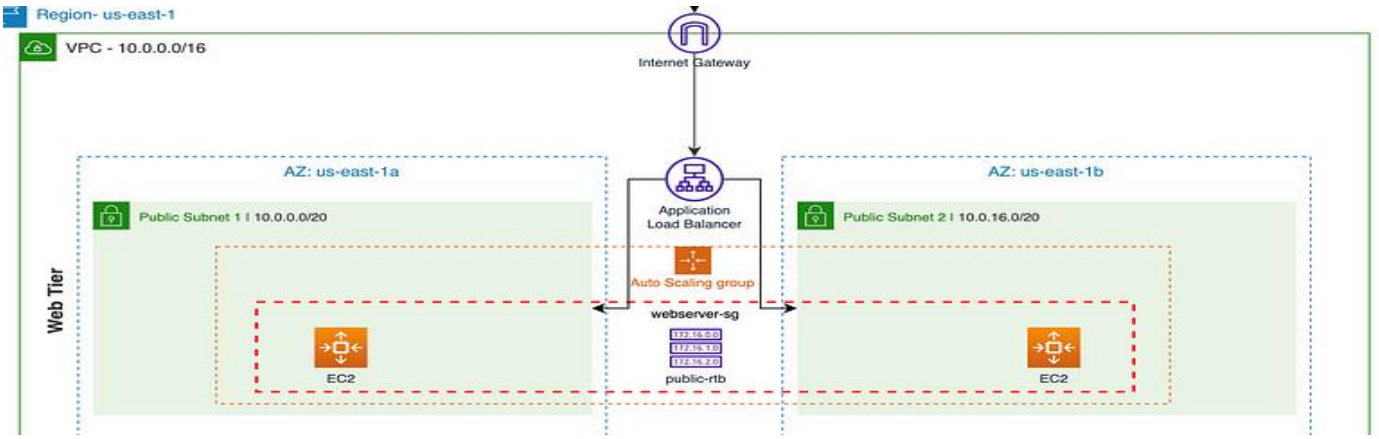
DOCUMENTATION OF CLOUD(AWS) PROJECT

Tier 1: Web tier (Frontend)

The Web Tier, also known as the 'Presentation' tier, is the environment where our application will be delivered for users to interact with. This is where we will launch our web servers that will host the frontend of our application.

What we will build:

1. A **web server launch template** to define what kind of EC2 instances will be provisioned for the application.
2. An **Auto Scaling Group (ASG)** that will dynamically provision EC2 instances.
3. An **Application Load Balancer (ALB)** to help route incoming traffic to the proper targets.



