**AWS 3-Tier Web Application Documentation**

*Designed and implemented by Aditya Ghatol*

# Table of Contents

|  |  |  |
| --- | --- | --- |
| Index | contents | Page no |
| 1 | 1. Introduction |  |
| 2 | 2. Architecture Overview |  |
| 3 | 3. Technologies Used |  |
| 4 | 4. Step-by-Step Implementation |  |
| 4.1 | 4.1 EC2 Instance Setup |  |
| 4.2 | 4.2 Application Load Balancer (ALB) Setup |  |
| 4.3 | 4.3 Auto Scaling Group Setup |  |
| 4.4 | 4.4 Amazon RDS Setup |  |
| 5 | 5. Best Practices |  |
| 6 | 6. Screenshots |  |
| 7 | 7. Conclusion |  |

# 1. Introduction

# In today's landscape, where web applications are crucial for business operations and user engagement, building scalable and resilient applications is more important than ever. This project focuses on designing and implementing a 3-tier architecture for a web application using Amazon Web Services (AWS), a leading cloud computing platform renowned for its reliability, flexibility, and extensive service offerings.

# Presentation Layer: Hosted on Amazon EC2, this layer manages the user interface and experience.

# Application Layer: Utilizes Auto Scaling Groups to handle application logic and dynamically adjust capacity based on demand.

# Database Layer: Managed by Amazon RDS, it provides reliable data storage with high availability and automated backups.

# An Application Load Balancer (ALB) is employed to distribute traffic across multiple instances, preventing any single instance from becoming a bottleneck. This ensures the application remains responsive even under high traffic conditions.

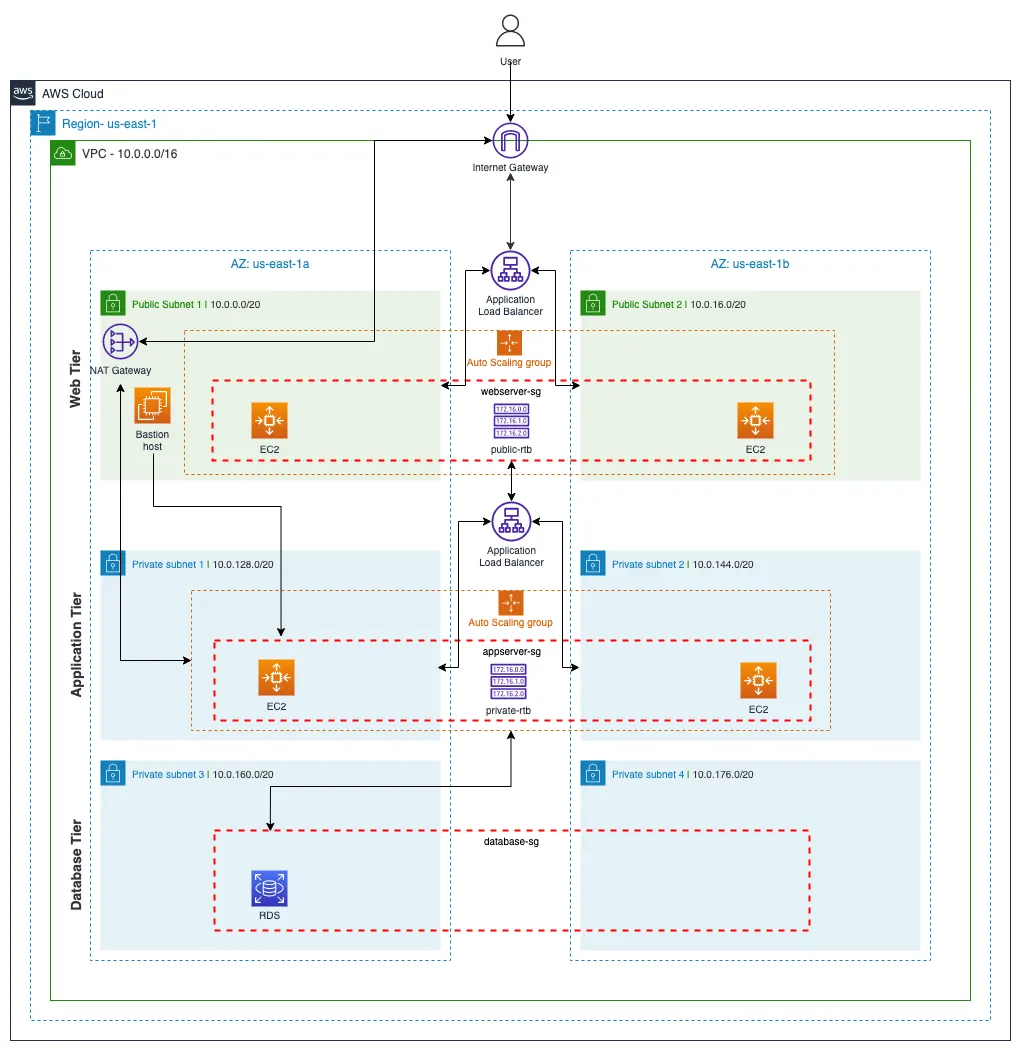
# By integrating these AWS services, the project designs a resilient and scalable web application that meets modern standards for availability and security. It addresses key challenges such as managing fluctuating workloads, maintaining performance, and protecting user data. Following AWS best practices, this project demonstrates how AWS's robust cloud services can be leveraged to build a powerful 3-tier architecture that exceeds contemporary web application expectations. This version keeps the essential details while improving readability.

# 2. Architecture Overview

The AWS 3-Tier architecture consists of the following components:

1. Web Tier: Amazon EC2 instance to host the frontend.

2. Application Tier: Application Load Balancer (ALB) to route traffic to instances.

3. Database Tier: Amazon RDS MySQL for persistent data storage. 

# 3. Technologies Used

 **Amazon EC2 (Elastic Compute Cloud):** Provides scalable compute capacity in the cloud, enabling the hosting of web applications. EC2 instances run the application code and handle the presentation layer, offering flexibility to choose different instance types based on performance needs.

 **Auto Scaling Groups:** Automatically adjusts the number of EC2 instances in response to varying traffic loads. This technology ensures that the application can scale up to handle high traffic and scale down during lower demand periods, optimizing cost and maintaining performance.

 **Application Load Balancer (ALB):** Distributes incoming application traffic across multiple EC2 instances. ALB improves the application's availability and fault tolerance by ensuring that no single instance bears too much load. It also supports advanced routing features, such as path-based routing, to direct traffic to specific instances based on URL paths.

 **Amazon RDS (Relational Database Service):** A managed database service that simplifies database management tasks such as backups, patching, and scaling. Amazon RDS supports MySQL, providing a reliable and secure database solution for storing application data. It ensures high availability through Multi-AZ deployments and automated backups, reducing administrative overhead and enhancing data durability.

# 4. Step-by-Step Implementation

## 4.1 EC2 Instance Setup

1. Launch an EC2 instance with Amazon Linux 2 AMI.

## 4.2 Application Load Balancer (ALB) Setup

1. Create an Application Load Balancer on the AWS console.  
2. Configure it to listen on port 80 and distribute traffic across EC2 instances.  
3. Test ALB by accessing the DNS name in a browser.

## 4.3 Auto Scaling Group Setup

1. Create an Auto Scaling Group for EC2 instances.  
2. Set scaling policies to handle traffic spikes by adding or removing instances based on CPU utilization.

## 4.4 Amazon RDS Setup

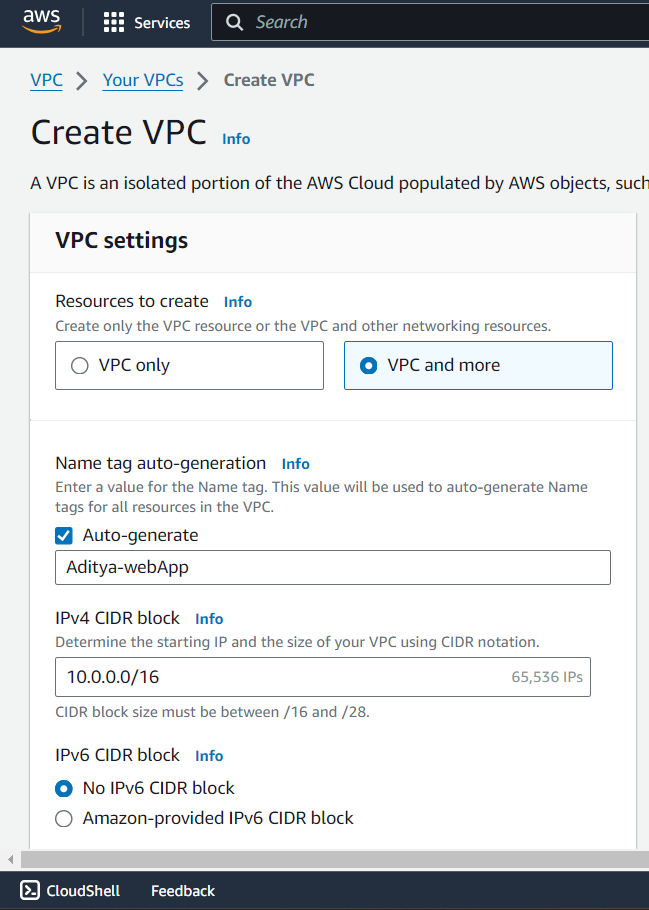
1. Create a MySQL database on RDS

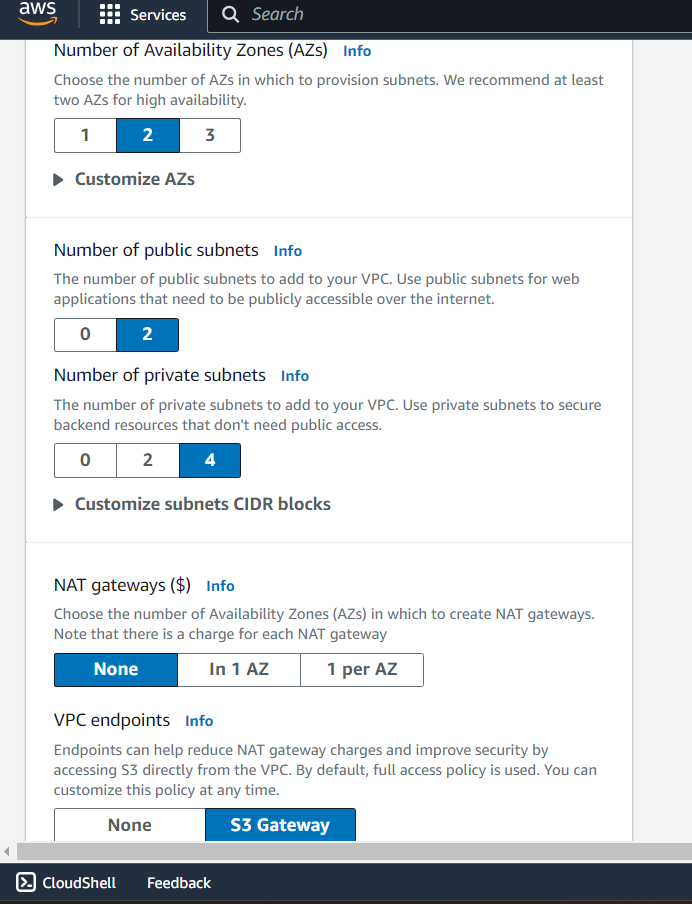
# 5. Best Practices

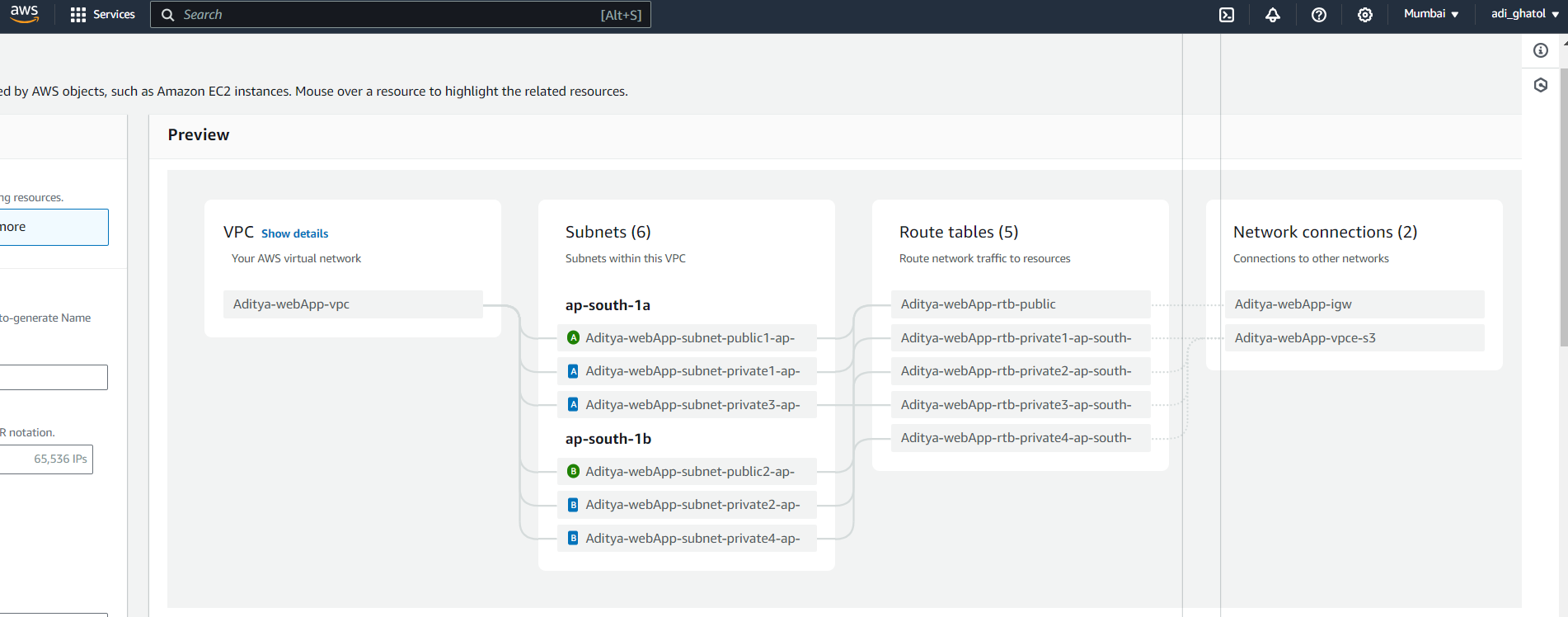
- Security: Configured security groups to allow only necessary traffic.  
- High Availability: Auto Scaling ensures the system handles load efficiently.  
- Fault Tolerance: ALB distributes traffic across healthy instances.  
- Backup and Recovery: Enabled automated backups for RDS.

