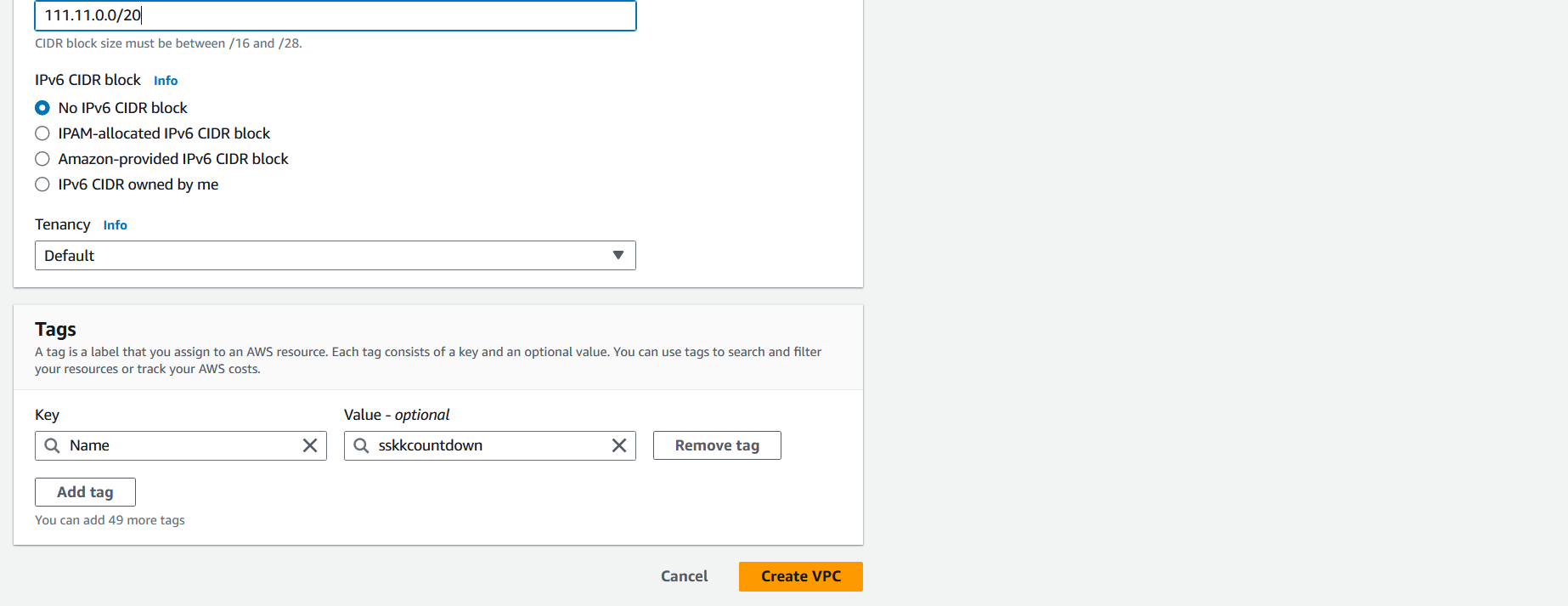
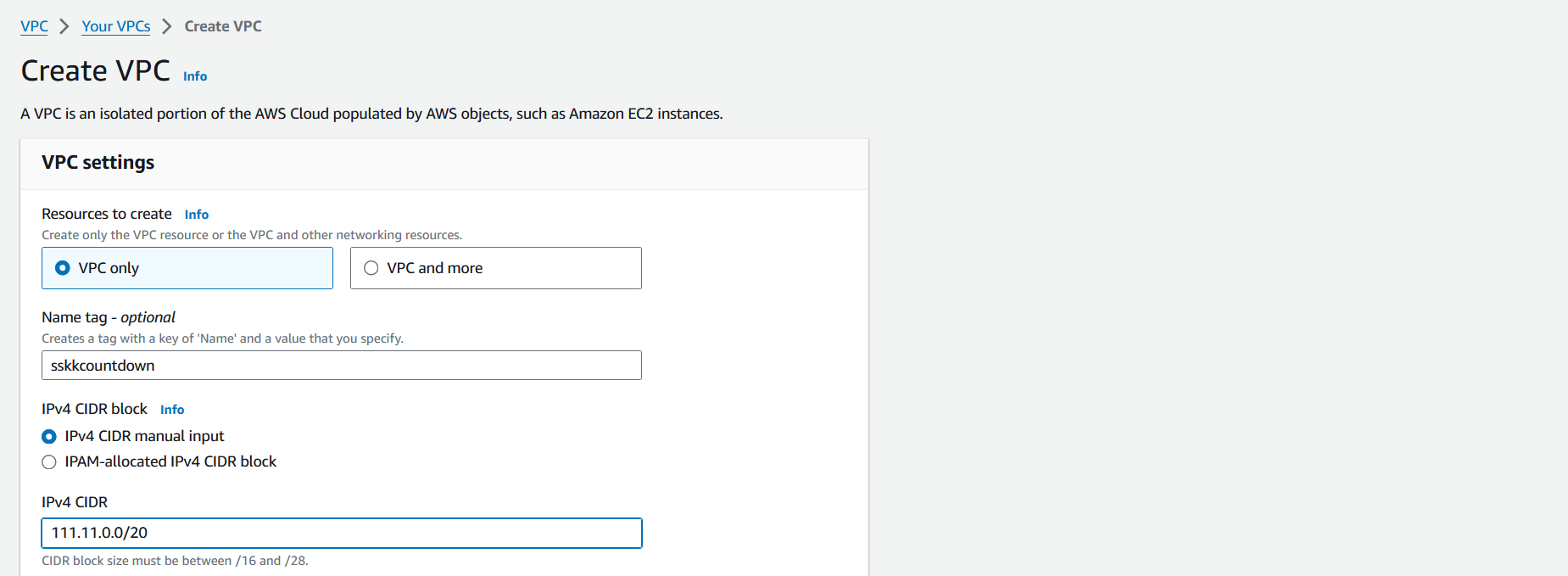
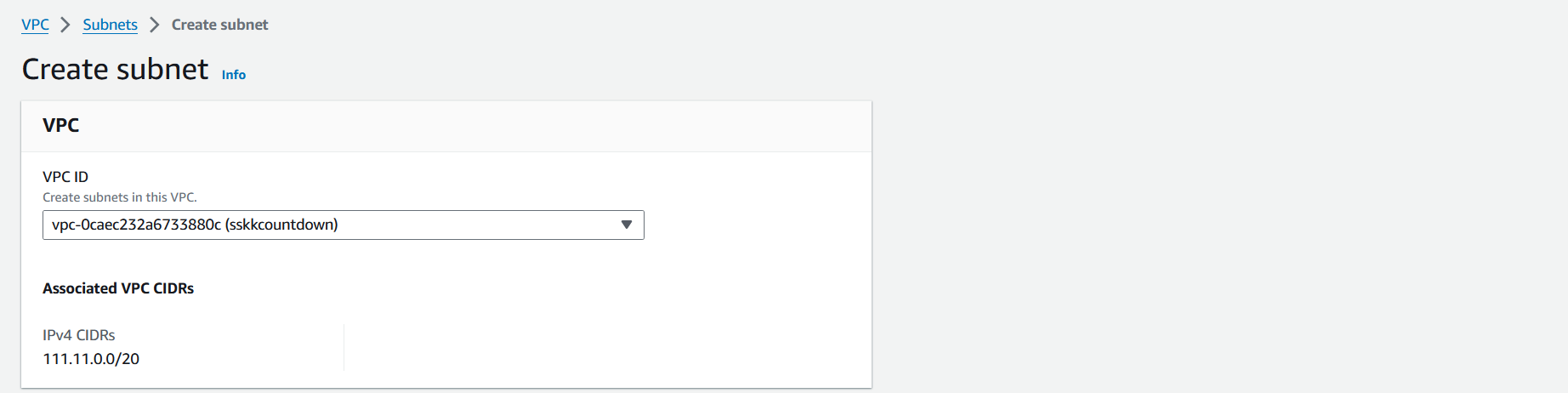
**Setting Up a Peering Connection - Capstone Project**

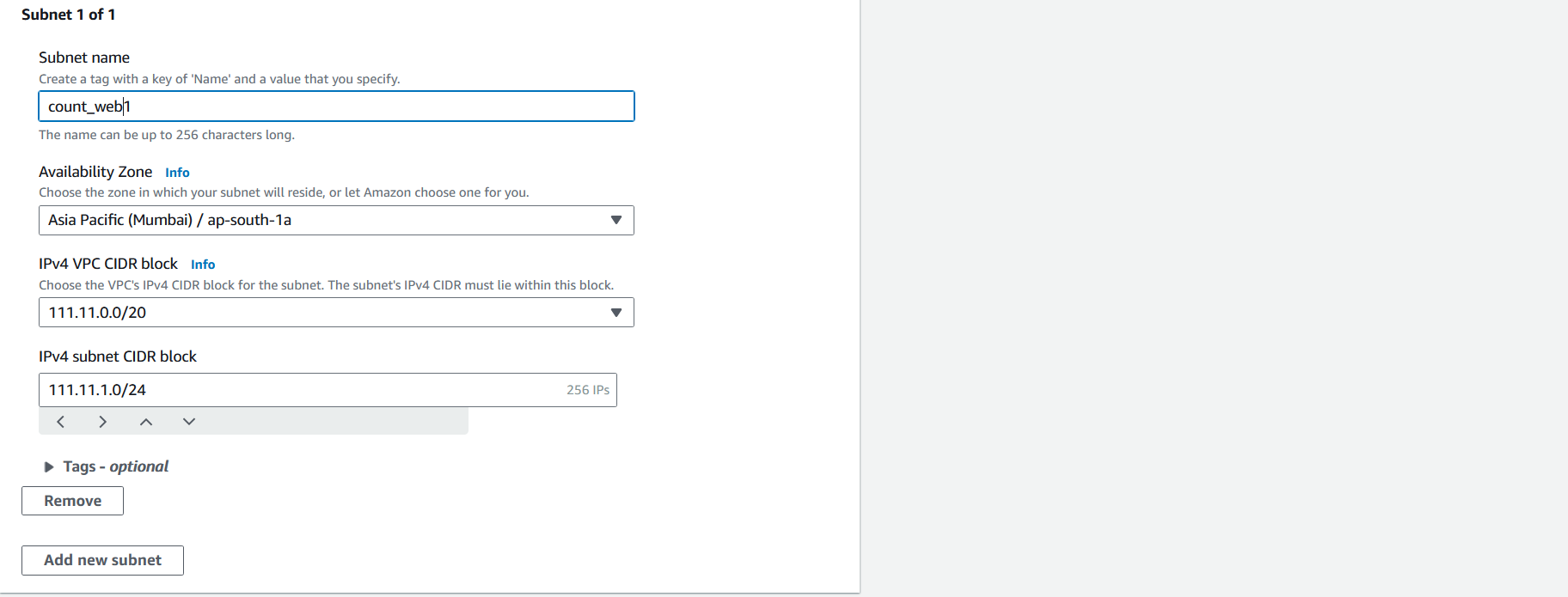
Create VPC and other resources in aws to launch a 3 tier site.