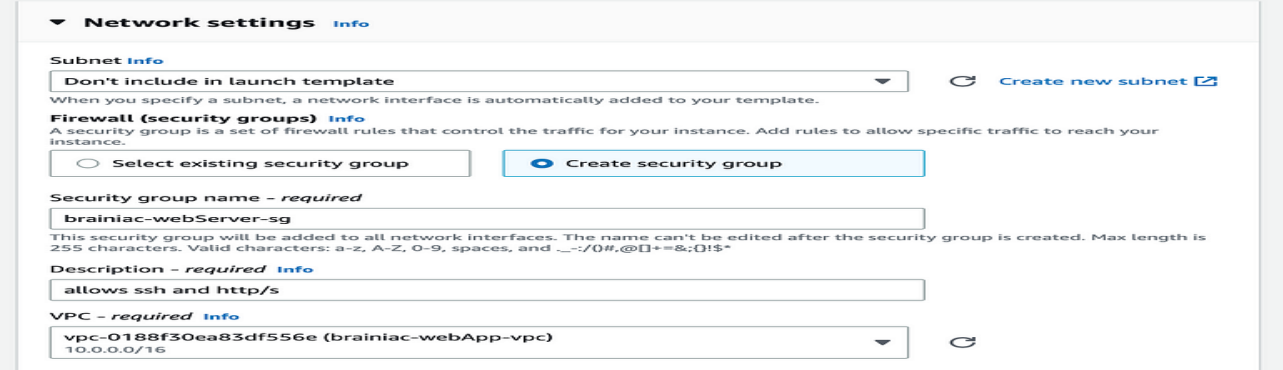
1. Create a web server launch template

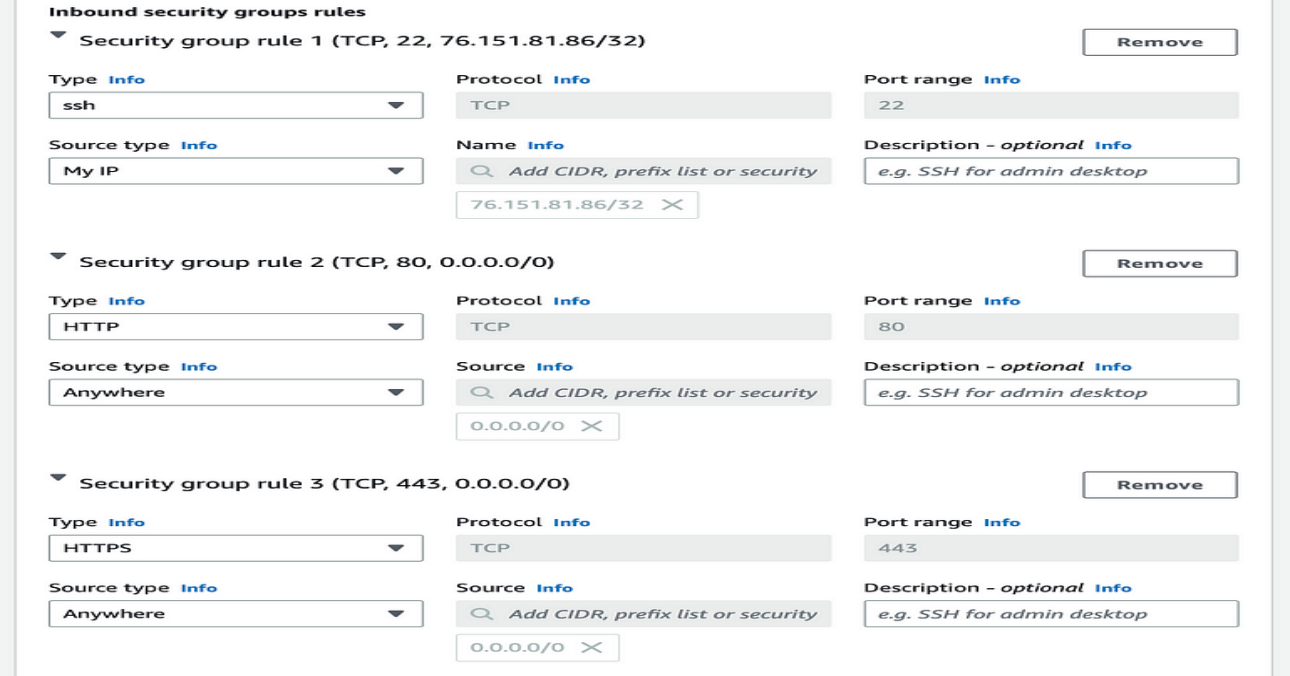
It is time to create a template that will be used by our ASG to dynamically launch EC2 instances in our public subnets.

In the EC2 console, navigate to '*Launch templates*' under the '*Instances*' sidebar menu. We're going to create a new template called '**frontend-web-server**' with the following provisions:

1. AMI: Amazon 2 Linux
2. Instance type: t2.micro (1GB – Free Tier)
3. A new or existing key pair

We are not going to specify subnets, but we will create a new security group with inbound **SSH, HTTP, and HTTPS** rules. Make sure the proper brainiac VPC is selected.