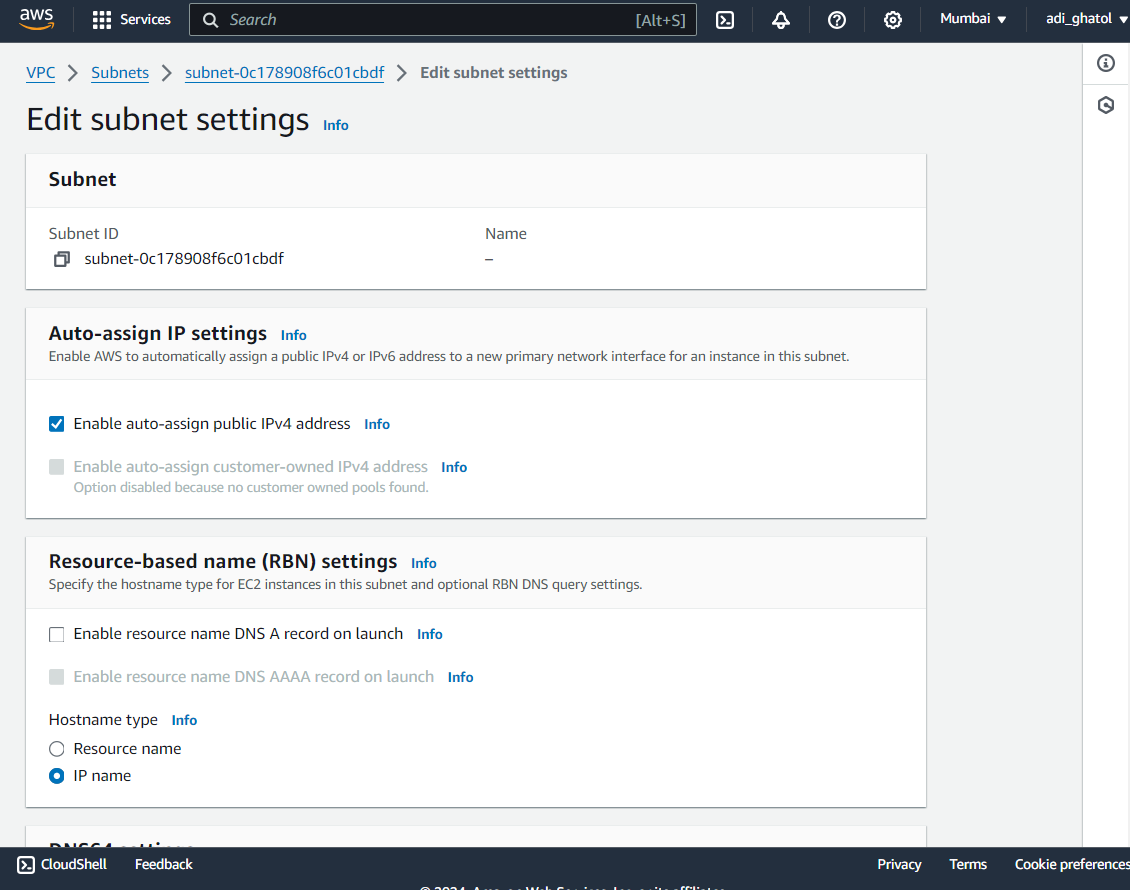
# 6. Screenshots

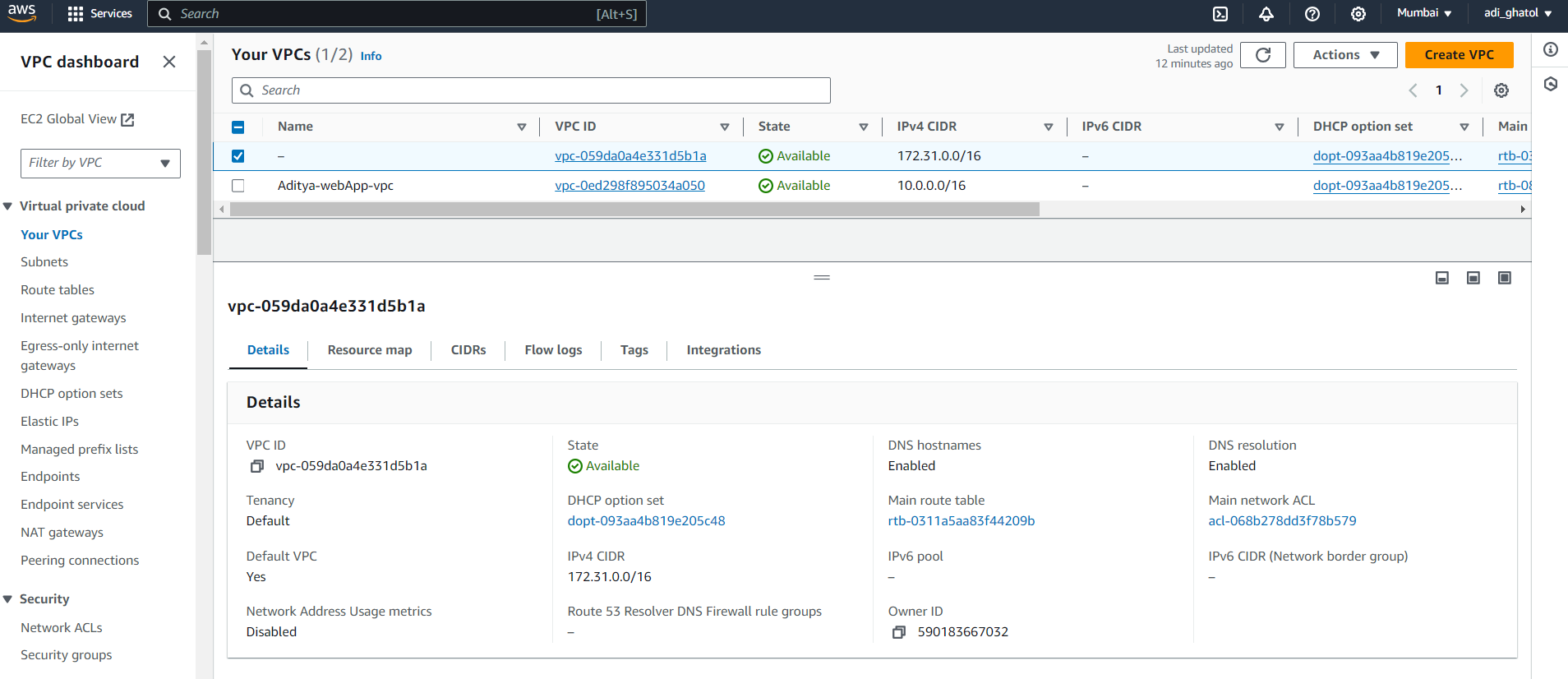
Insert relevant screenshots here for each step (EC2 setup, ALB, Auto Scaling, RDS).

