Route
PublicRoute:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref CafeIGW

# Associate Public Subnets with Route Table
```

```
PubSubnetRouteTableAssociationAZ1:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref PubSubAZ1
    RouteTableId: !Ref PublicRouteTable

PubSubnetRouteTableAssociationAZ2:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref PubSubAZ2
    RouteTableId: !Ref PublicRouteTable

# EIP for NAT Gateway in AZ1
NatEIPAZ1:
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc

# NAT Gateway in AZ1
NatGatewayAZ1:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    SubnetId: !Ref PubSubAZ1
    AllocationId: !GetAtt NatEIPAZ1.AllocationId
    Tags:
      - Key: Name
        Value: NatGatewayAZ1

# EIP for NAT Gateway in AZ2
NatEIPAZ2:
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc

# NAT Gateway in AZ2
NatGatewayAZ2:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    SubnetId: !Ref PubSubAZ2
    AllocationId: !GetAtt NatEIPAZ2.AllocationId
    Tags:
      - Key: Name
        Value: NatGatewayAZ2

# Private Route Table for AZ1
PrivateRouteTableAZ1:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref CafeVPC
    Tags:
      - Key: Name
```

```
        Value: PrivateRouteTableAZ1

    # Private Route for AZ1
    PrivateRouteAZ1:
      Type: 'AWS::EC2::Route'
      Properties:
        RouteTableId: !Ref PrivateRouteTableAZ1
        DestinationCidrBlock: '0.0.0.0/0'
        NatGatewayId: !Ref NatGatewayAZ1

    # Associate Private Subnet with Route Table in AZ1
    PrivSubnetRouteTableAssociationAZ1:
      Type: 'AWS::EC2::SubnetRouteTableAssociation'
      Properties:
        SubnetId: !Ref PrivSubAZ1
        RouteTableId: !Ref PrivateRouteTableAZ1

    # Private Route Table for AZ2
    PrivateRouteTableAZ2:
      Type: 'AWS::EC2::RouteTable'
      Properties:
        VpcId: !Ref CafeVPC
        Tags:
          - Key: Name
            Value: PrivateRouteTableAZ2

    # Private Route for AZ2
    PrivateRouteAZ2:
      Type: 'AWS::EC2::Route'
      Properties:
        RouteTableId: !Ref PrivateRouteTableAZ2
        DestinationCidrBlock: '0.0.0.0/0'
        NatGatewayId: !Ref NatGatewayAZ2

    # Associate Private Subnet with Route Table in AZ2
    PrivSubnetRouteTableAssociationAZ2:
      Type: 'AWS::EC2::SubnetRouteTableAssociation'
      Properties:
        SubnetId: !Ref PrivSubAZ2
        RouteTableId: !Ref PrivateRouteTableAZ2


Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref CafeVPC

  PubSubAZ1Id:
    Description: Public Subnet ID in AZ1
    Value: !Ref PubSubAZ1
```

```
PrivSubAZ1Id:
  Description: Private Subnet ID in AZ1
  Value: !Ref PrivSubAZ1

PubSubAZ2Id:
  Description: Public Subnet ID in AZ2
  Value: !Ref PubSubAZ2

PrivSubAZ2Id:
  Description: Private Subnet ID in AZ2
  Value: !Ref PrivSubAZ2
```