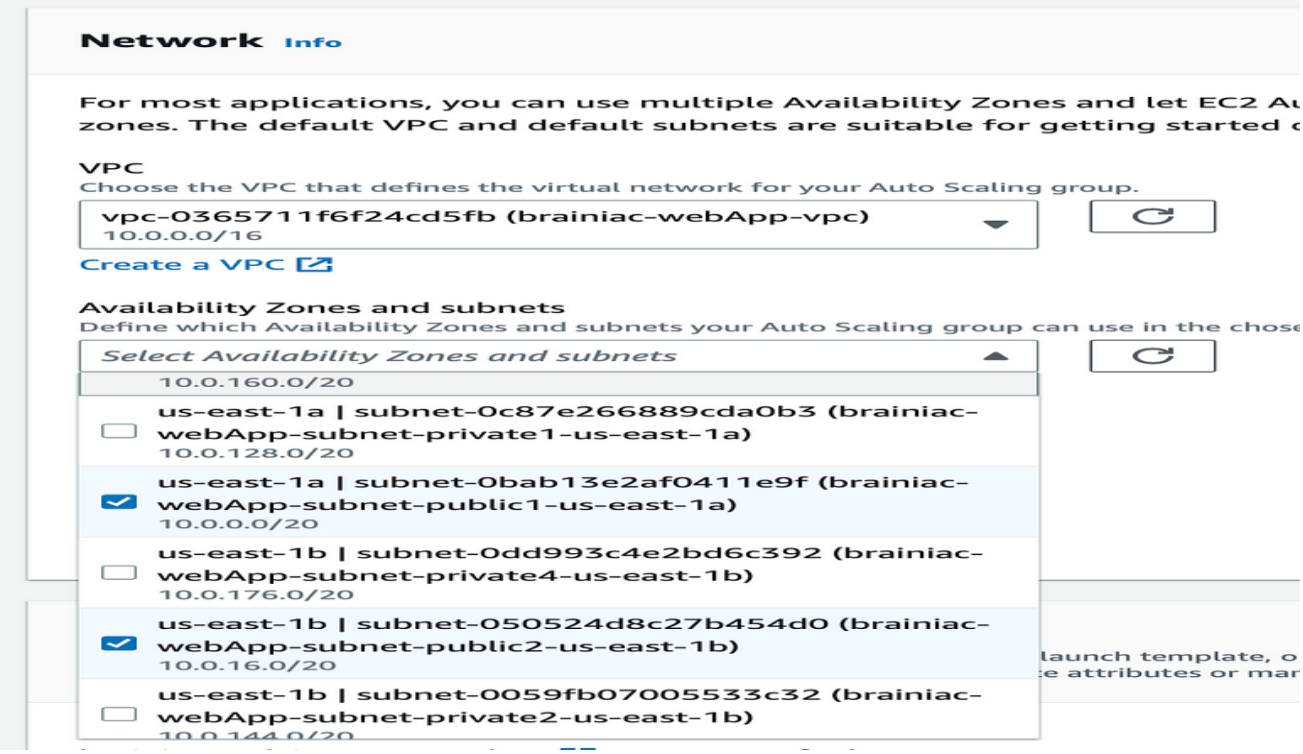


2. Create an Auto scaling group (ASG)

To ensure high availability for the Brainiac web app and limit single points of failure, we will create an ASG that will dynamically provision EC2 instances, as needed, across multiple AZs in our public subnets.

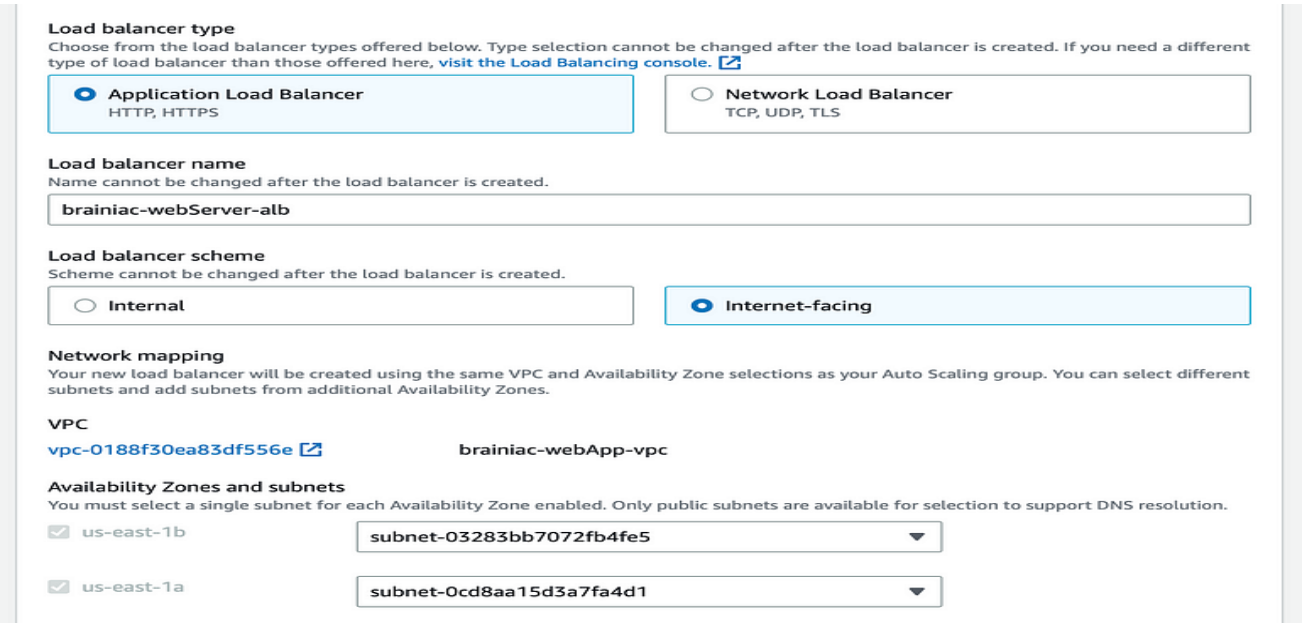
Navigate to the ASG console from the sidebar menu and create a new group. The ASG will use the **frontend-webserver-template** launch template we set up in the previous step.