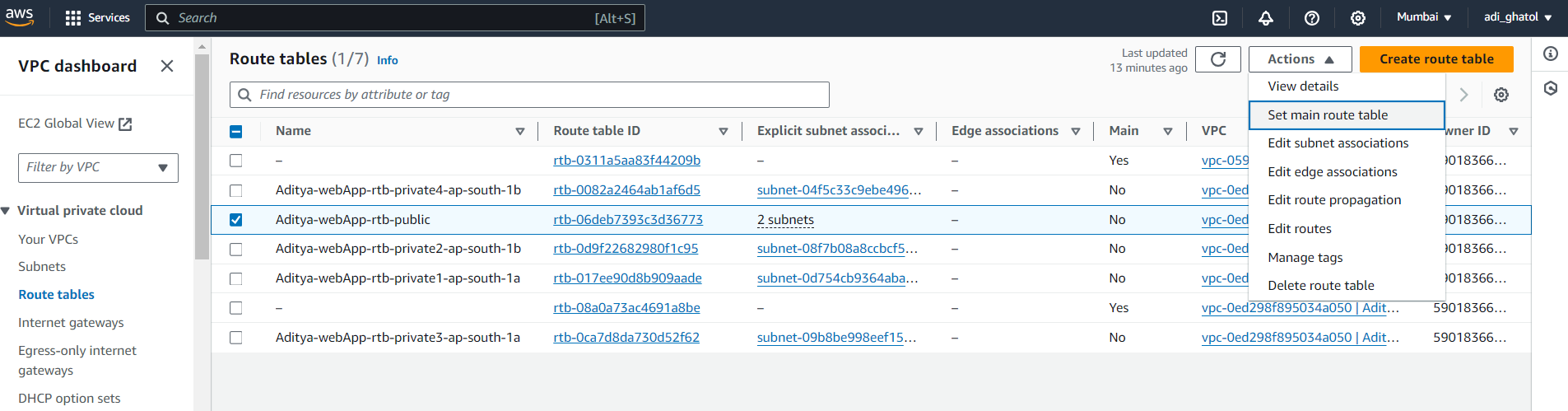


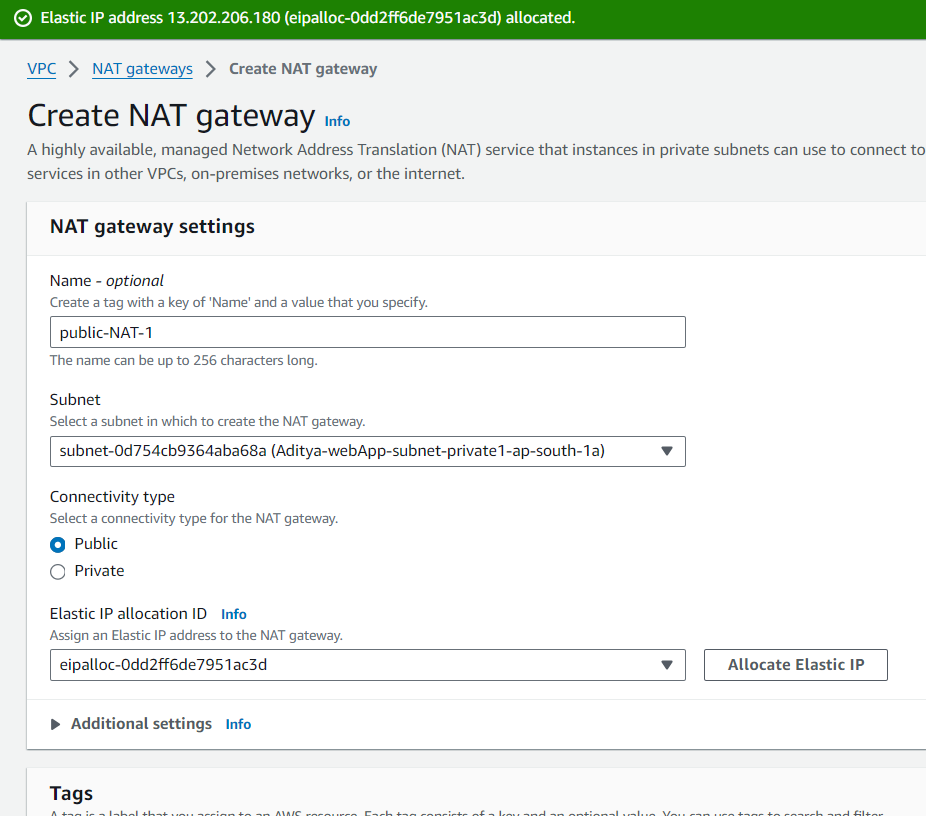


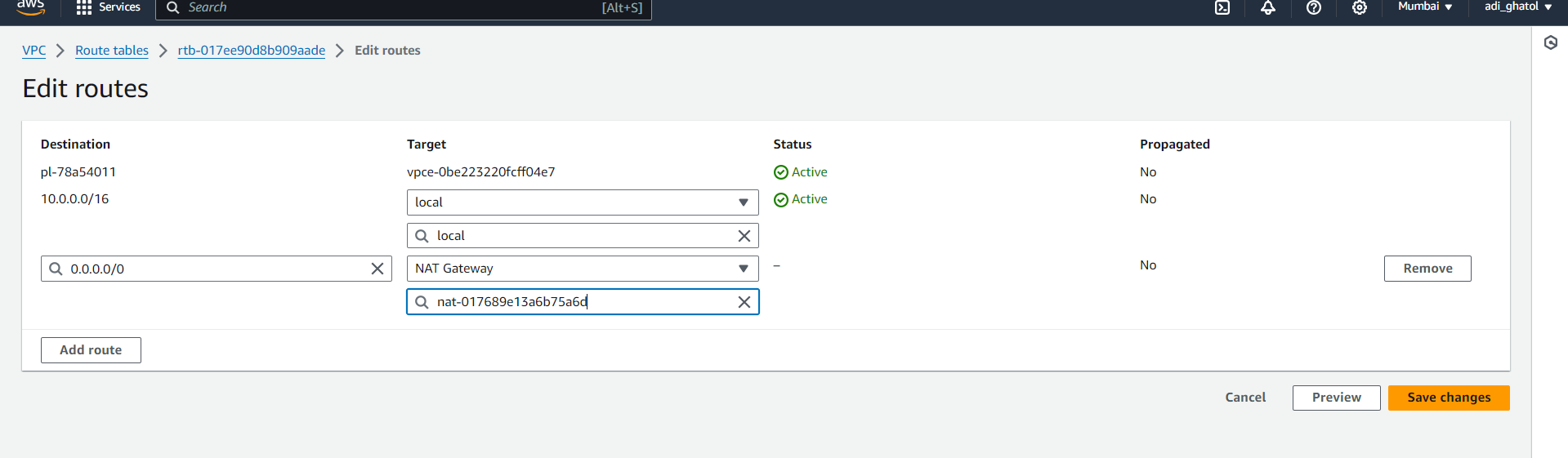


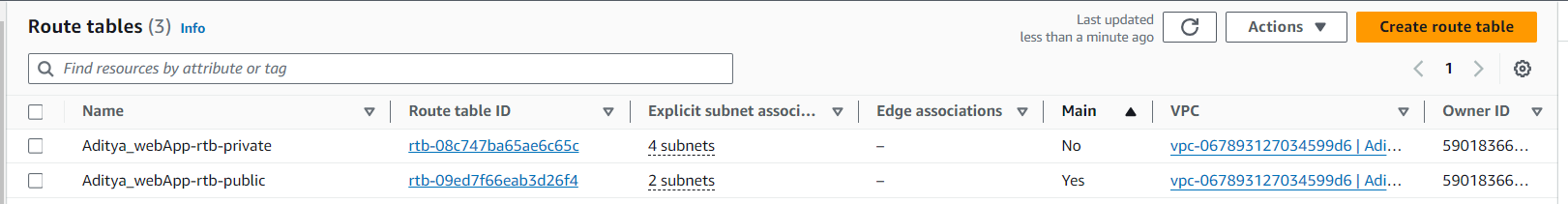










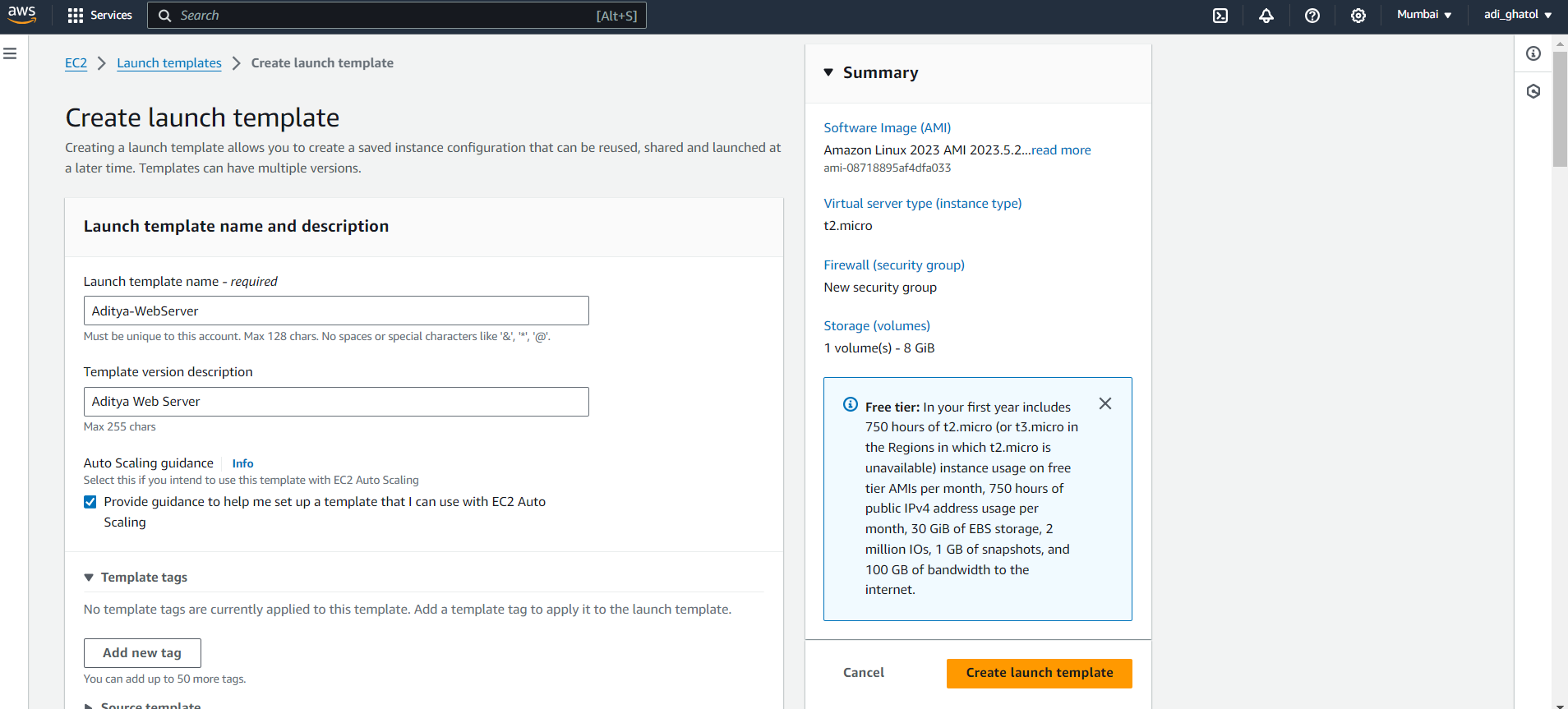


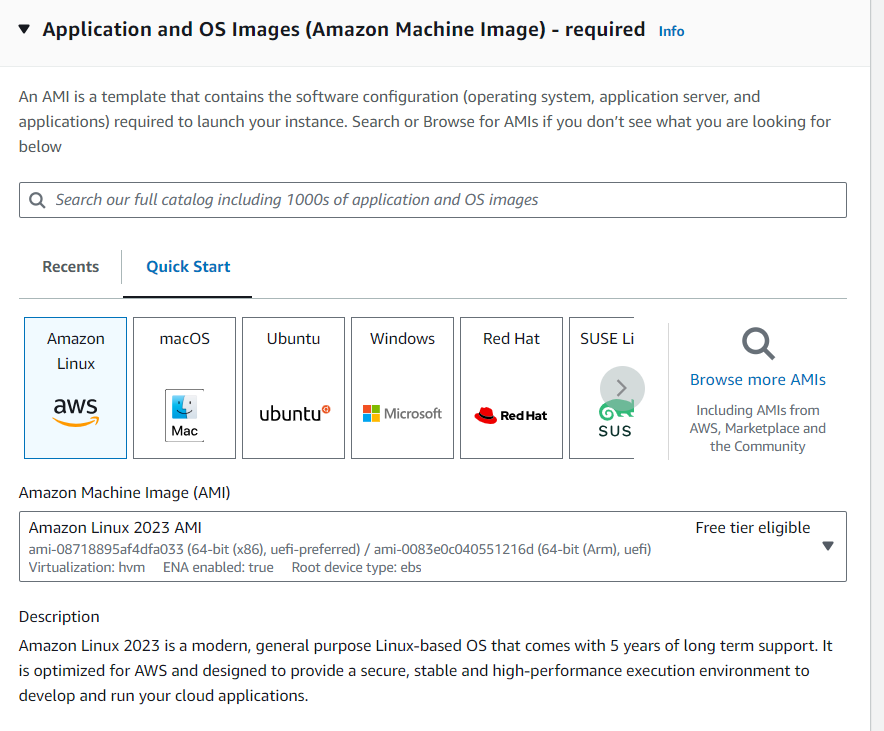
Tier 1: Web tier (Frontend)

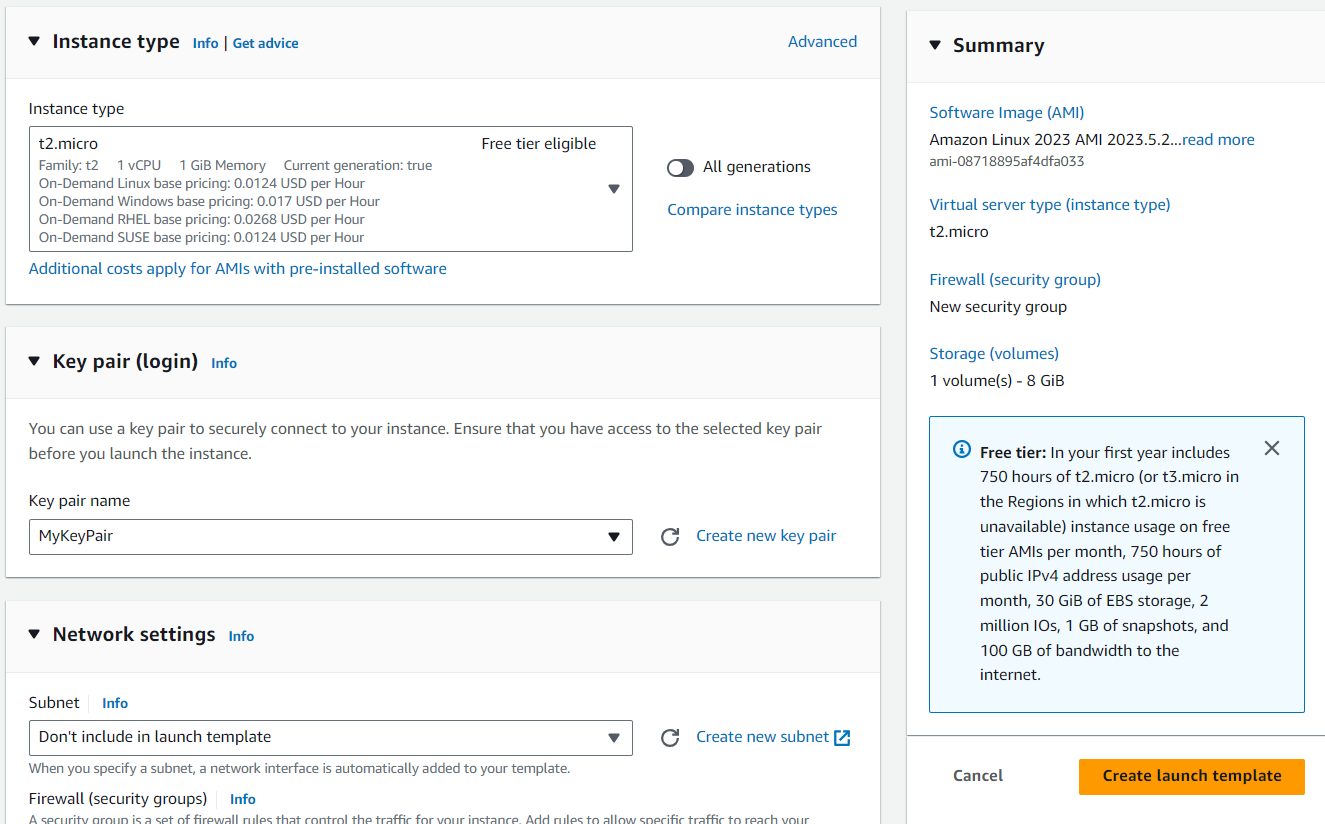
The Web Tier, also known as the ‘Presentation’ tier, is the environment where our application will be delivered for users to interact with. For Brainiac, this is where we will launch our web servers that will host the frontend of our application.