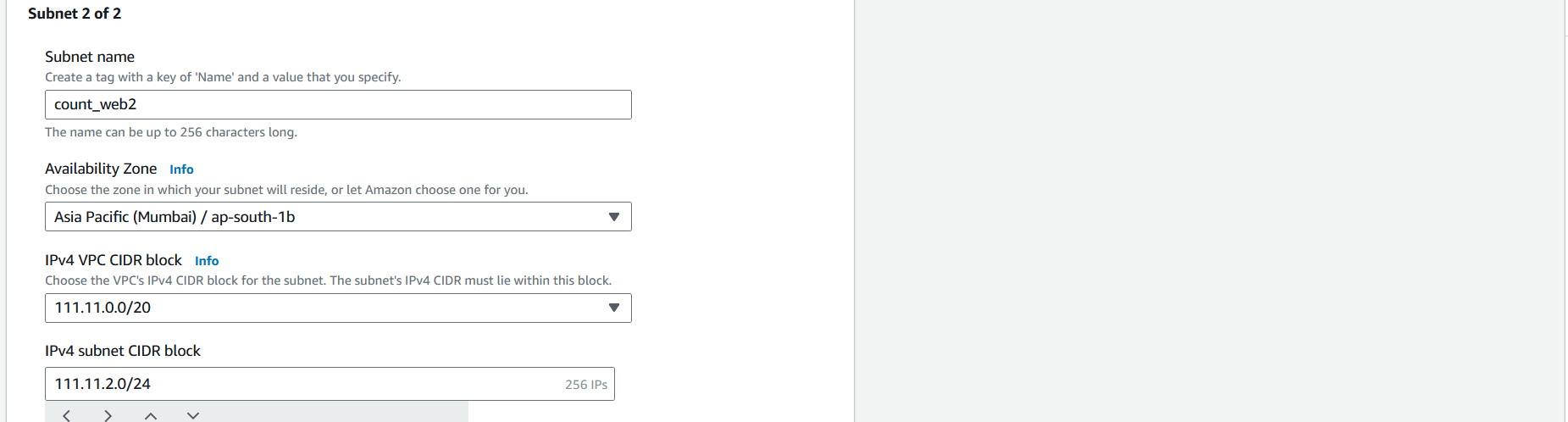
* Create VPC



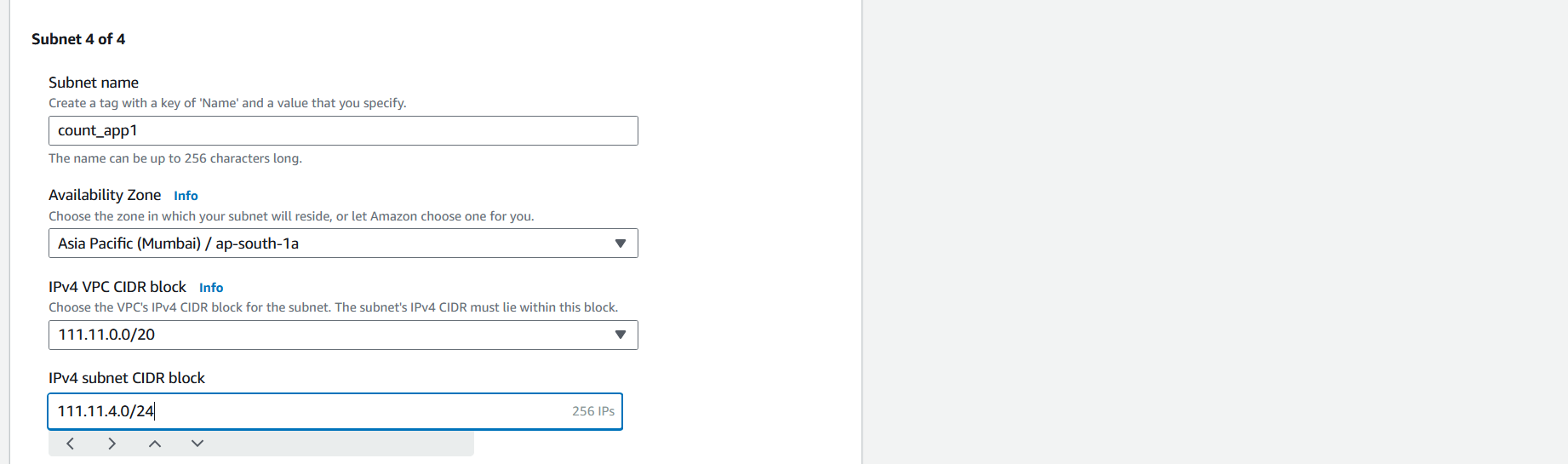
* Create Subnets

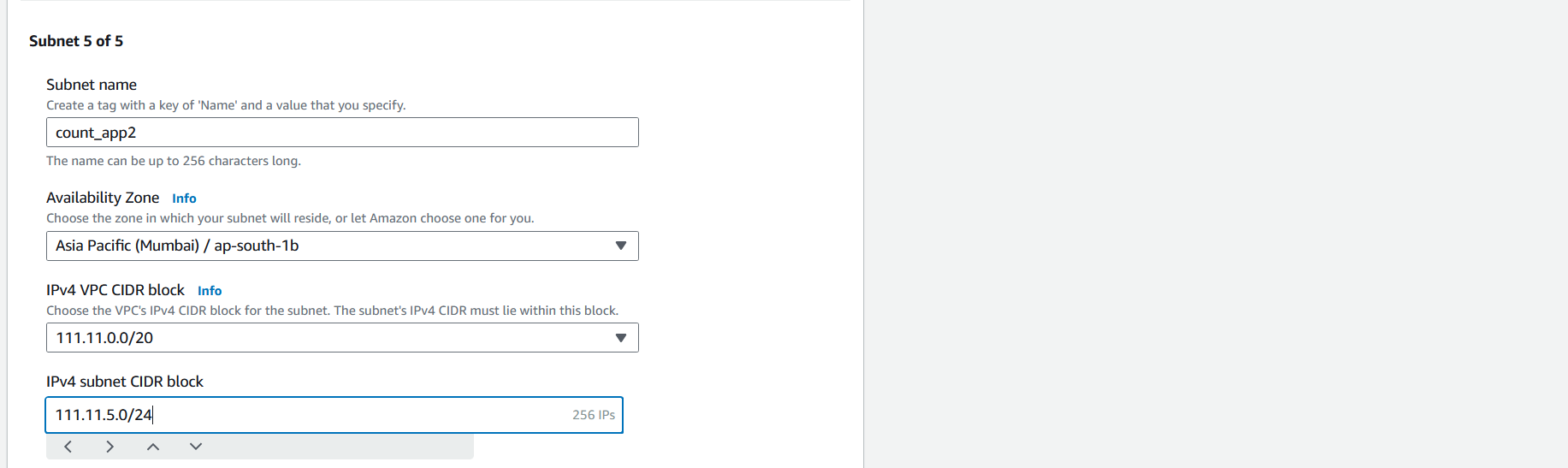


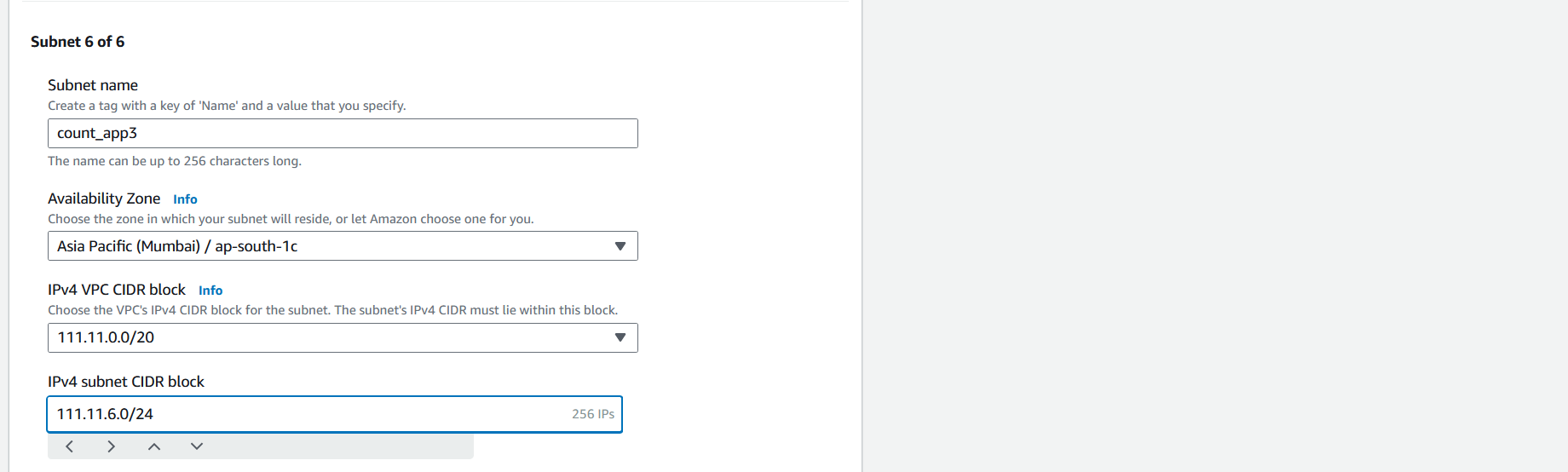


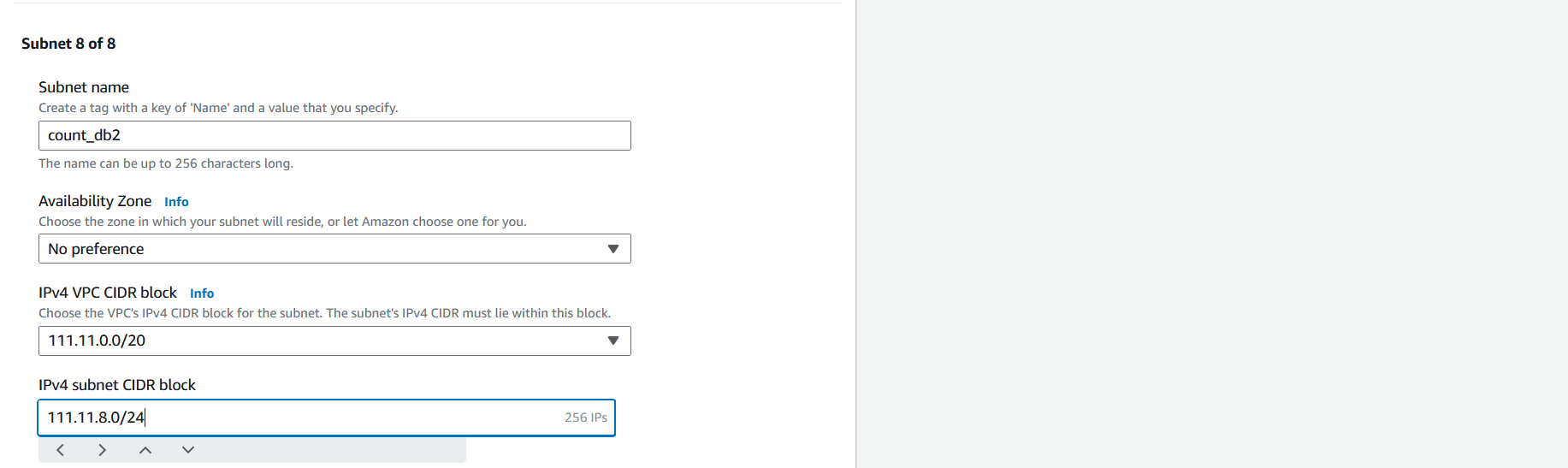
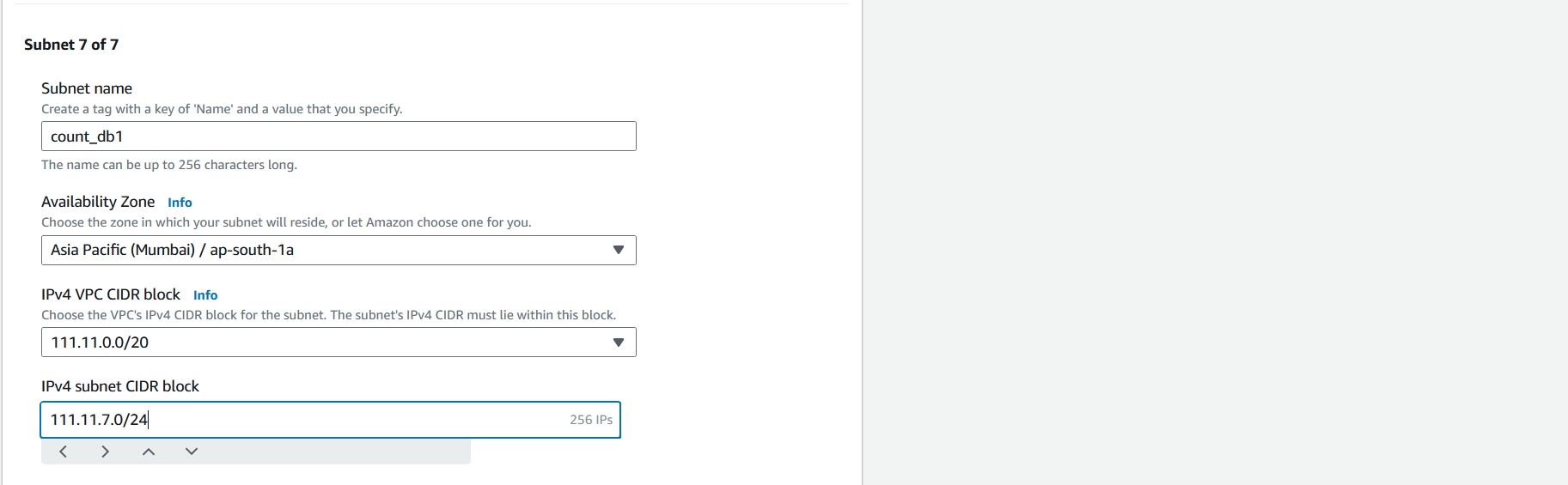