Select the frontend-vpc along with the two public subnets.



3. Application load balancer (ALB)

We will need an ALB to distribute incoming HTTP traffic to the proper targets (our EC2s). The ALB will be named, '**frontend-webServer-alb.'** We want this ALB to be '*Internet-facing,*' so it can listen for HTTP/S requests.



The ALB needs to 'listen' over HTTP on port 80 and a target group that routes to our EC2 instances. We'll also add a dynamic scaling policy that tells the ASG when to scale up or down EC2 instances. For this build, we'll monitor the CPU usage and create more instances when the usage is above 50% (feel free to use whatever metric is appropriate for your application).

Group size
We want to set a minimum and maximum number of instances the ASG can provision:

- **Desired capacity:** 2
- **Minimum capacity:** 2
- **Maximum capacity:** 5

Review the ASG settings and create the group!

Once the ASG is fully initialized, we can go to our EC2 dashboard and see that two EC2 instances have been deployed.

To see if our ALB is properly routing traffic, let's go to its public DNS. We should be able to access the website we implemented when creating our EC2 launch template.

SSH

Let us confirm that we can SSH into our EC2 server.

```
Aaloks-MacBook-Pro:Keys aaloktrivedi$ ssh -i "webServer_key.pem" ec2-user@ec2-54-209-2
50-120.compute-1.amazonaws.com
The authenticity of host 'ec2-54-209-250-120.compute-1.amazonaws.com (54.209.250.120)'
 can't be established.
ED25519 key fingerprint is SHA256:AaP55Xce1YZQ6aABToTGdMhQo7GVsVG1Nx/w2kyYOGY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-209-250-120.compute-1.amazonaws.com' (ED25519) to t
he list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-28-148 ~]$
```

We've successfully built the architecture for the Web Tier for our wanderlust application! Remember, this is the 'Presentation' layer, where our users will directly interact with our app.

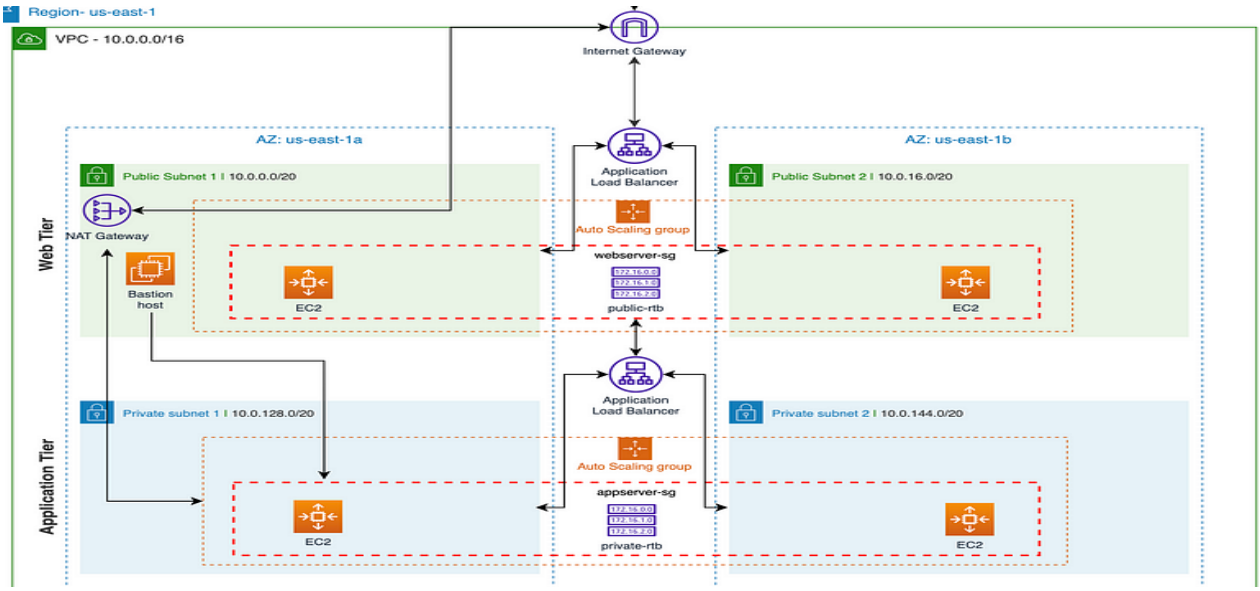
Tier 2: Application tier (Backend)