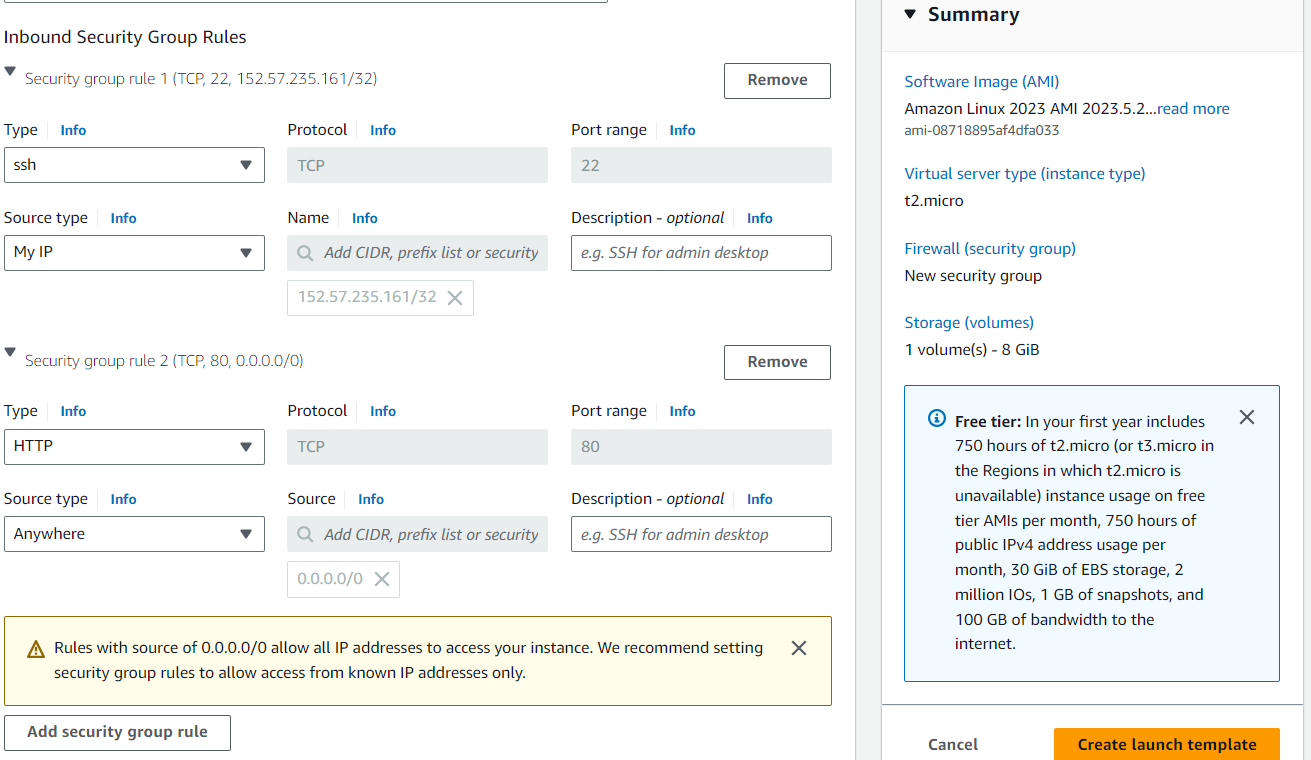
What we’ll build:

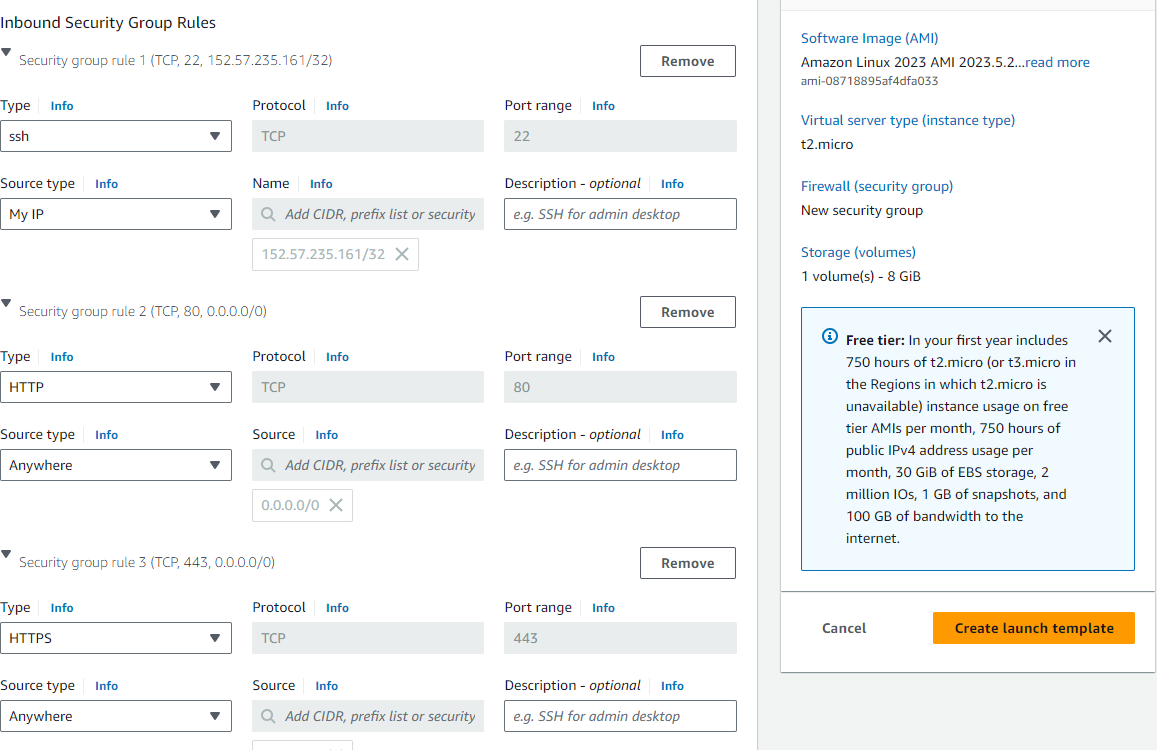
1. A web server launch template to define what kind of EC2 instances will be provisioned for the application.

