COMP4349

Assignment 2: AWS Project Report

Ehab El Cheikh 490434606

Solution Architecture and Resource Provision

VPC

First a VPC was created called “project-vpc”. This VPC consists of four subnets with two public subnets and two private subnets. These are split into two different availability zones 1a and 1b. We then have 3 route tables associated with the subnets. The two public subnets are associated with the same route table whereas each private subnet has its own route table. These private subnet route tables then connect to their own NAT gateway whereas the public route table is connected to an internet gateway allowing access to the public subnets through the internet. The private subnets cannot be accessed directly thus we provide them a NAT gateway.

Security Groups

Next, we have the security groups:

- WebAppSecurityGroup

Type	IpProtocol	FromPort	ToPort	IP Ranges	Source
Inbound/Ingress	tcp	80	80		applb-sg
Inbound/Ingress	tcp	22	22		BastionHost-SG
Outbound/Egress				0.0.0.0/0	

- API-SG

Type	IpProtocol	FromPort	ToPort	IP Ranges	Source
Inbound/Ingress	tcp	22	22		BastionHost-SG
Inbound/Ingress	tcp	5000	5000		applb-sg
Outbound/Egress				0.0.0.0/0	

- applb-sg

Type	IpProtocol	FromPort	ToPort	IP Ranges	Source
Inbound/Ingress	tcp	80	80	My IP	
Inbound/Ingress	tcp	5000	5000	My IP	
Outbound/Egress				0.0.0.0/0	

- BastionHost-SG

Type	IpProtocol	FromPort	ToPort	IP Ranges	Source
Inbound/Ingress	tcp	22	22	My IP	
Outbound/Egress				0.0.0.0/0	

- RDS-SG

Type	IpProtocol	FromPort	ToPort	IP Ranges	Source
Inbound/Ingress	tcp	3306	3306		WebAppSecurityGroup
Inbound/Ingress	tcp	3306	3306		API-SG
Outbound/Egress				0.0.0.0/0	

Key pairs

Two key pairs were created:

- One for the bastion host called BastionHostKP
- One for the web app and API called webserver

Bastion Host

The bastion host is an EC2 instance with the following configuration:

- Type: an Amazon Linux 2023 AMI t2.micro instance
- VPC: the project-vpc as the VPC for the instance
- KeyPair: BastionHostKP
- Subnet: The subnet is a public subnet from either availability zone 1a or 1b.
- Security Group: The security group is the BastionHost-SG

Webapp

For the web application an EC2 instance was originally created and configured with all the files and settings before being converted into an AMI, where a launch template was created. For this original EC2 instance the configuration was:

- Type: an Amazon Linux 2023 AMI t2.micro instance
- VPC: the project-vpc as the VPC for the instance
- KeyPair: webserver
- Subnet: The subnet is a private subnet from either availability zone 1a or 1b.
- Security Group: The security group is the WebAppSecurityGroup which only allows SSH from the Bastion Host and the load balancer security group to access it from port 80.
- Userdata:
    - yum update -y
    - yum install -y unzip
    - systemctl start httpd
    - systemctl enable httpd

Following the creation of the EC2 instance, I used SSH to first connect to the bastion host. From the bastion host I then SSH into the EC2 instance. After verifying the

/var/www/html/ folders presence I uploaded the files from my pc for the webserver to the EC2 instance via this command:

- scp -i "webserver2" -o ProxyCommand="ssh -i "BastionHostKPUpdated2" ec2-user@bastionHostPublicIP -W %h:%p" "cafe_app_release-1.zip" ec2-user@webServerPrivateIP:

After this I unzipped the files then followed the same steps as assignment one. I moved the café folder to /var/www/html/. I updated the RDS url then ran the database creation script. After creating the load balancer I was able to verify the app was working.

An AMI was created from this using the default configuration. An EBS volume was attached with 8gb. Using this a launch template called webServerTemplate was created.