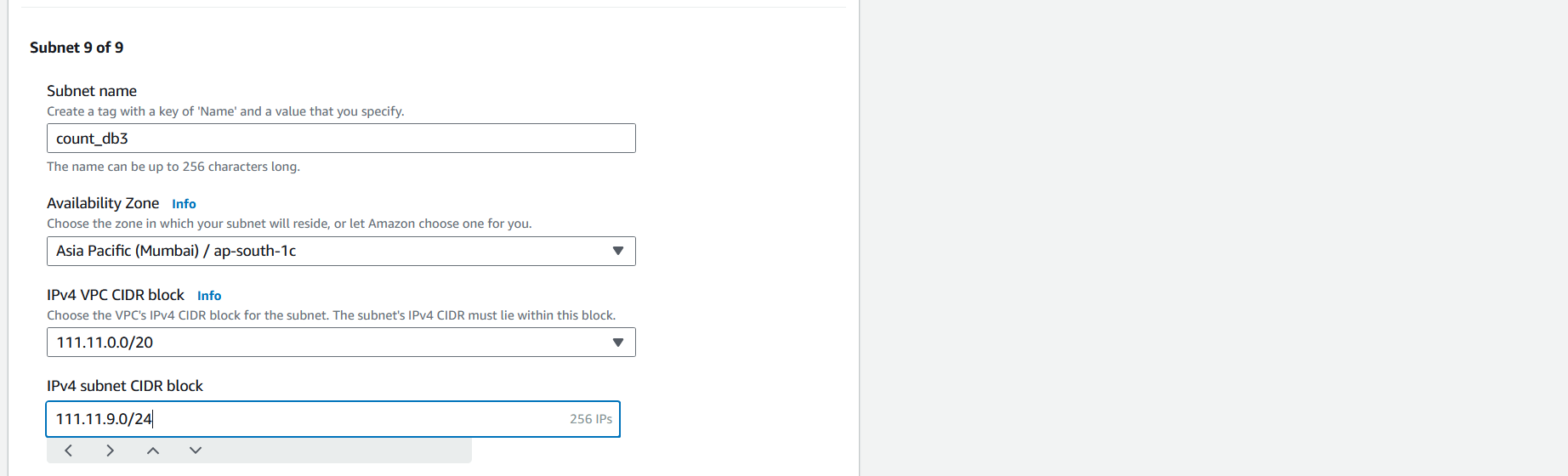


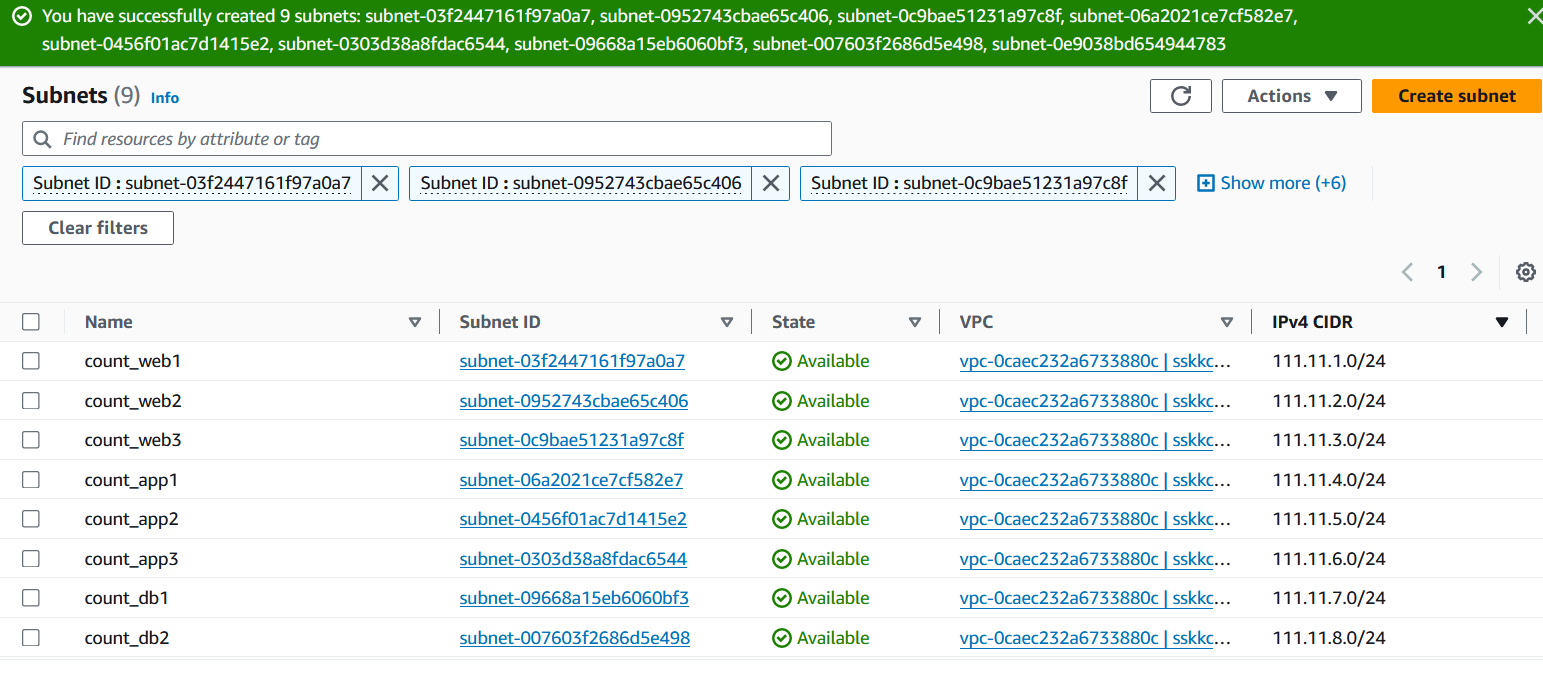




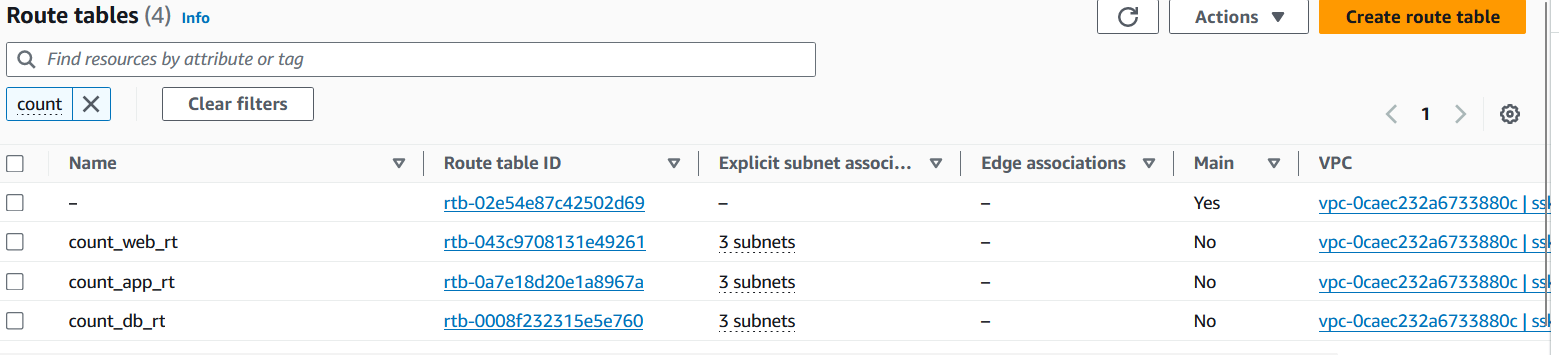




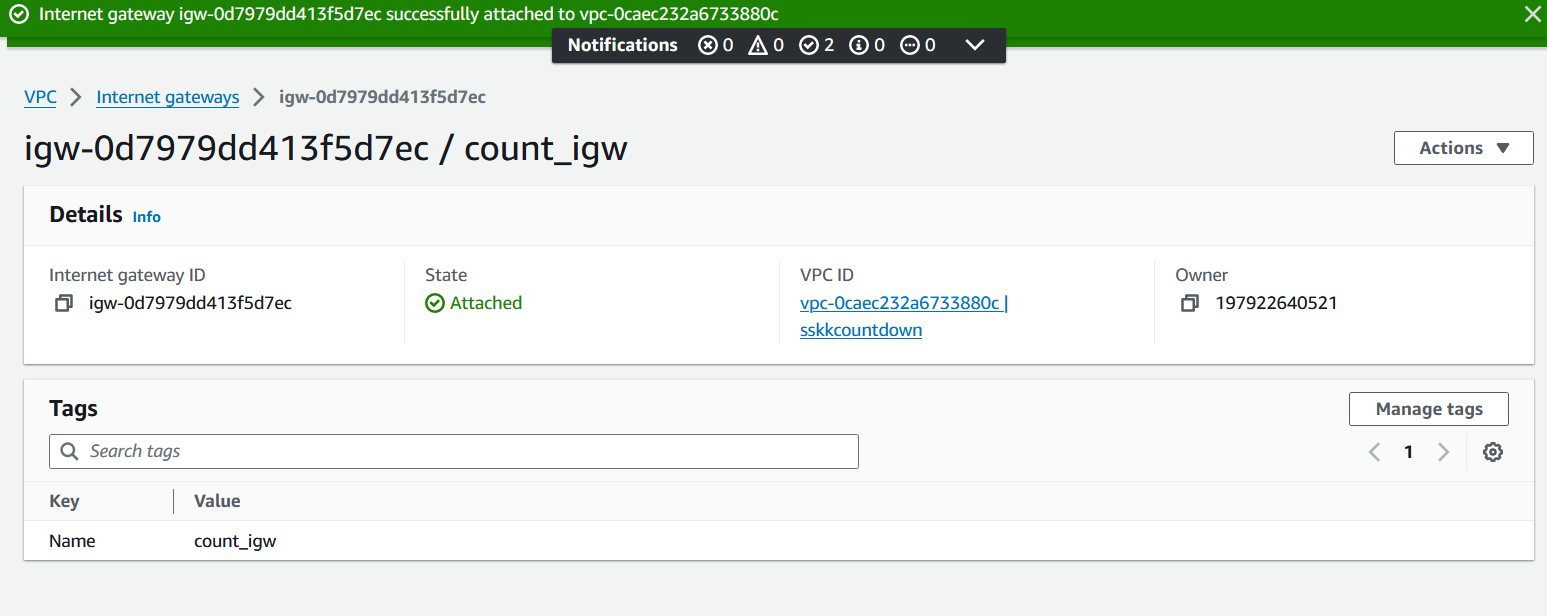


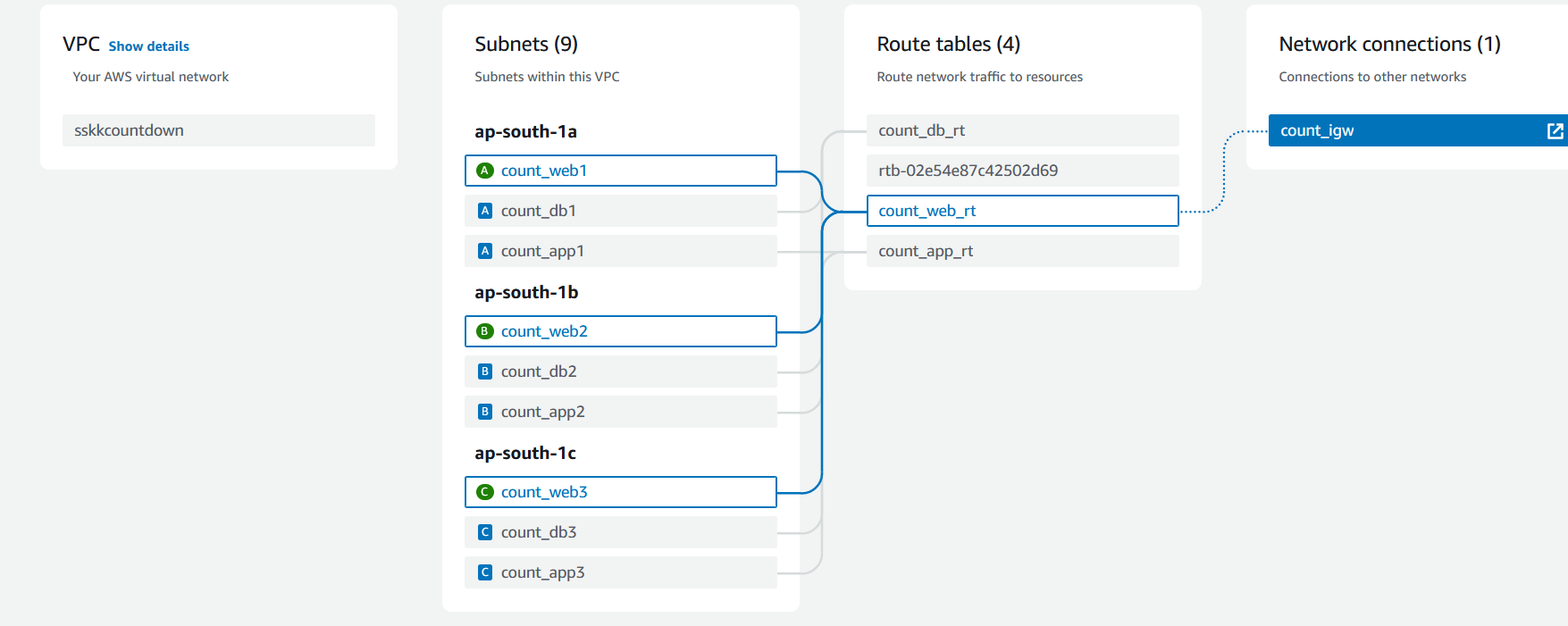