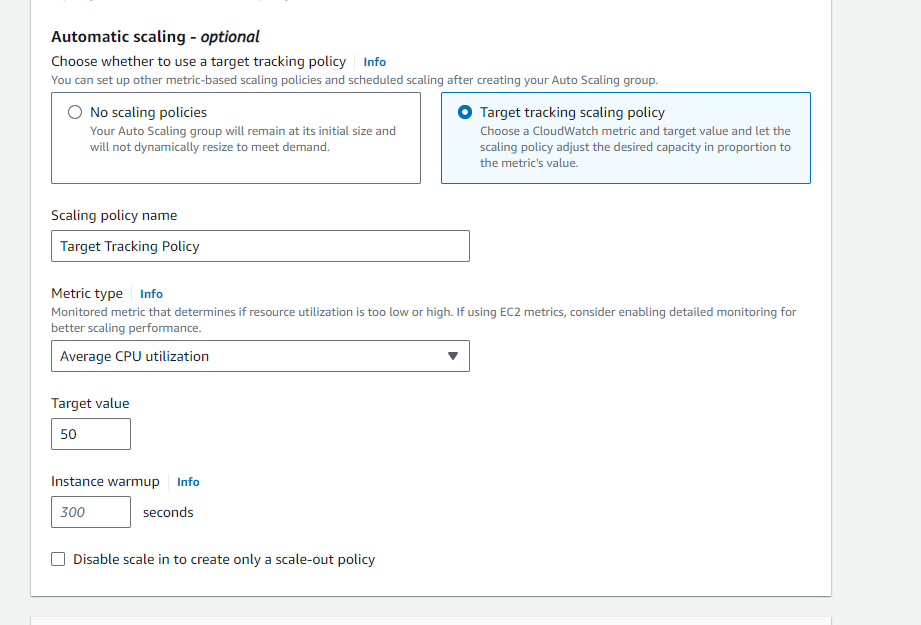


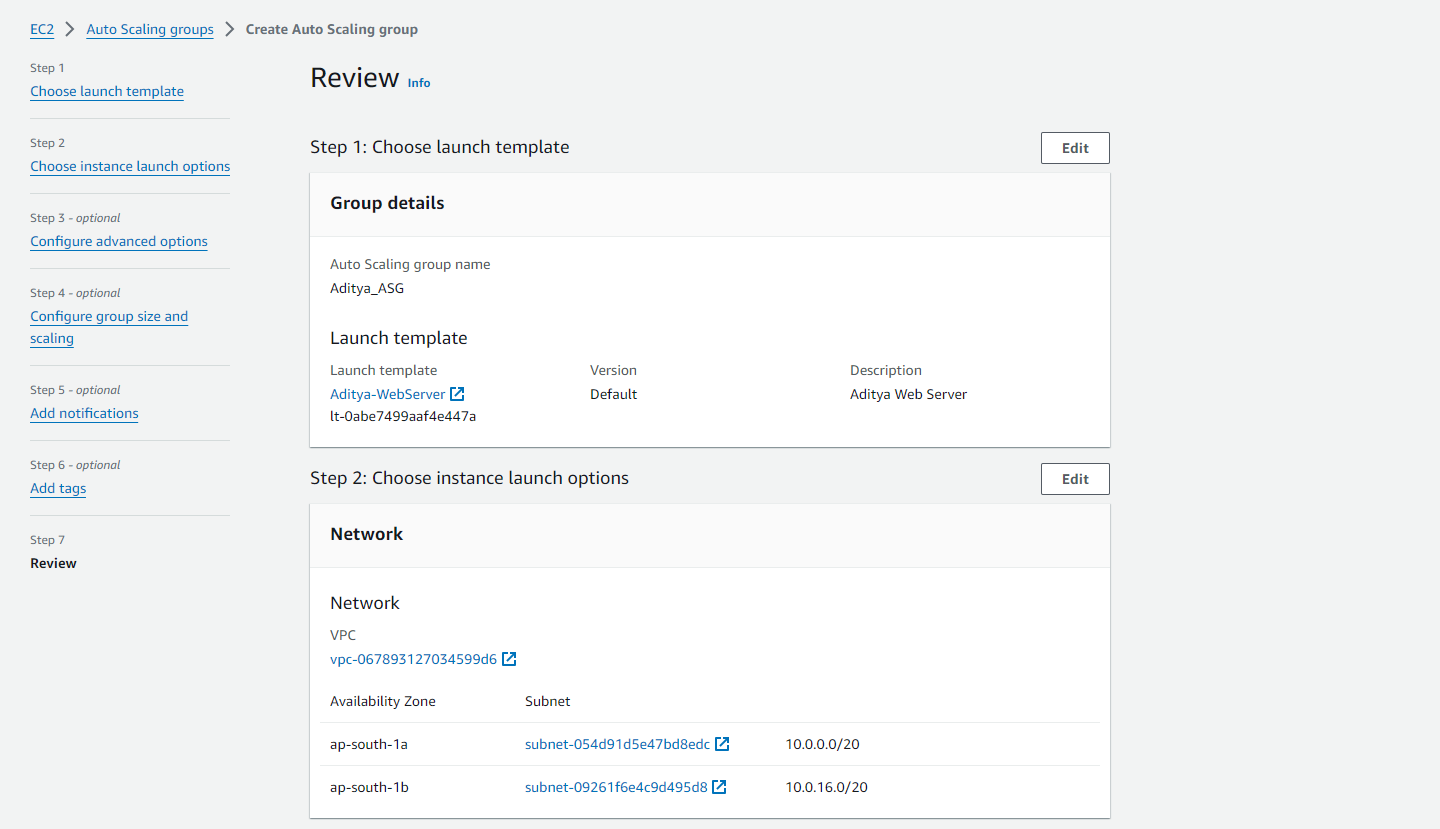


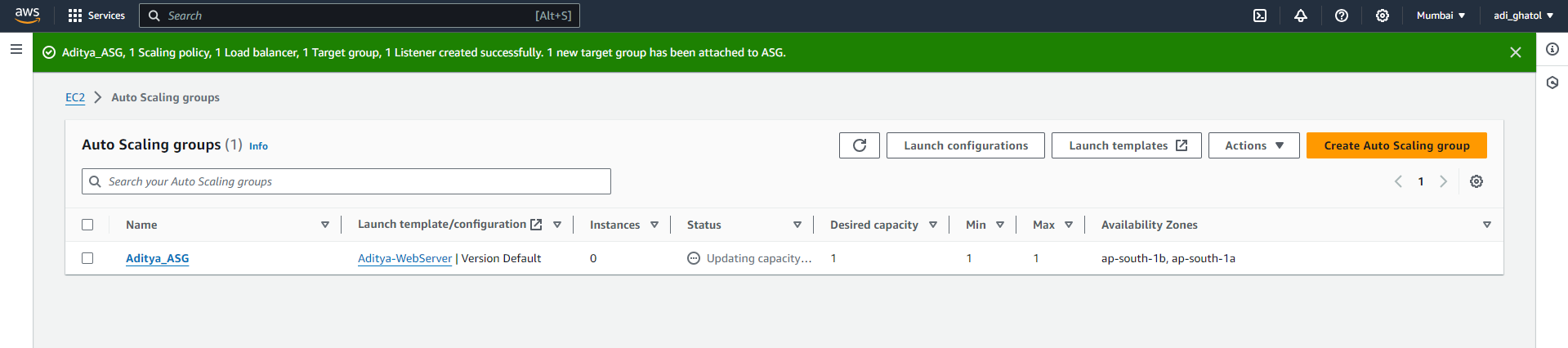


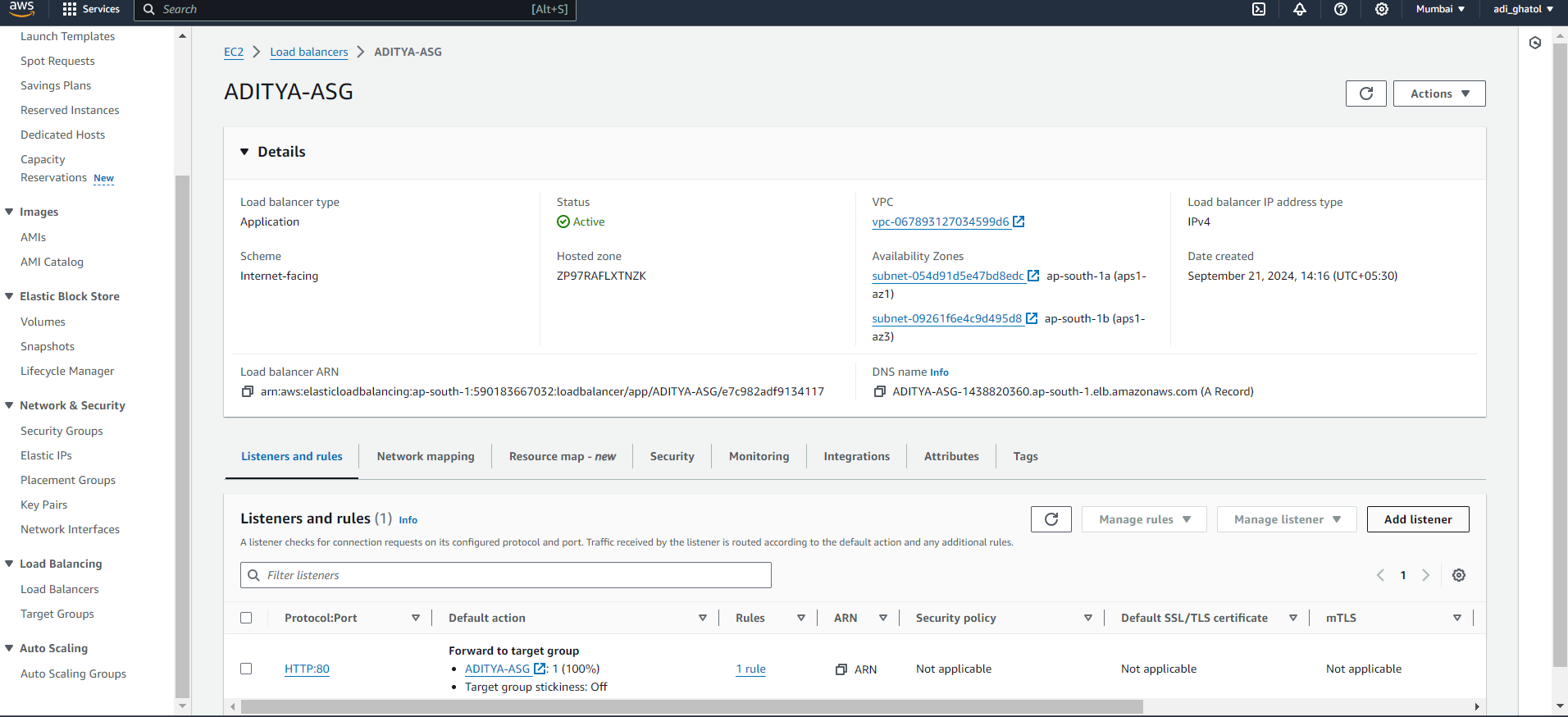


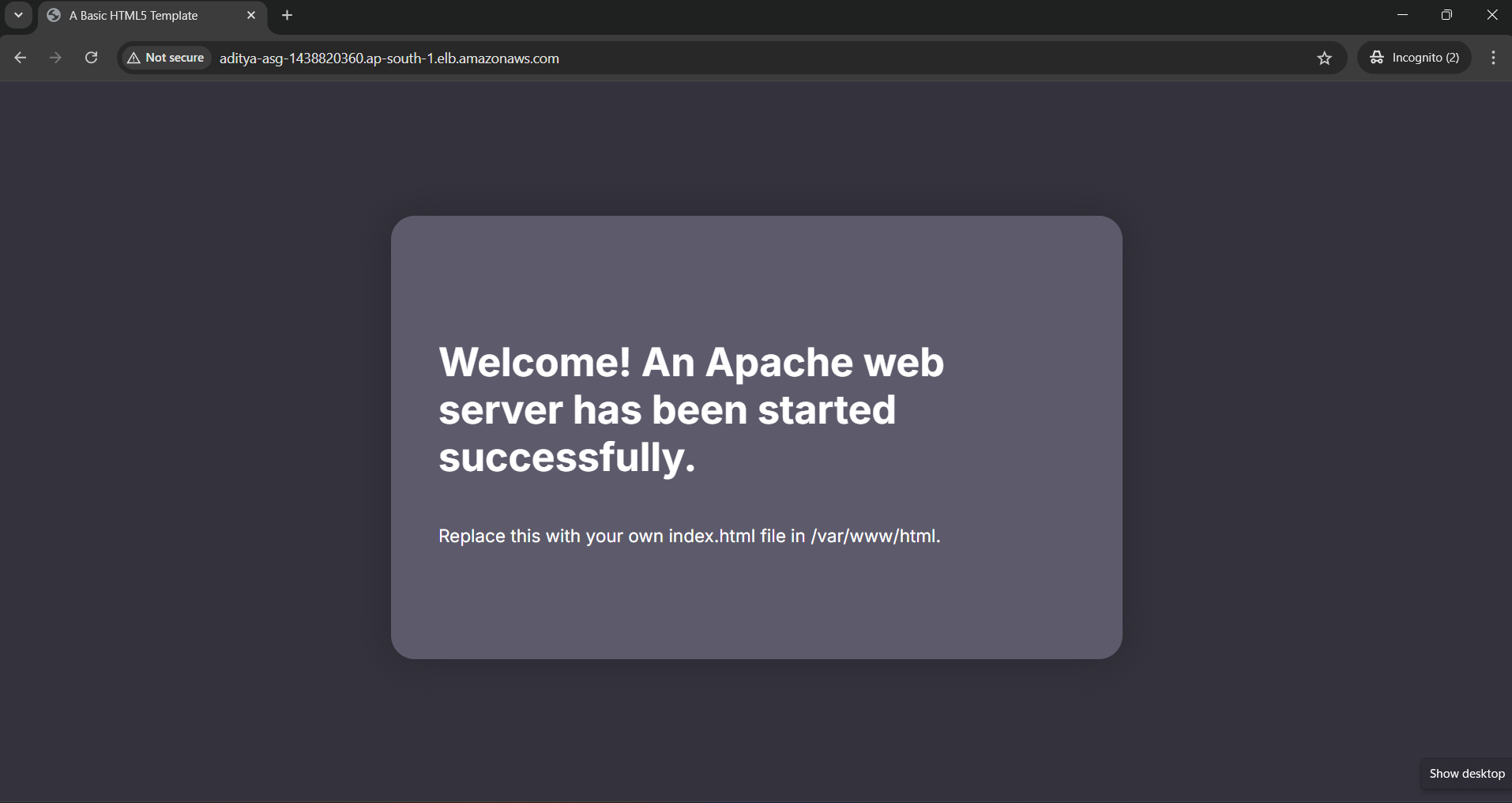


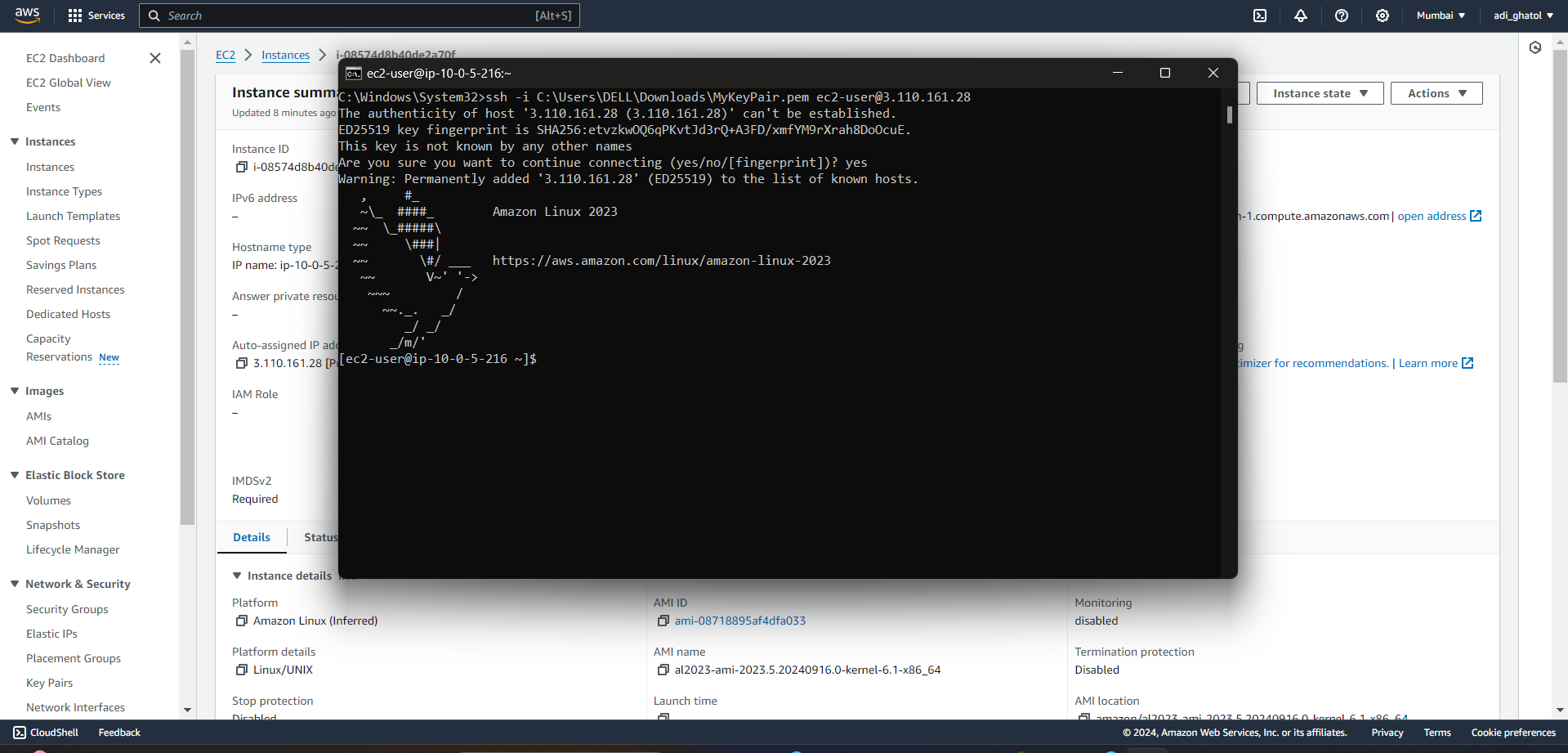












Success!

We’ve successfully built the architecture for the Web Tier for our Brainiac 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 Brainiac app 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.