- AMI: The WebServerAMI was selected
- Type: t2.micro
- KeyPair: WebServer
- Subnet: no Subnet was selected so that the autoscaling group can distribute them across the availability zones
- Security Group: WebAppSecurityGroup
- Userdata:
    - yum update -y
    - yum install -y unzip
    - systemctl start httpd
    - systemctl enable httpd

Unzip would already be installed which makes this command redundant but the last two are important for starting the Apache webserver and the enable command will start the server automatically when the system boots.

Next we have the load balancer and autoscaling groups related to the web app.

The load balancer is called webserverLB and has the following configuration:

- Type: Application
- Scheme: Internet-facing
- VPC: project-vpc
- Security Group: applb-sg
- Availability Zones:
    - 1a - public subnet 1
    - 1b - public subnet 2
- Listener
    - Protocol&Port: HTTP:80
    - Target Group: app-lb-tg (webserver target group)

The target group is called app-lb-tg and has the following configuration:

- Type: Instance
- Protocol&Port: HTTP:80
- VPC: project-vpc
- Health checks: /cafe/index.php

The Auto Scaling group is called webserverscalinggroup and has the following config:

- Template: webServerTemplate
- Capacity:
    - Desired: 2
    - Minimum: 2
    - Maximum: 4
- VPC: project-vpc
- Subnets:
    - Private subnet 1
    - Private subnet 2
- Scaling policy: Target tracking with average CPU utilisation at 50%

API

Similarly, for the API an EC2 instance was originally created and configured with all the files and settings before being converted into an AMI, where a launch template was created. For this original EC2 instance the configuration was:

- Type: an Amazon Linux 2023 AMI t2.micro instance
- VPC: the project-vpc as the VPC for the instance
- KeyPair: webserver
- Subnet: The subnet is a private subnet from either availability zone 1a or 1b.
- Security Group: The security group is the API-SG which only allows SSH from the Bastion Host and the load balancer security group to access it from port 5000.
- Userdata:
    - yum update -y
    - yum install -y unzip
    - yum install -y python3 python3-pip
    - pip3 install flask mysql-connector-python

Following the creation of the EC2 instance, I used SSH to first connect to the bastion host. From the bastion host I then SSH into the EC2 instance. After verifying that pip and python had been installed I uploaded the files from my pc for the webserver to the EC2 instance via this command:

- scp -i "webserver2" -o ProxyCommand="ssh -i "BastionHostKPUpdated2" ec2-user@ bastionHostPublicIP -W %h:%p" "cafe_api_release-1.zip" ec2-user@ apiPrivateIP:

After this I unzipped the files then created a script to ensure the API ran on the instance startup. I moved the created a folder called /opt/myapp/ and copied the unzipped api files into there. The system file that I created to run the API on instance start consisted of the following

[Unit]

Description=My API Application

After=network.target

[Service]

Type=simple

WorkingDirectory=/opt/myapp

ExecStart=/usr/bin/python3 app.py

Restart=always

RestartSec=3

User=ec2-user

[Install]

WantedBy=multi-user.target

The service was started and verified using the command sudo systemctl status myapp.

An AMI called apiAMI2 was created from this using the default configuration. An EBS volume was attached with 8gb. Using this a launch template called apiLaunchTemplate was created.

- AMI: The apiAMI2 was selected
- Type: t2.micro
- KeyPair: WebServer
- Subnet: no Subnet was selected so that the autoscaling group can distribute them across the availability zones
- Security Group: API-SG
- Userdata:
    - sudo systemctl start myapp (A script was created that keeps the app running as soon as the instance runs using a script)

Next we have the load balancer and autoscaling groups related to the web app.