* Create Route tables and associate them with the subnets



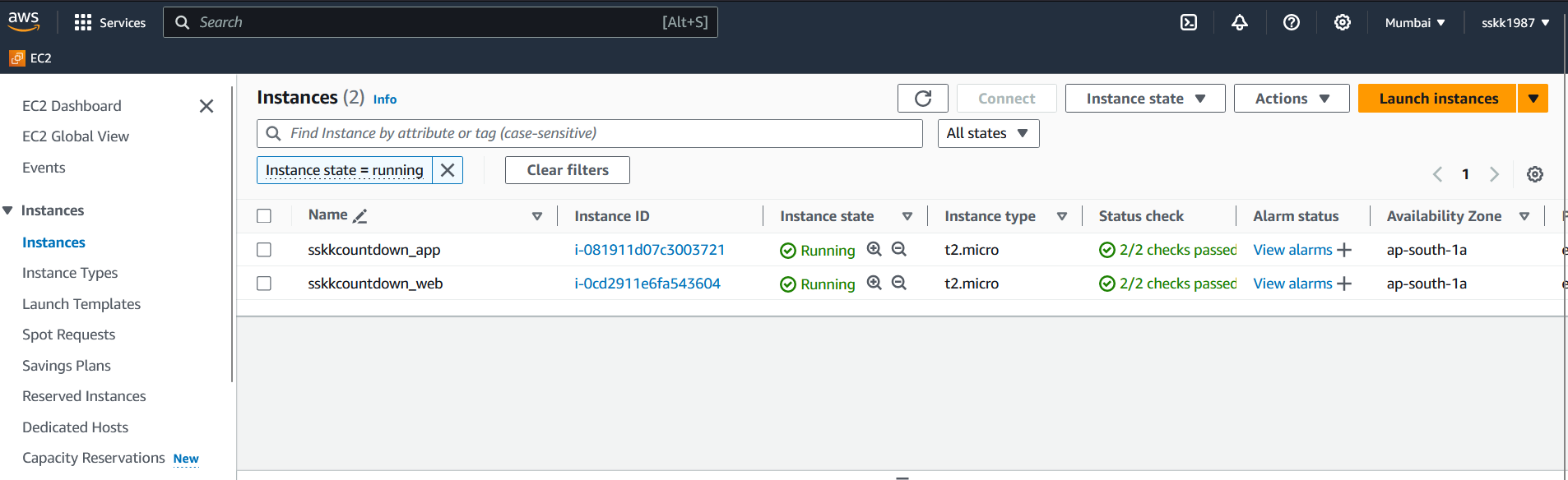
* Create an internet gateway and attach it to the VPC. Make sure to associate the route table with internet gateway