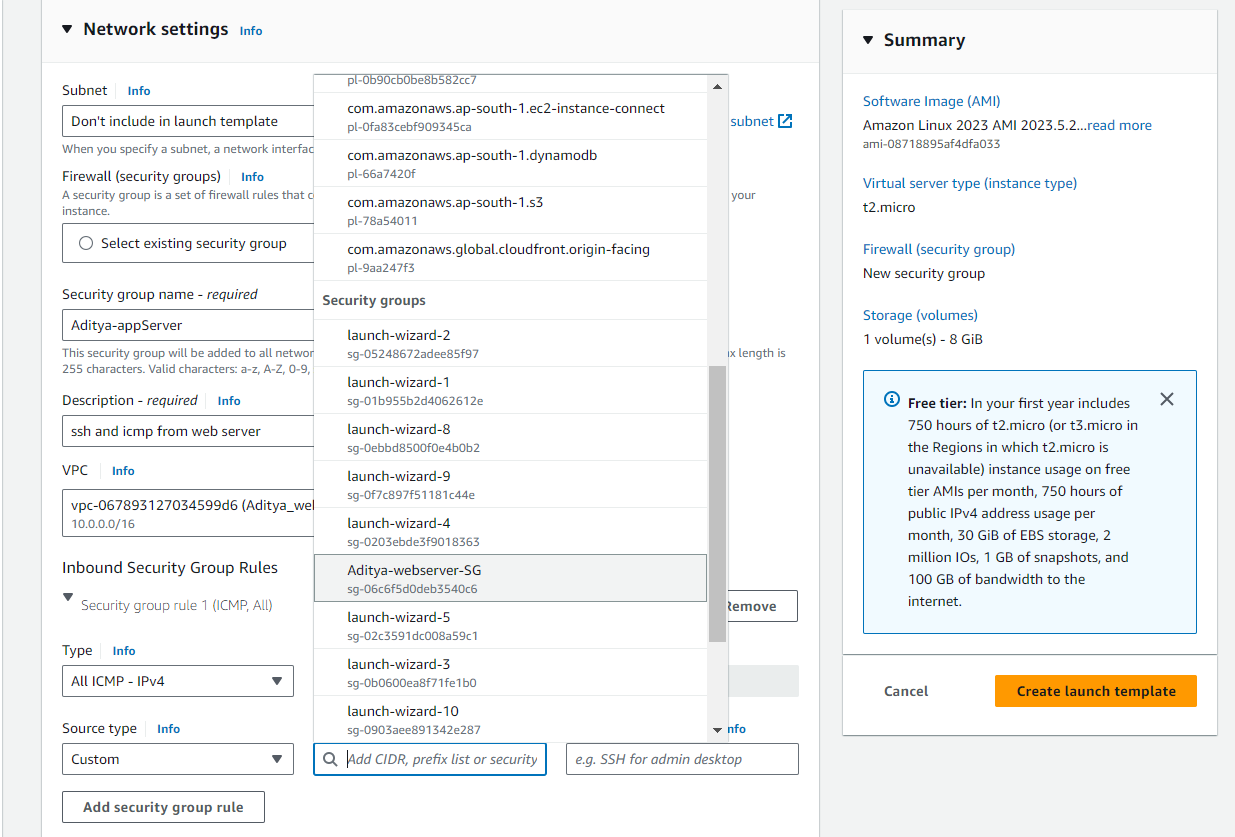
What we will build:

A launch template to define the type of EC2 instances.

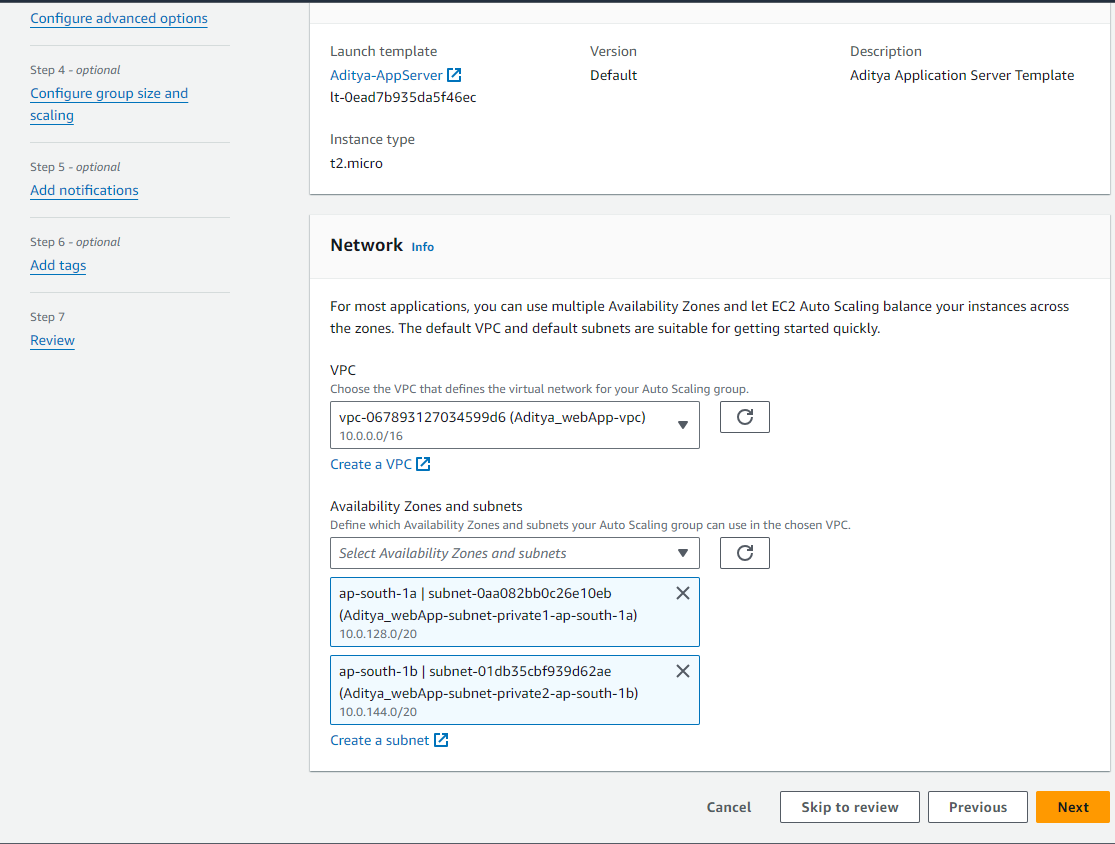
An Auto Scaling Group (ASG) to dynamically provision EC2 instances.

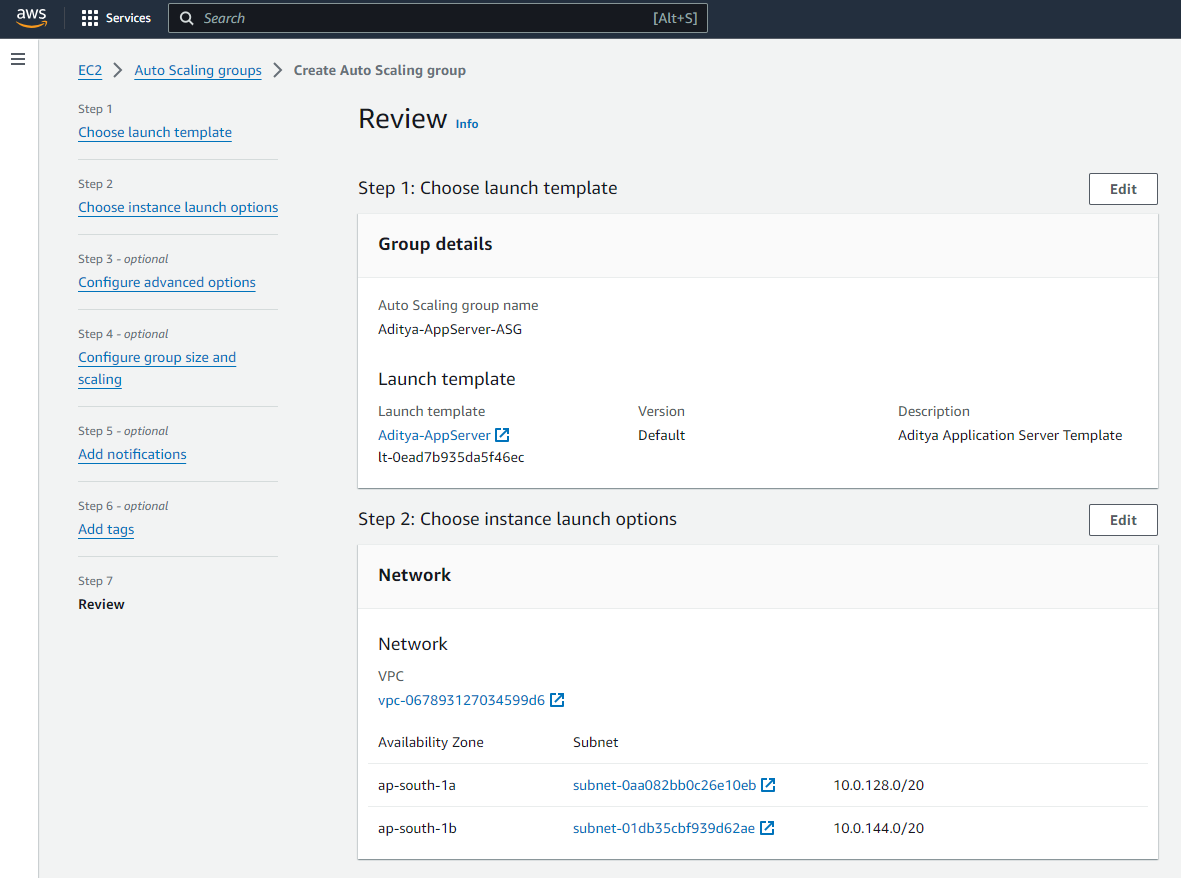
An Application Load Balancer (ALB) to route traffic from the Web tier.

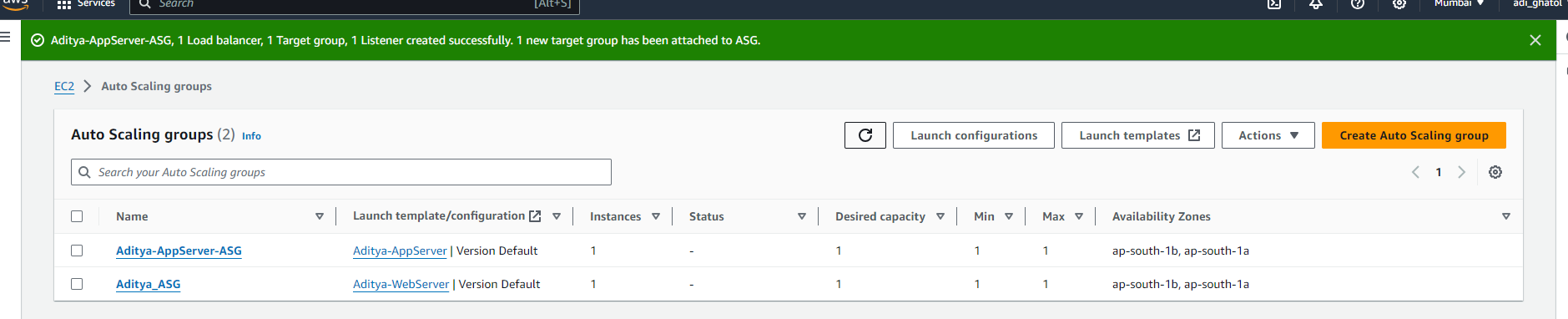
A Bastion host to securely connect to our application servers.