The load balancer is called apiLoadBalancer and has the following configuration:

- Type: Application
- Scheme: Internet-facing
- VPC: project-vpc
- Security Group: applb-sg
- Availability Zones:
    - 1a - public subnet 1
    - 1b - public subnet 2
- Listener
    - Protocol&Port: HTTP:5000
    - Target Group: api-tg2 (webserver target group)

The target group is called api-tg2 and has the following configuration:

- Type: Instance
- Protocol&Port: HTTP:5000
- VPC: project-vpc
- Health checks: /products

The Auto Scaling group is called apiautoscaler and has the following config:

- Template: apiLaunchtemplate
- Capacity:
    - Desired: 2
    - Minimum: 2
    - Maximum: 4
- VPC: project-vpc
- Subnets:
    - Private subnet 1
    - Private subnet 2
- Scaling policy: Target tracking with average CPU utilisation at 50%

Database

Similarly, for the API an EC2 instance was originally created and configured with all the files and settings before being converted into an AMI, where a launch template was created. For this original EC2 instance the configuration was:

- Creation method: Standard create
- Engine options: MySQL
- Version: 8.0.32
- Availability: Multi-AZ DB Instance
- Username: admin
- Password: as specified in the files
- Type: db.t3.micro

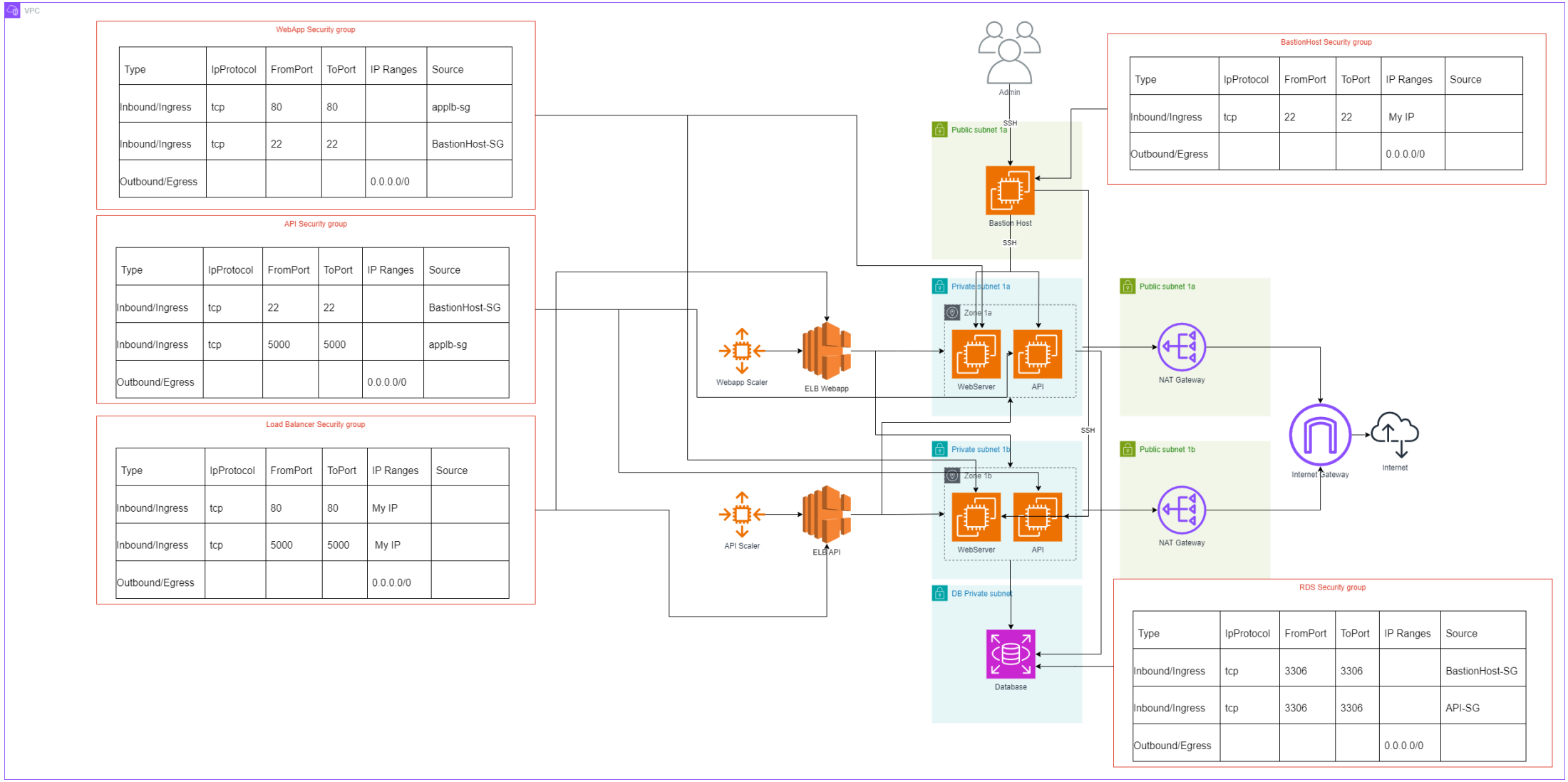
- VPC: project-vpc
- Security Group: RDS-SG