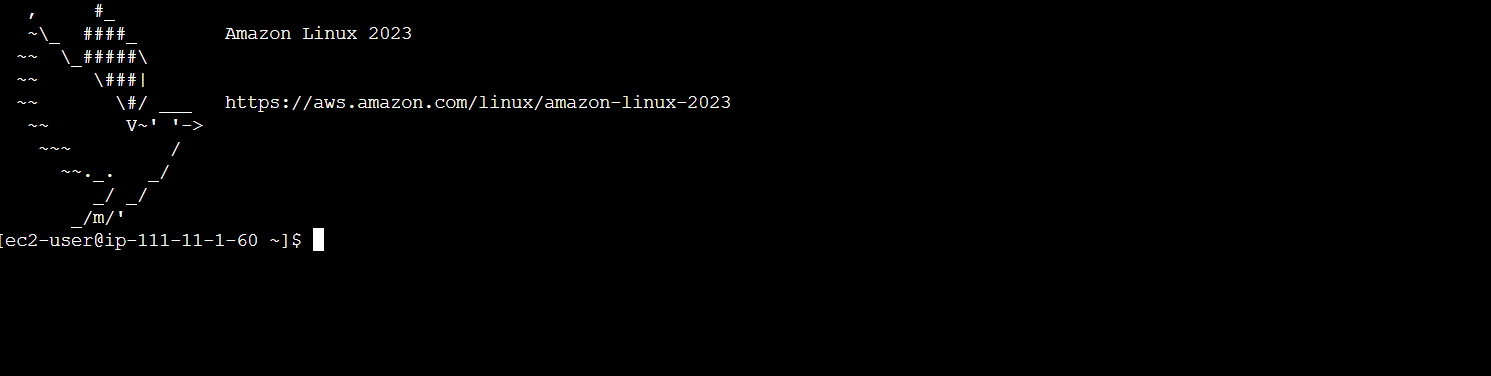


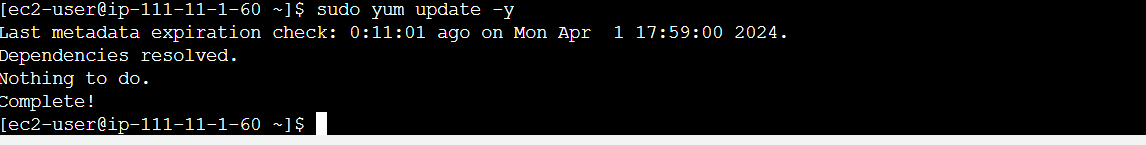
1. **Start with Web Tier**:

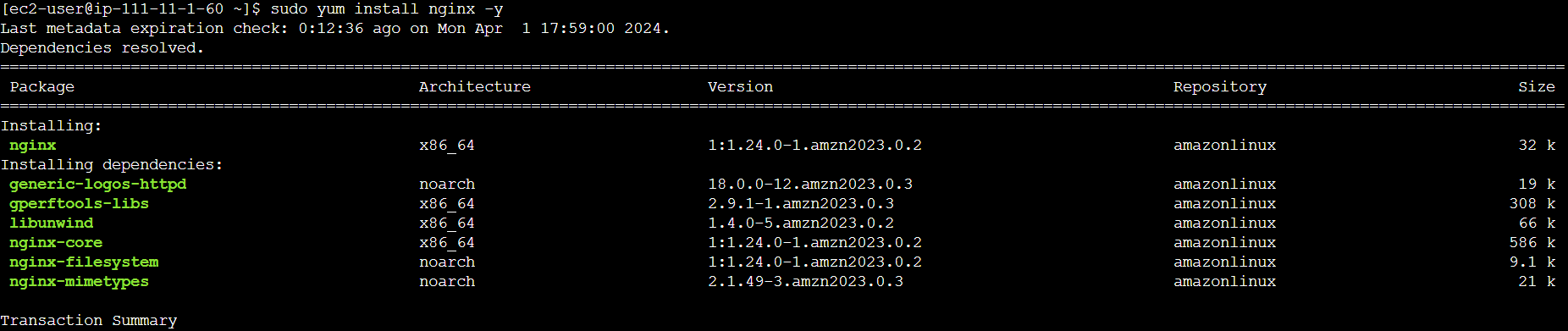
* Set up an EC2 instance to serve as the web tier.

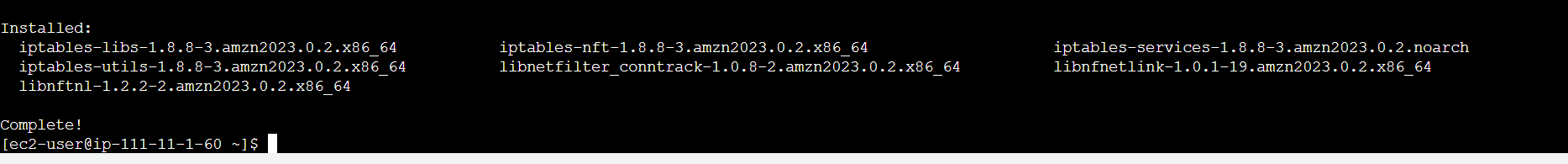
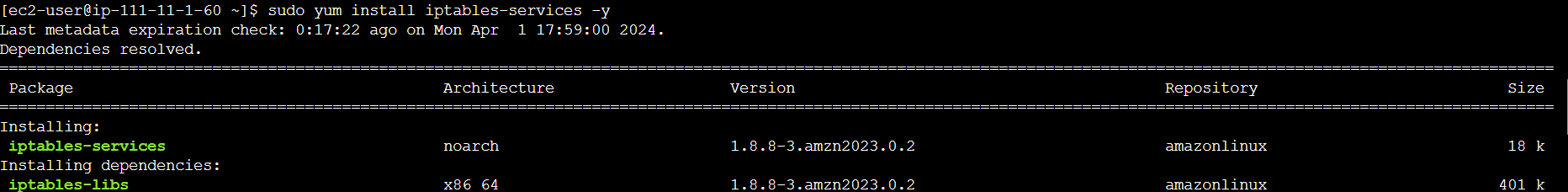
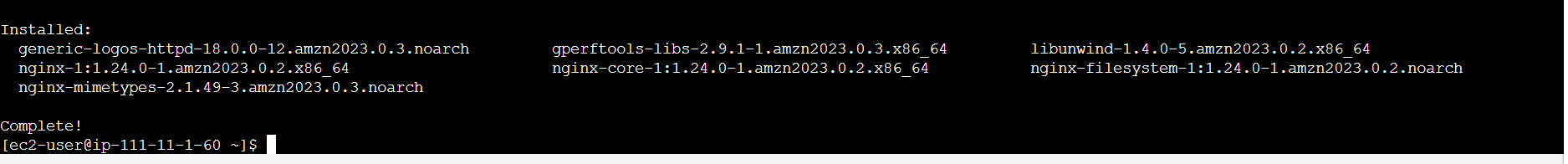


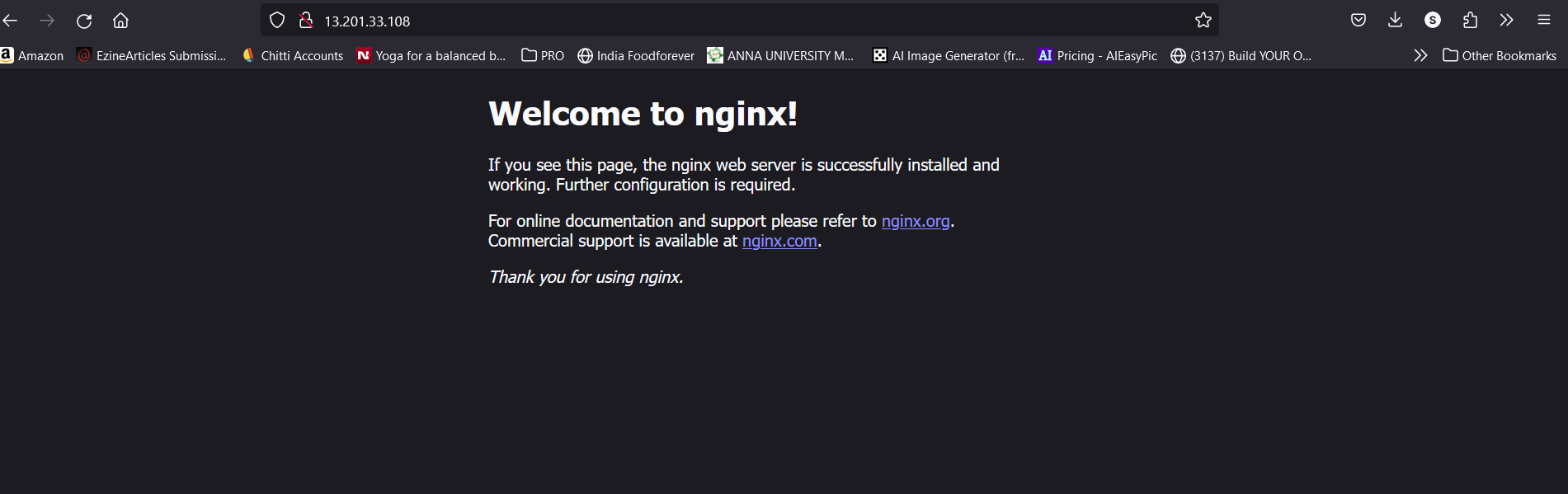
* Install and configure a web server (e.g., Nginx) to serve static files and forward requests to the app tier.





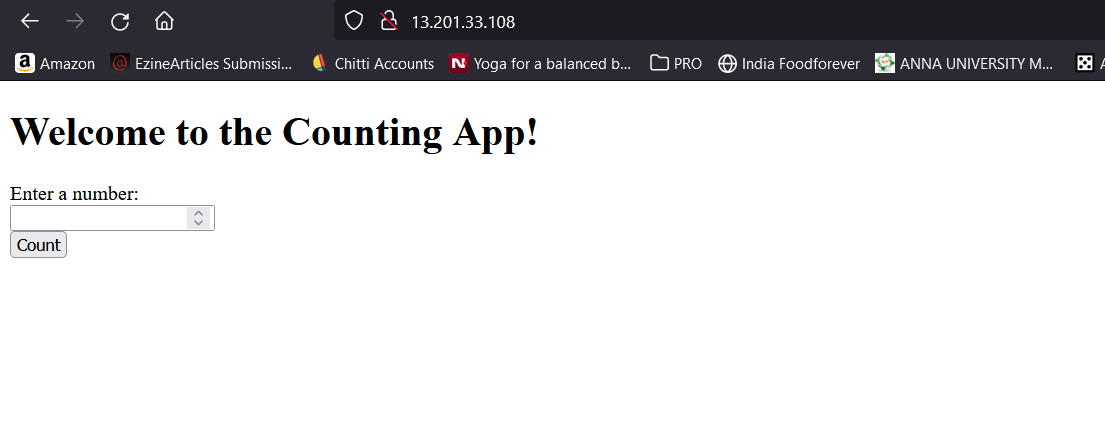






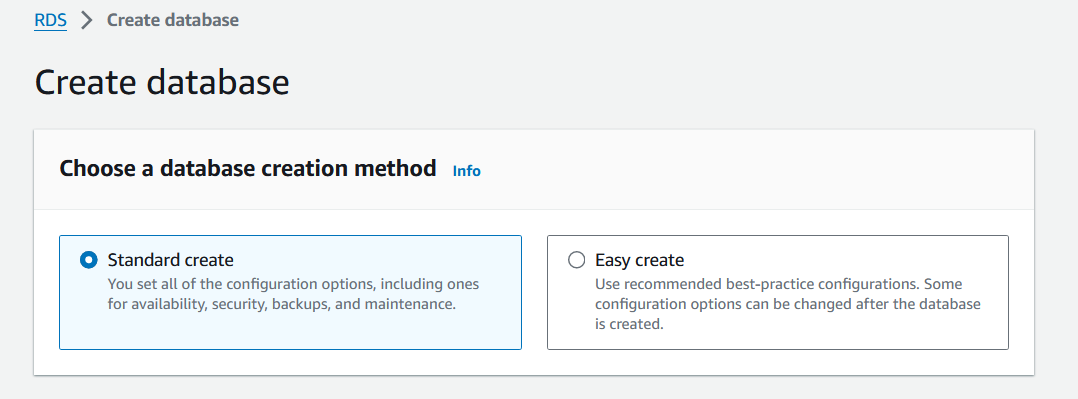
* Copy your static HTML files (index.html, count.html, etc.) to the web server's document root directory.