The Application Tier is essentially where the heart of our application lives. This is where the source code and core operations send/retrieve data to/from the Web and Database tiers.

The structure is very similar to the Web Tier but with some minor additions and considerations.

What we will build:

1. A **launch template** to define the type of EC2 instances.
2. An **Auto Scaling Group (ASG)** to dynamically provision EC2 instances.
3. An **Application Load Balancer (ALB)** to route traffic from the Web tier.
4. A **Bastion host** to securely connect to our application servers.

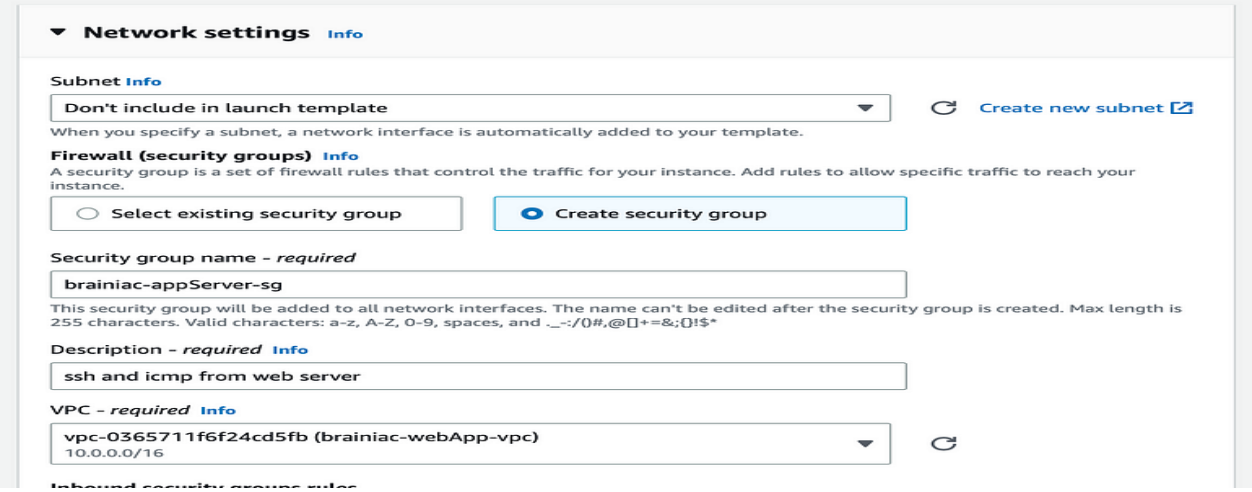


1. Create an application server launch template

This template will define what kind of EC2 instances our backend services will use, so let's create a new template called, '**backend-appServer-template.**'

We will use the same settings as the brainiac-webServer-template (Amazon 2 Linux, t2.micro-1GB, same key pair).