Demonstration Plan

Timestamp	Content
00:15 – 01:08	Slides of Architecture
01:08 – 01:41	AWS console VPC
01:41 – 02:44	Security Groups
02:44 – 03:20	Load Balancer
03:22 – 04:18	Target Groups
04:18 – 04:35	AutoScaling groups
04:35 – 06:04	Instances and SSH into private ec2 webserver instance. Run stress test. SSH into second webserver instance. Run stress test.
06:04 – 06:35	Finish explaining autoscaling groups
06:35 – 06:53	Database explanation
06:53 – 07:56	Website and API demo
07:56: end	Autoscaling of webserver in action

Architecture Diagram next page...

Architecture diagram



Appendix

Appendix A

AWS security groups

A	B	C	D	E	F	G	H	I	J
GroupId	GroupName	Type	IpProtocol	FromPort	ToPort	IpRanges	Ipv6Ranges	PrefixListIds	UserIdGroupPairs
sg-0d27b7d26afce0987	default	Inbound/Ingress	'-1						sg-0d27b7d26afce0987
sg-0d27b7d26afce0987	default	Outbound/Egress	'-1			0.0.0.0/0			
sg-0282276494b078e93	WebAppSecurityGroup	Inbound/Ingress	tcp	80	80				sg-054b1acbbbf607556
sg-0282276494b078e93	WebAppSecurityGroup	Inbound/Ingress	tcp	22	22				sg-0ff17c794e9fafa87
sg-0282276494b078e93	WebAppSecurityGroup	Outbound/Egress	'-1			0.0.0.0/0			
sg-010d6b99083213268	default	Inbound/Ingress	'-1						sg-010d6b99083213268
sg-010d6b99083213268	default	Outbound/Egress	'-1			0.0.0.0/0			
sg-0a16c02661fde511f	API-SG	Inbound/Ingress	tcp	22	22				sg-0ff17c794e9fafa87
sg-0a16c02661fde511f	API-SG	Inbound/Ingress	tcp	5000	5000				sg-054b1acbbbf607556
sg-0a16c02661fde511f	API-SG	Outbound/Egress	'-1			0.0.0.0/0			
sg-0ff17c794e9fafa87	BastionHost-SG	Inbound/Ingress	tcp	22	22	203.40.97.43/32			
sg-0ff17c794e9fafa87	BastionHost-SG	Outbound/Egress	'-1			0.0.0.0/0			
sg-054b1acbbbf607556	applb-sg	Inbound/Ingress	tcp	80	80	203.40.97.43/32			
sg-054b1acbbbf607556	applb-sg	Inbound/Ingress	tcp	5000	5000	203.40.97.43/32			
sg-054b1acbbbf607556	applb-sg	Outbound/Egress	'-1			0.0.0.0/0			
sg-0f039baa97bed28c6	RDS-SG	Inbound/Ingress	tcp	3306	3306				sg-0282276494b078e93
sg-0f039baa97bed28c6	RDS-SG	Inbound/Ingress	tcp	3306	3306				sg-0a16c02661fde511f
sg-0f039baa97bed28c6	RDS-SG	Outbound/Egress	'-1			0.0.0.0/0			