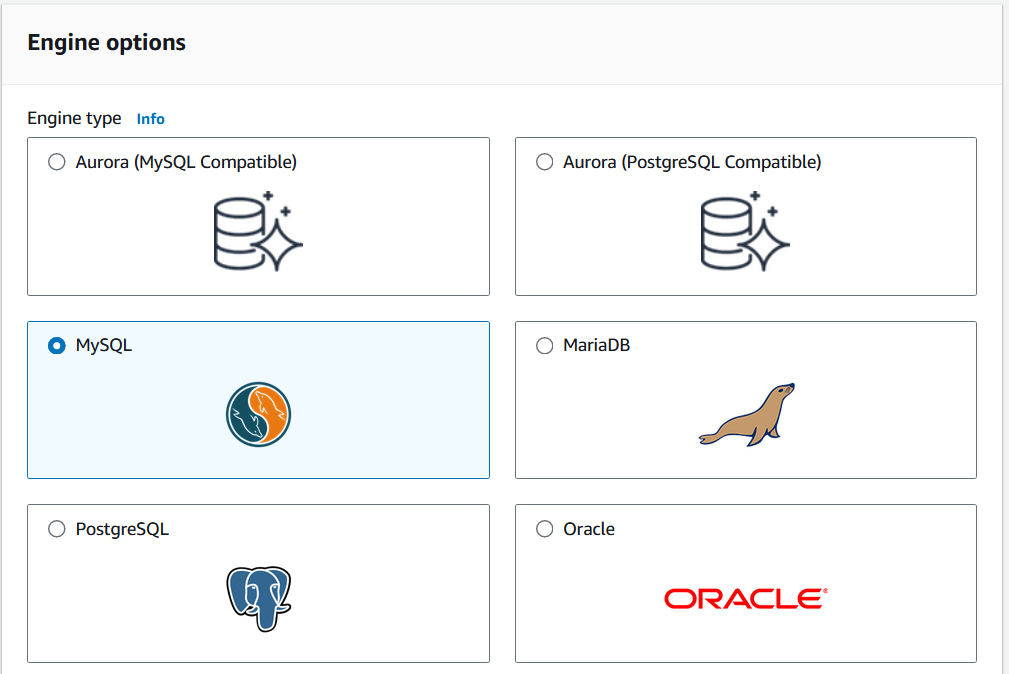


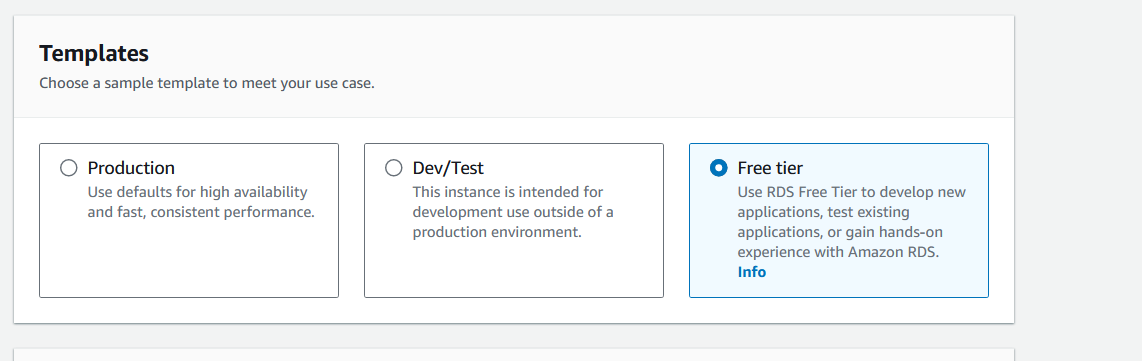


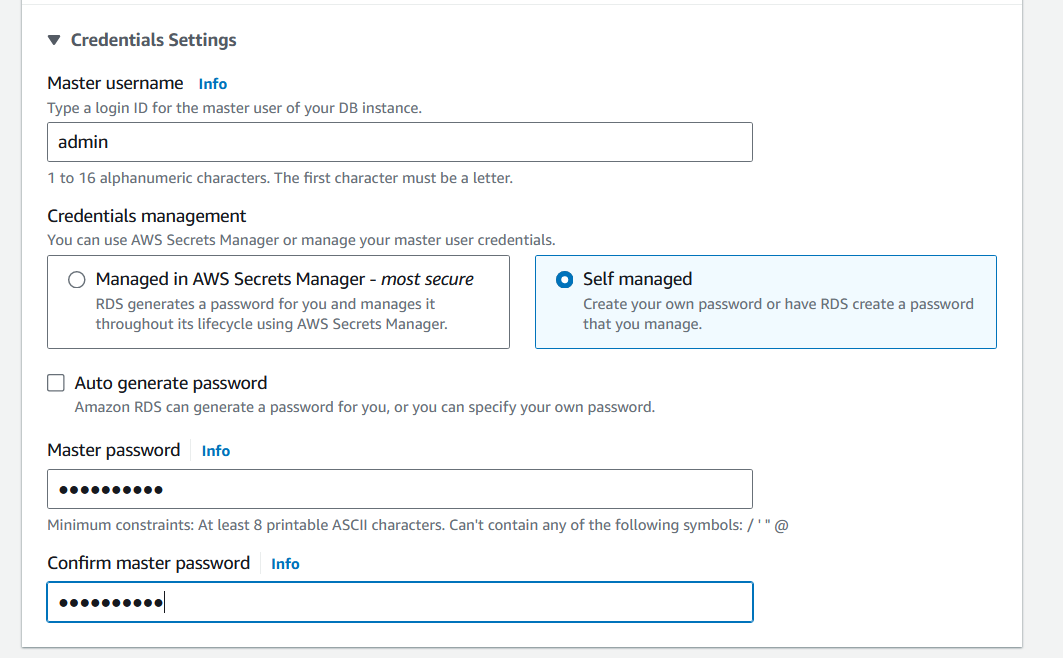
1. **Create the Database**:

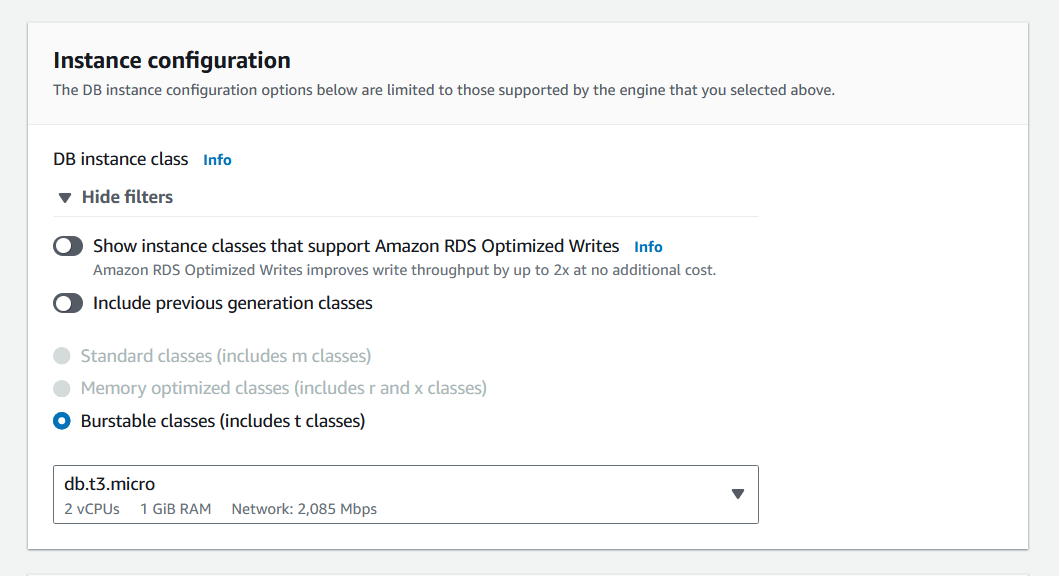
* Choose a suitable database service. In this case, MySQL on Amazon RDS.