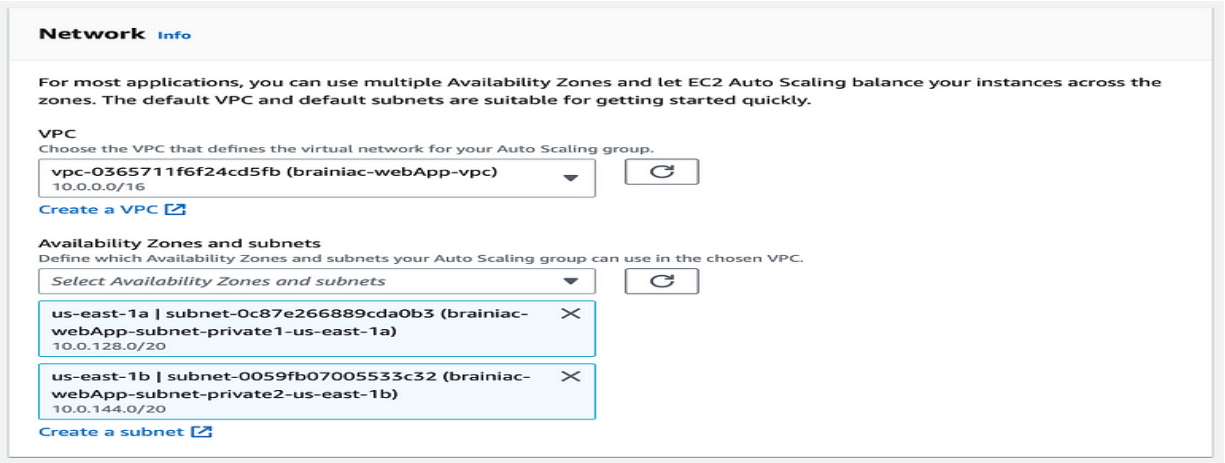
Our security group settings are where things will differ. Remember, this is a private subnet, where all our application source code will live. We need to take precautions so it cannot be accessible from the outside.



2. Create an Auto Scaling Group (ASG)

Like the Web Tier, we will create an ASG from the backend-appServer-template called, '**backend-server-asg.**'

Make sure to select the brainiac-vpc and the 2 private subnets (**subnet-private1 and subnet-private2**).



3. Application Load Balancer (ALB)

Now we'll create another ALB that routes traffic from the Web Tier to the Application Tier. We'll name it '**backend-server-alb.**'

This time, we want the ALB to be '*Internal*,' since we're routing traffic from our Web Tier, not the Internet.

We'll also create another target group that will target our appServer EC2 instances.

Confirm connectivity from the Web Tier

Our application servers are up and running. Let's verify connectivity by pinging the application server from one of the web servers.

SSH into the web server EC2 and ping the private IP address of one of the app server EC2s.

```
ssh -i "webServer_key.pem" ec2-user@ec2-54-209-250-120.compute-1.amazonaws.com
```

If successful, you should get a repeating response like this:

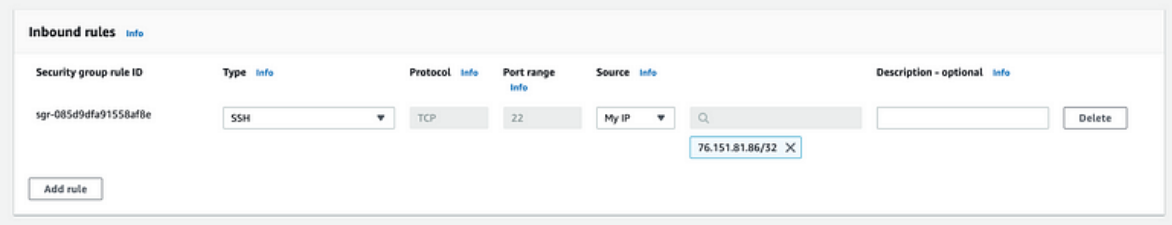
```
[ec2-user@ip-10-0-28-148 ~]$ ping 10.0.190.165
PING 10.0.190.165 (10.0.190.165) 56(84) bytes of data.
64 bytes from 10.0.190.165: icmp_seq=1 ttl=255 time=0.833 ms
64 bytes from 10.0.190.165: icmp_seq=2 ttl=255 time=0.439 ms
64 bytes from 10.0.190.165: icmp_seq=3 ttl=255 time=0.441 ms
64 bytes from 10.0.190.165: icmp_seq=4 ttl=255 time=0.423 ms
64 bytes from 10.0.190.165: icmp_seq=5 ttl=255 time=0.434 ms
64 bytes from 10.0.190.165: icmp_seq=6 ttl=255 time=0.405 ms
64 bytes from 10.0.190.165: icmp_seq=7 ttl=255 time=0.457 ms
64 bytes from 10.0.190.165: icmp_seq=8 ttl=255 time=0.509 ms
64 bytes from 10.0.190.165: icmp_seq=9 ttl=255 time=0.438 ms
64 bytes from 10.0.190.165: icmp_seq=10 ttl=255 time=0.493 ms
64 bytes from 10.0.190.165: icmp_seq=11 ttl=255 time=0.460 ms
64 bytes from 10.0.190.165: icmp_seq=12 ttl=255 time=0.448 ms
64 bytes from 10.0.190.165: icmp_seq=13 ttl=255 time=0.453 ms
ex64 bytes from 10.0.190.165: icmp_seq=14 ttl=255 time=0.438 ms
64 bytes from 10.0.190.165: icmp_seq=15 ttl=255 time=0.479 ms
64 bytes from 10.0.190.165: icmp_seq=16 ttl=255 time=0.798 ms
64 bytes from 10.0.190.165: icmp_seq=17 ttl=255 time=0.458 ms

64 bytes from 10.0.190.165: icmp_seq=18 ttl=255 time=0.438 ms
64 bytes from 10.0.190.165: icmp_seq=19 ttl=255 time=0.466 ms
^C
--- 10.0.190.165 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18419ms
```