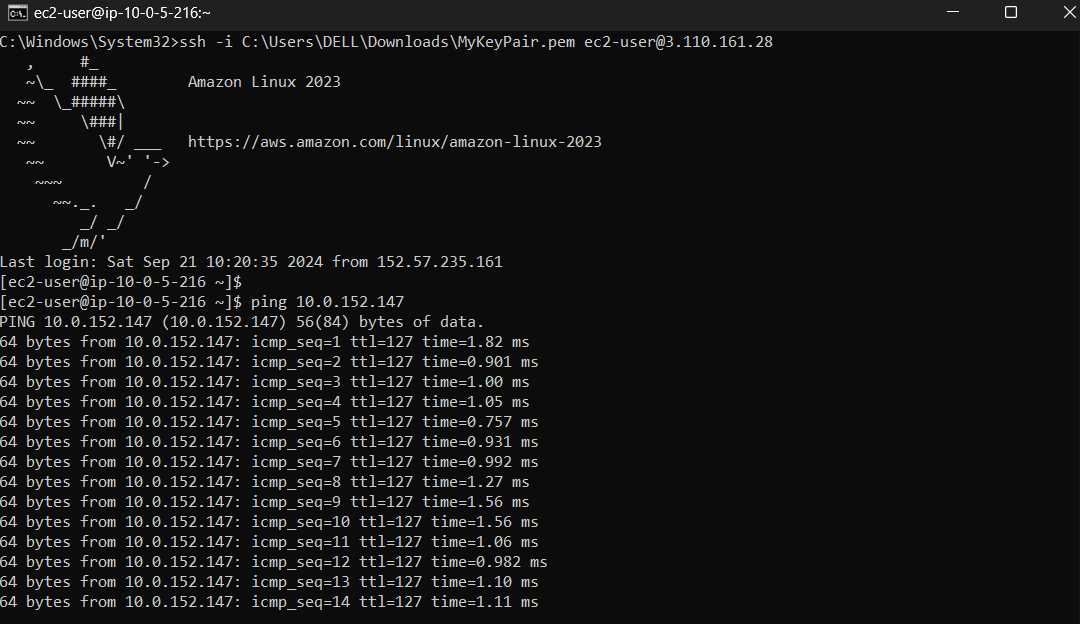


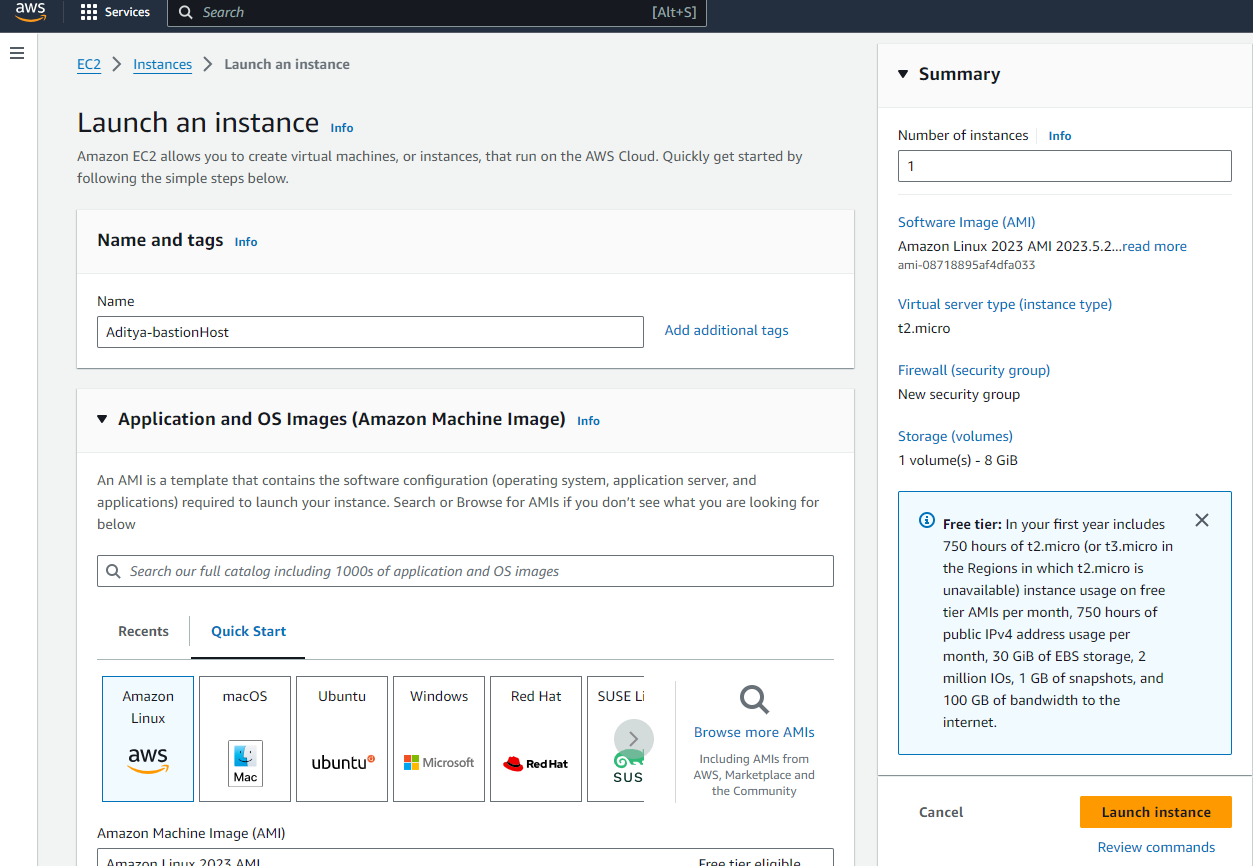


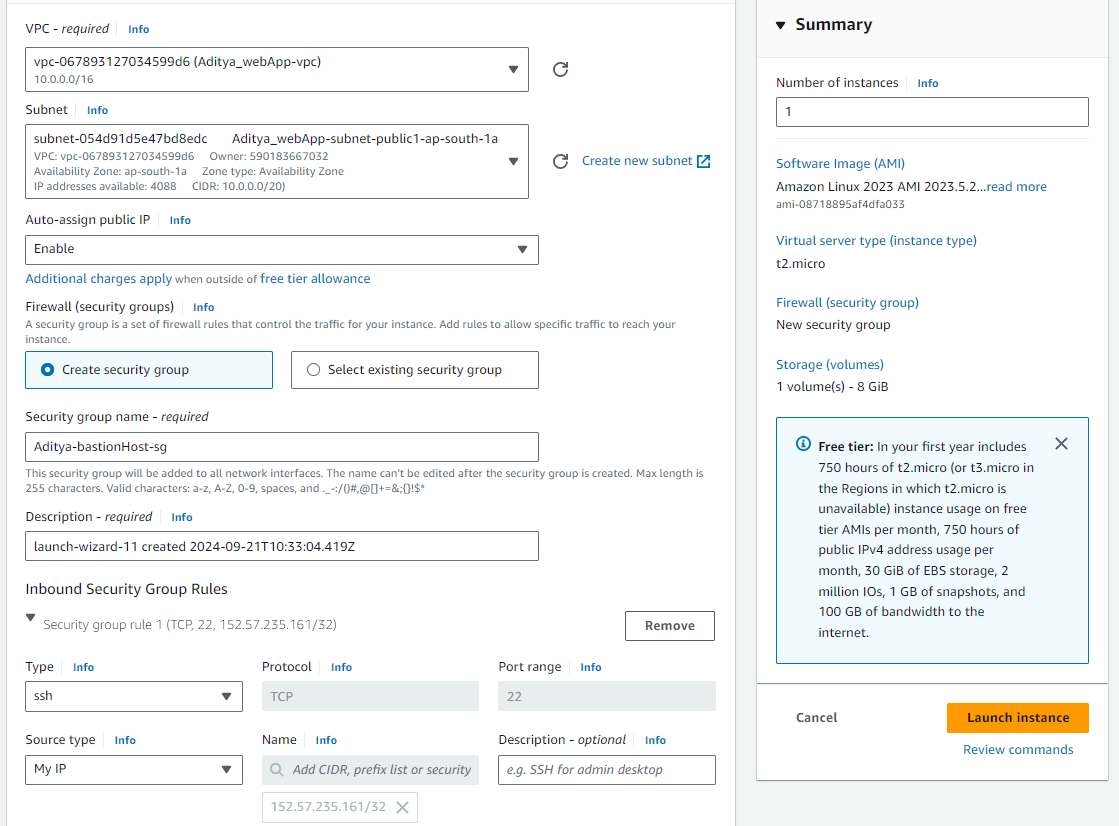


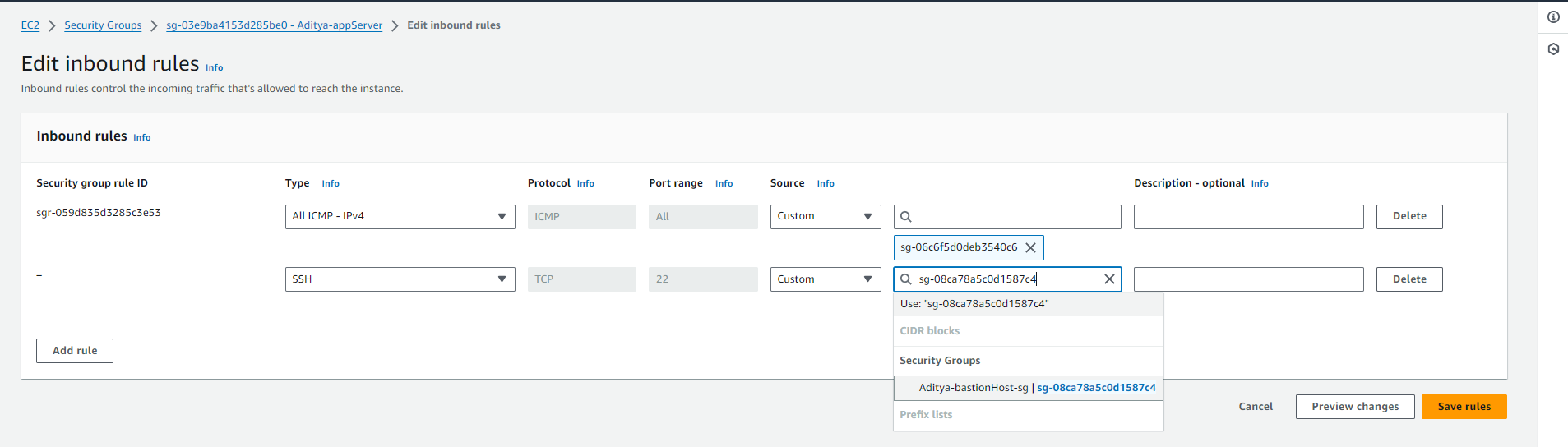


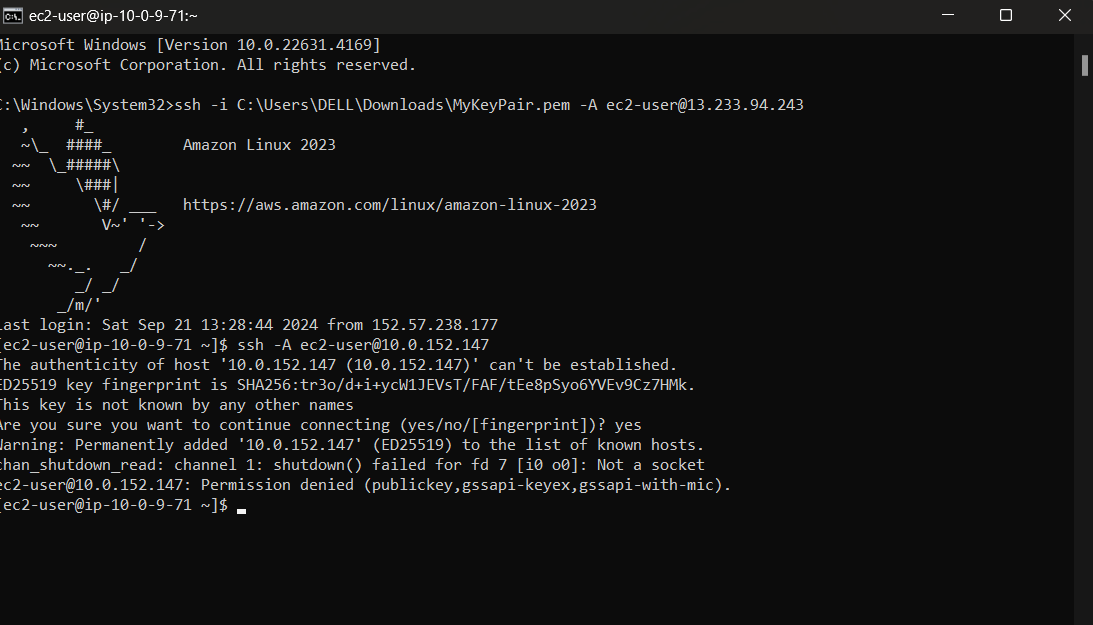












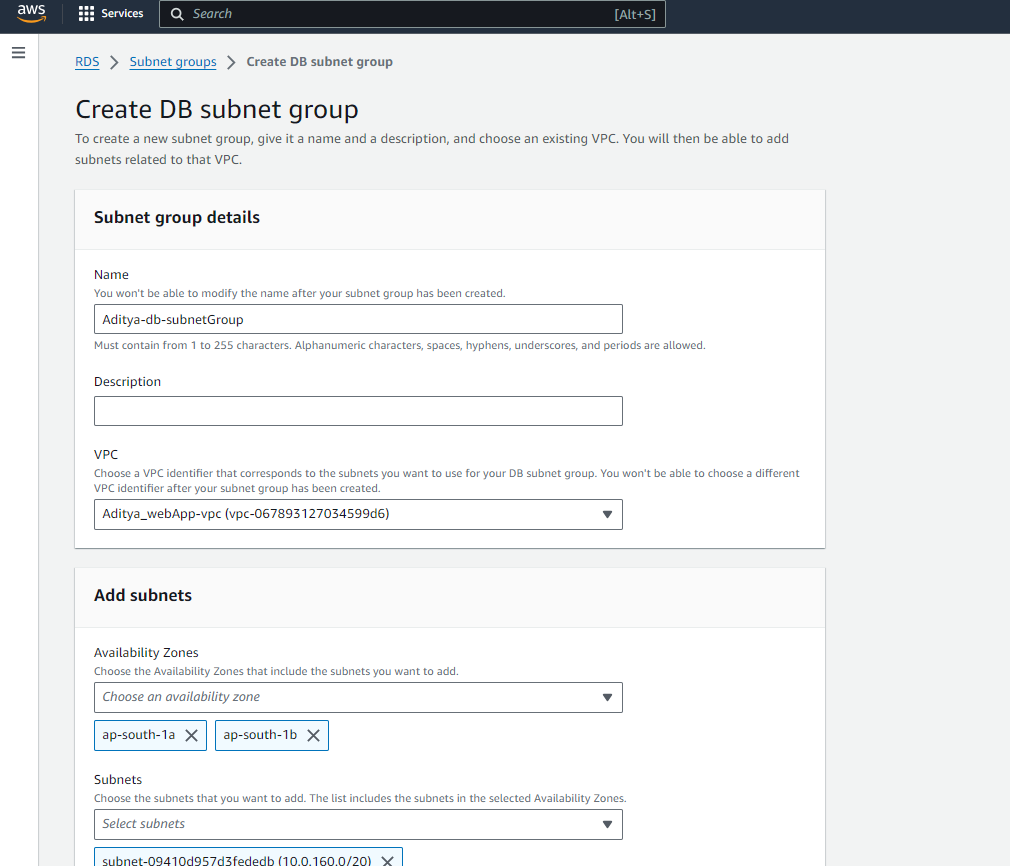
An Auto Scaling Group (ASG) that will dynamically provision EC2 instances.