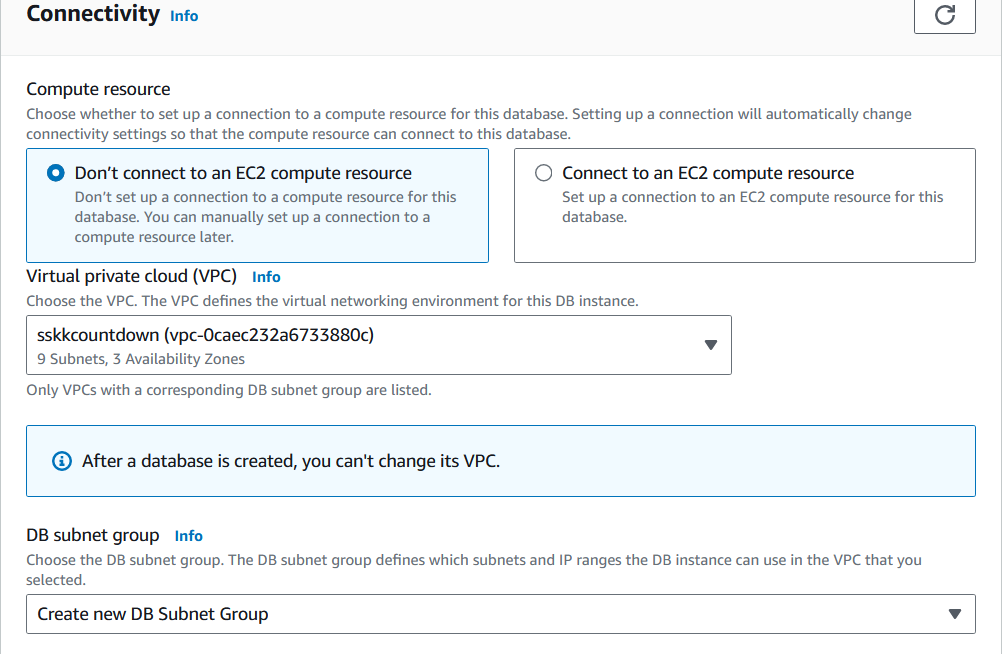


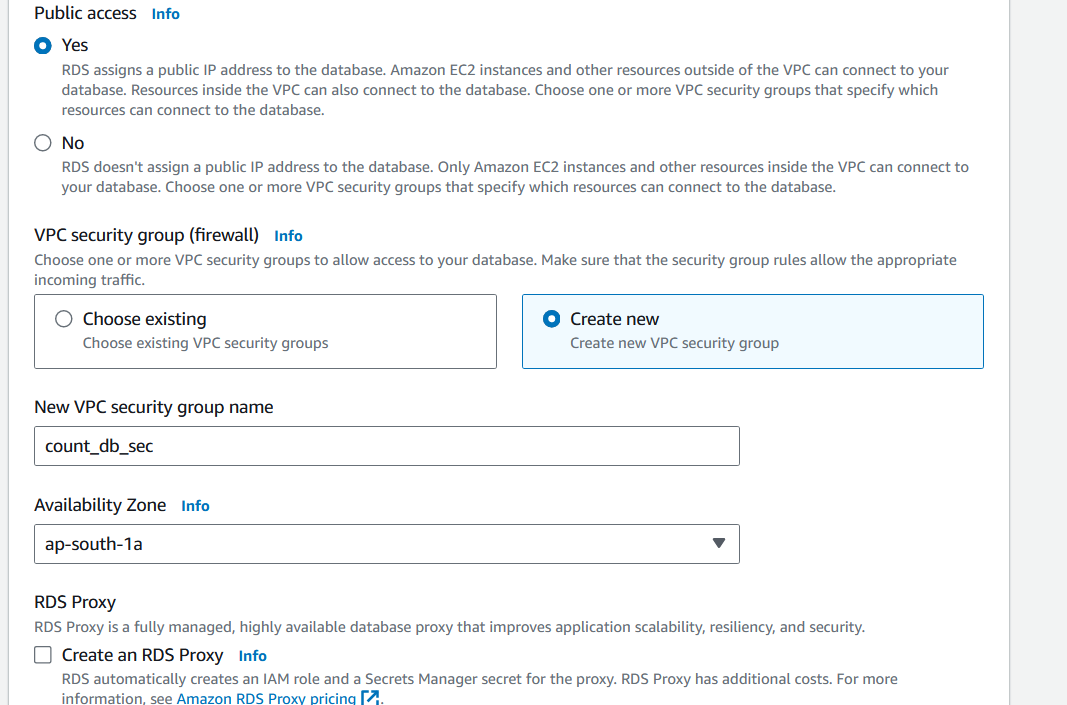




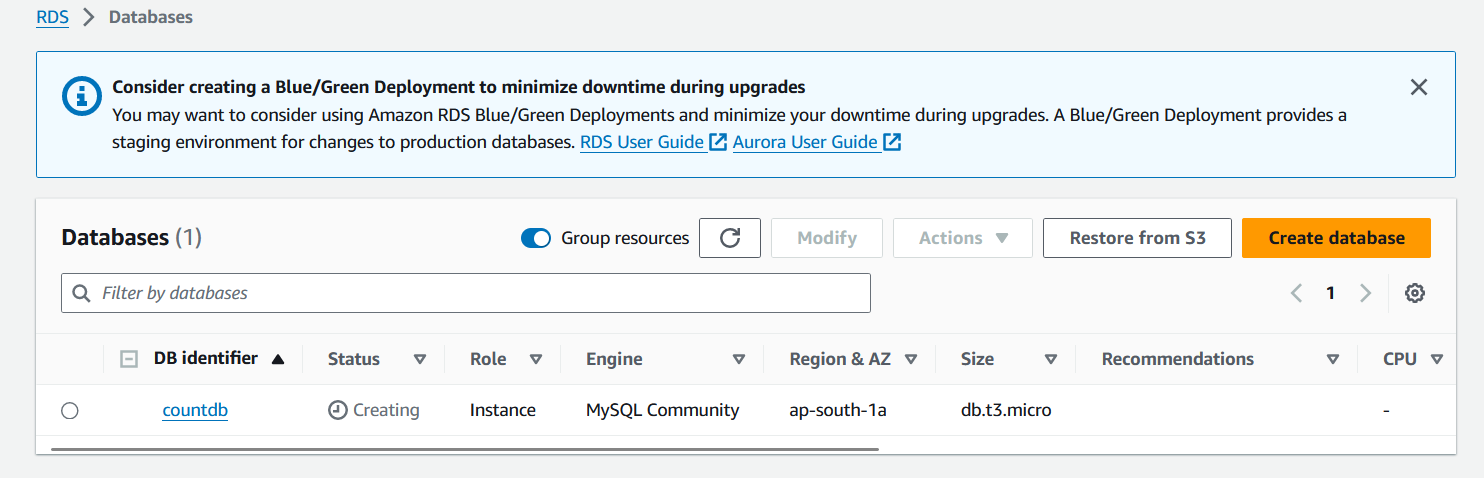






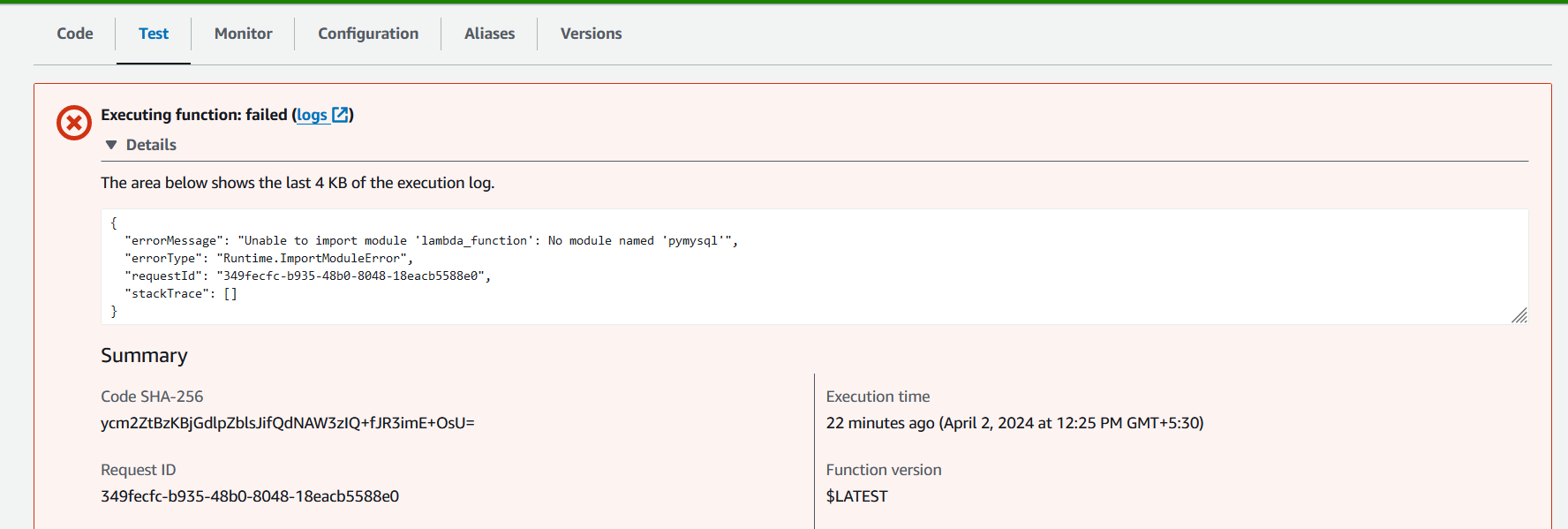
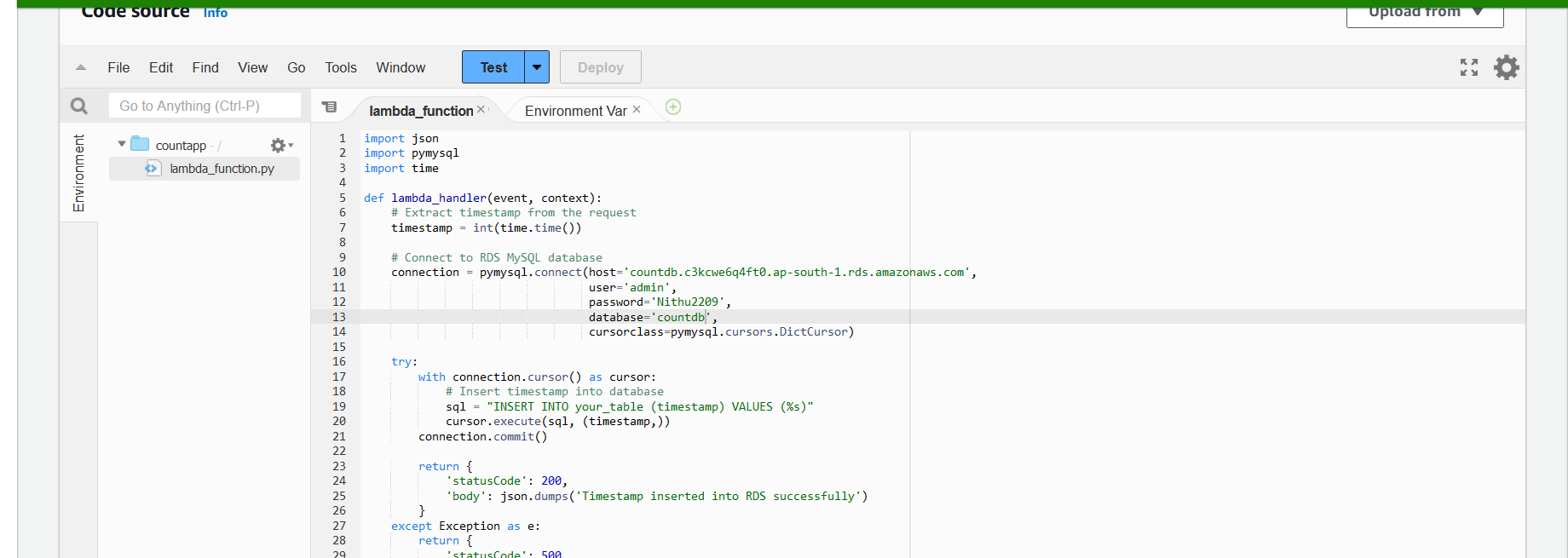
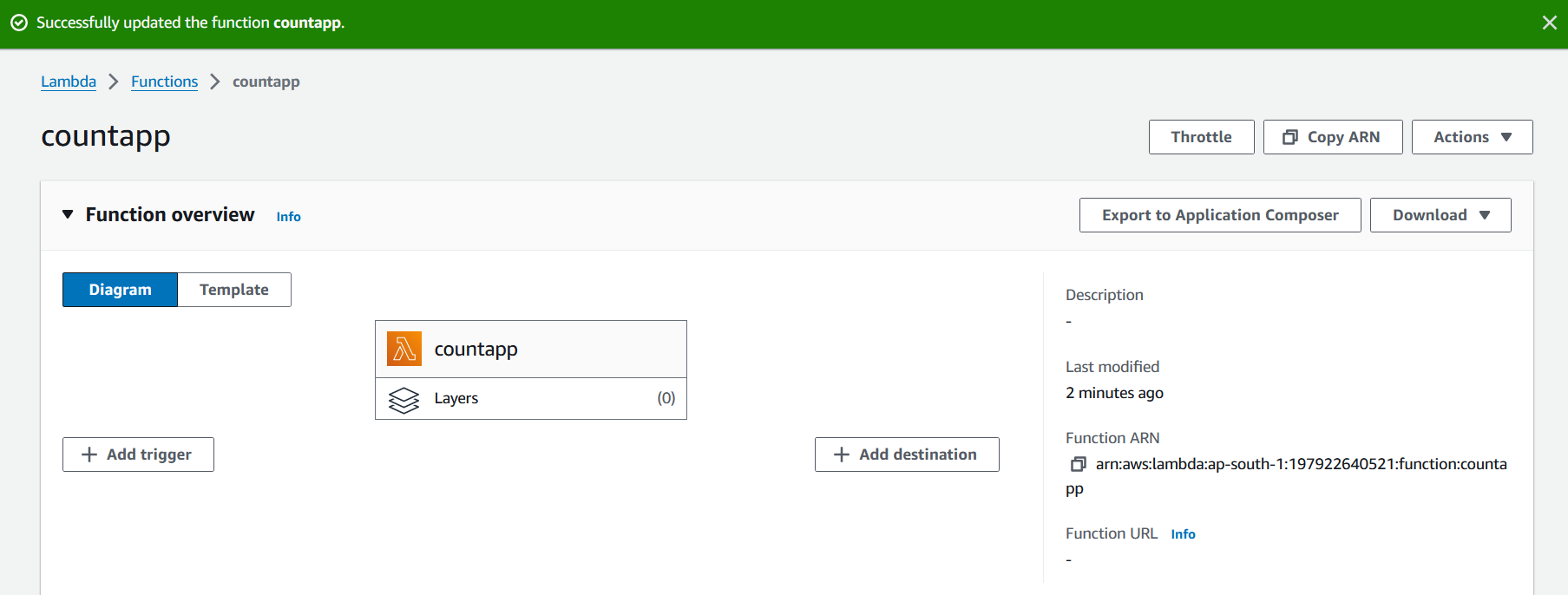