We've successfully pinged the app server and received a response!

4. Create a Bastion host

A bastion host is a dedicated server used to securely access a private network from a public network. We want to protect our Application Tier from potential outside access points, so we will create an EC2 instance in the Web Tier, outside of the ASG. This is the only server that will be used as a gateway to our app servers.

In the EC2 console, launch a new instance called, '**app-bastionHost.'** We'll use the same provisions as before (Amazon Linux2, t2.micro).



We've successfully built the Application Tier architecture for our Wanderlust application! Remember, this is the 'Backend' layer, where our source code lives and backend operations send/retrieve data to/from the Web Tier and Database Tier.

Tier 3: Database tier (CRUD)

Since, I have used MongoDB for database. Hence, we will discuss how to connect MonogDB with AWS using EC2 instance with mongoose. Let's go….

1. Select EC2 Instance of backend-appServer template:

Security Group

Create or pick existing security group that has SSH port with your IP set for inbound rules

- Create new security group, eg. Mongodb-sg
- Type: SSH; Port 22; Source: your IP (so you can access the box directly later)

2. Download MongoDB

Update the packages

```
sudo apt -y update
```

Install MongoDB packages

```
sudo apt-get install mongodb-org
sudo systemctl start mongod
sudo systemctl enable mongod
```

Add MongoDB_URI to the AWS secrets Manager. That's all your MongoDB database is connected to the EC2 instance and is ready for all CRUD operations.

3.Install PM2 on your EC2 instance:

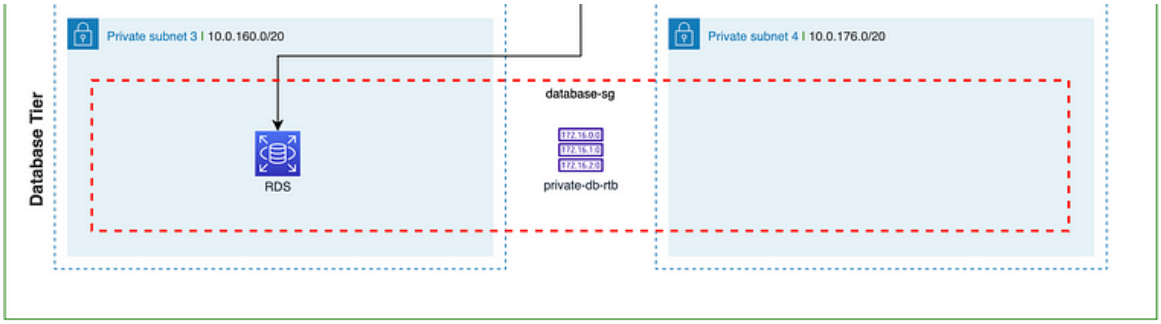
Process Manager (PM2) is a npm package that ensures all the services are running even when the system is switched off.

```
# install pm2 with npm

sudo npm install -g pm2

# set pm2 to start automatically on system startup

sudo pm2 startup systemd
```

We'll also see how to set database on AWS i.e using AWS RDS which is a SQL database.