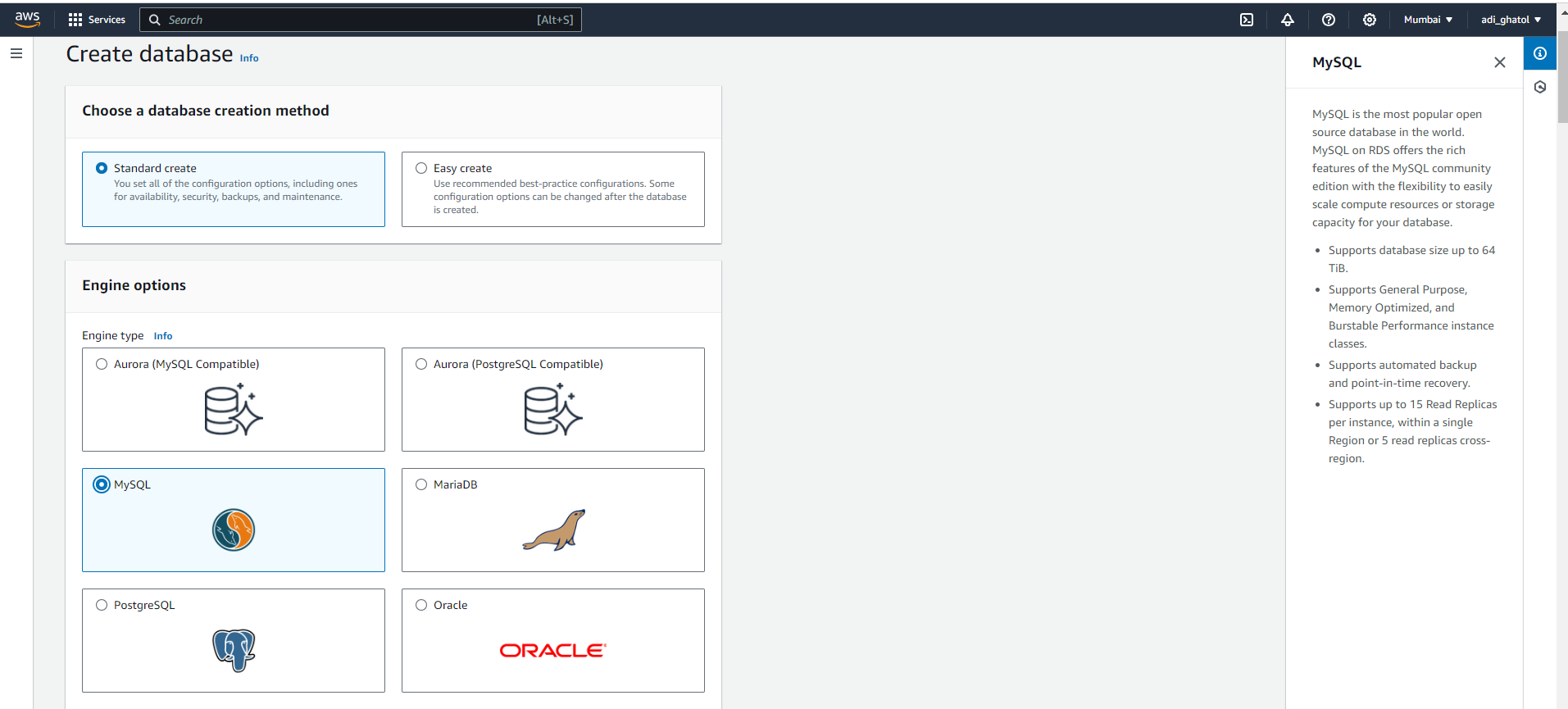
An Application Load Balancer (ALB) to help route incoming traffic to the proper targets.

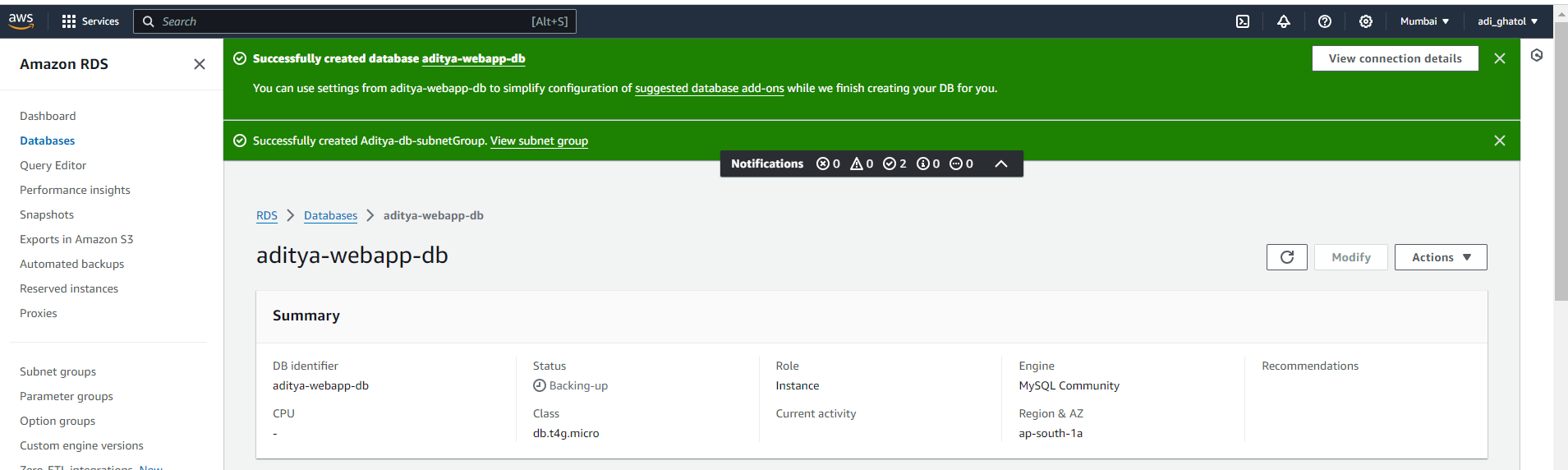
2. Set up a security group to allow SSH (port 22) and HTTP (port 80).

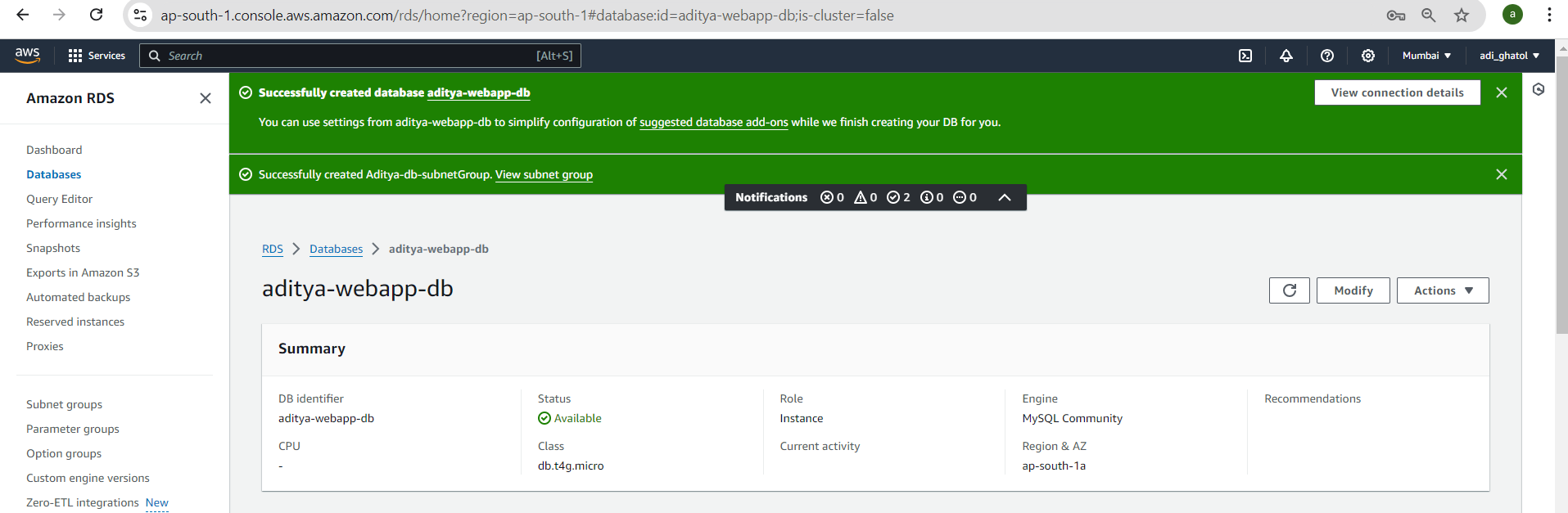
3. Install and start Apache Web Server on the instance:  
 sudo yum update -y  
 sudo yum install httpd -y  
 sudo service httpd start

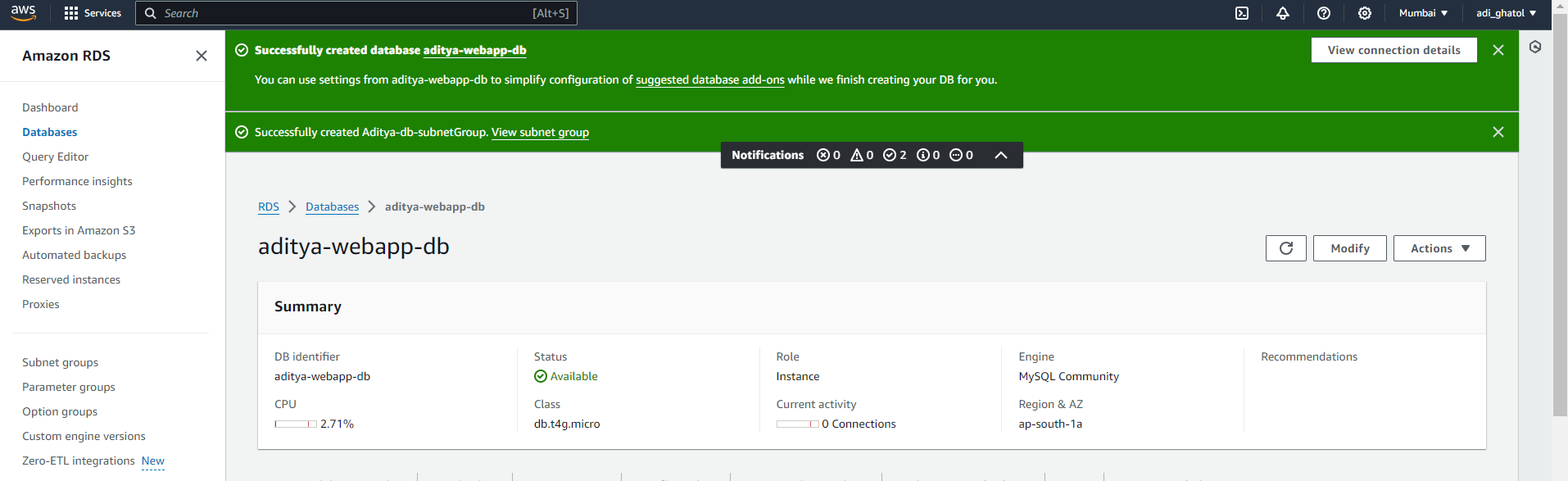
4. Access the instance via SSH using the public IP and verify the web server.

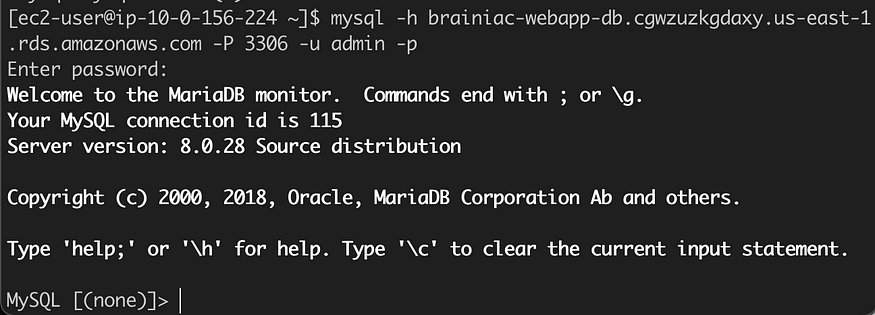












Great! We successfully connected to our database from our application server!

# Success!

Man, that was quite the journey! I know it wasn’t easy, but we took it step-by-step and pulled through. We’ve successfully created a highly available, 3-tier application architecture that’s ready for our Brainiac App!

2. Configure the security group to allow EC2 access to the database.  
3. Connect the web application to the RDS instance.

# 7. Conclusion

The AWS 3-tier web application project demonstrates the efficient use of cloud infrastructure for creating scalable, secure, and fault-tolerant applications. This architecture can be expanded further by incorporating additional AWS services such as S3, CloudFront, and Lambda for greater performance and flexibility.