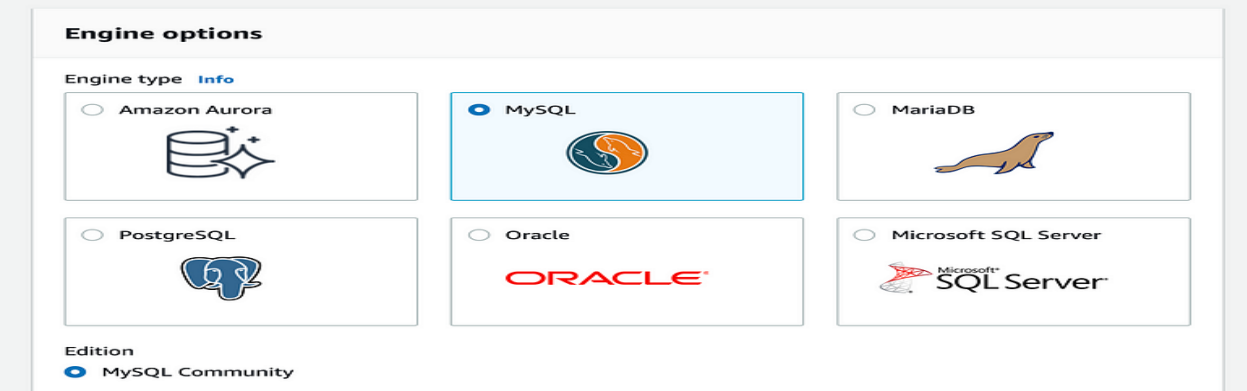
What we'll build:

1. A **database security group** that allows outbound and inbound mySQL requests to and from our app servers.
2. A **DB subnet group** to ensure the database is created in the proper subnets.
3. An **RDS database** with MySql.



1. Create an RDS database

Under the RDS console and the '*Databases*' sidebar menu, create a new database with a MySQL engine.



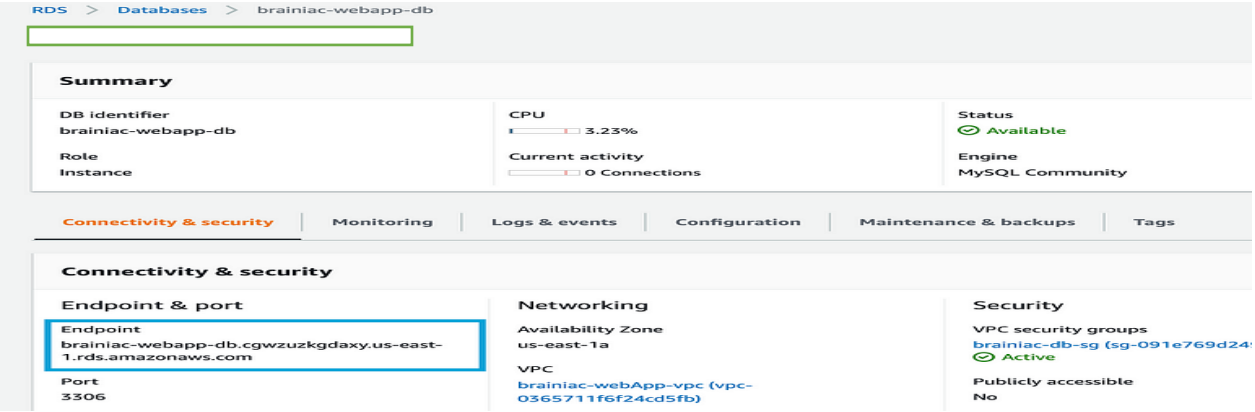
For '*Instance configuration*,' we'll use a **db.t2.micro** and leave the defaults for '*Storage.*'

For '*Connectivity,*' we do not want to connect an EC2 instance but make sure the **appServer-vpc** is selected.

Ensure to set Security group for the database same way as done in the previous two tiers above.

Connect to the Database

After the DB has been created, we'll need the database endpoint to establish a connection from the app server.



If you haven't yet, SSH into the app server through our bastion host.

We should already have mySQL installed on the server, so we can run this command:

mysql -h YOUR_DB_ENDPOINT -P 3306 -u YOUR_DB_USERNAME -p

When prompted, enter the password you chose when creating the DB.

```
[ec2-user@ip-10-0-156-224 ~]$ mysql -h brainiac-webapp-db.cgwzuzkgdaxy.us-east-1
.rds.amazonaws.com -P 3306 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 115
Server version: 8.0.28 Source distribution

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

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

MySQL [(none)]>
```

Great! We successfully connected to our database from our application server!