* Set up the MySQL database on Amazon RDS by configuring the necessary settings such as instance type, storage, username, and password.



1. **Use Lambda to Set up App Tier**:

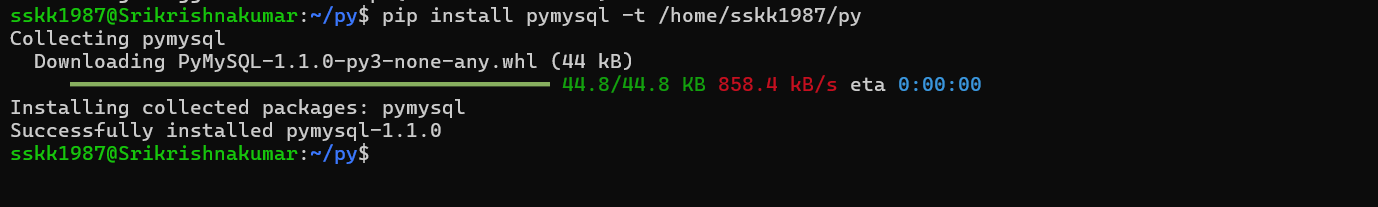
* Create a Lambda function that process the countdown and sends the timestamp from the frontend and inserts it into your RDS database.

The error message indicates that the pymysql module is not available in the Lambda execution environment, causing the Lambda function to fail during import.

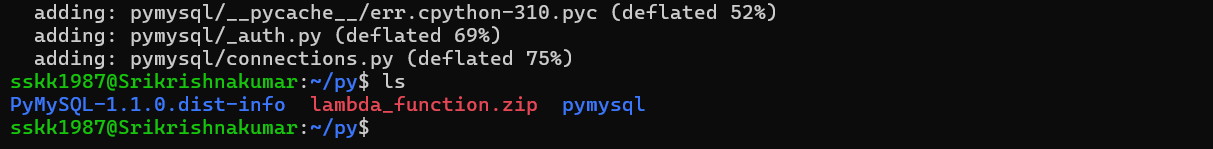
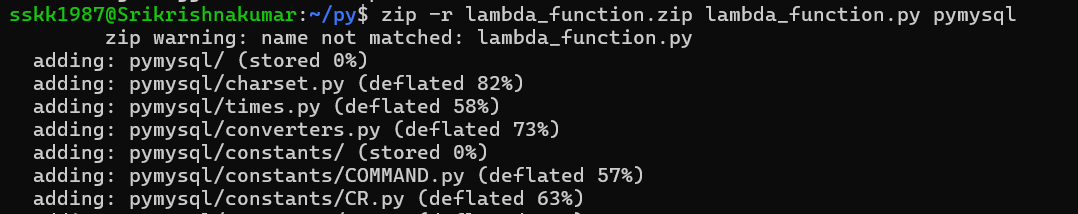
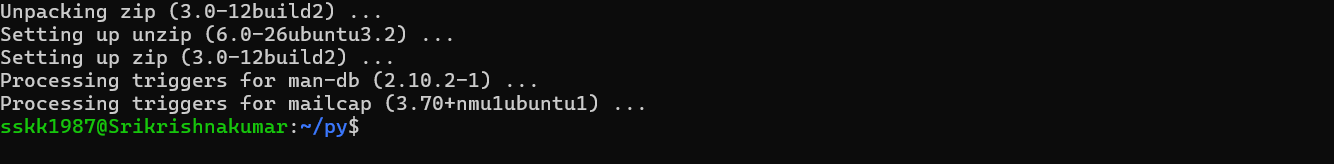
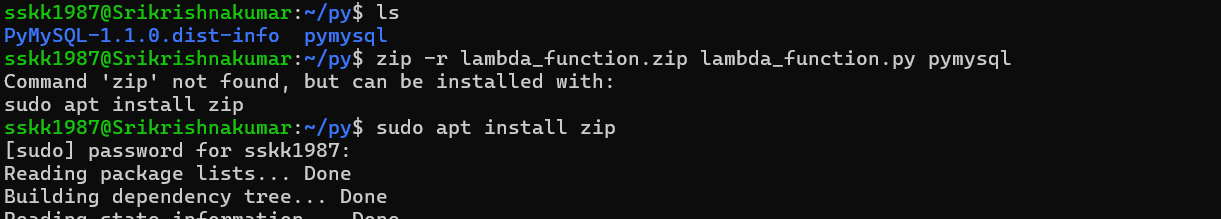
To resolve this issue, you need to include the pymysql module in your Lambda deployment package.

 **Create a Deployment Package**: You need to create a ZIP file containing your Lambda function code (lambda\_function.py) and any required dependencies, including the pymysql module.

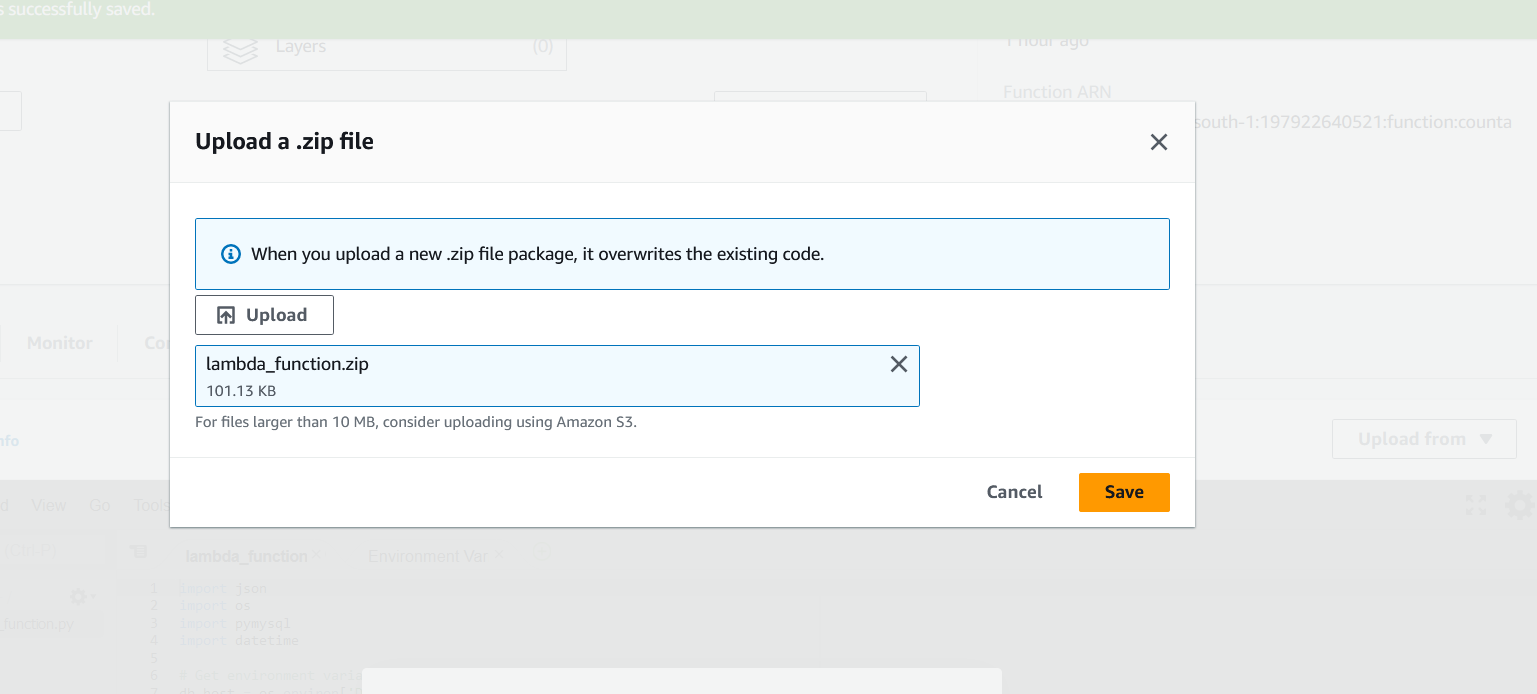
 **Install pymysql Locally**: On your local machine, install the pymysql module using pip

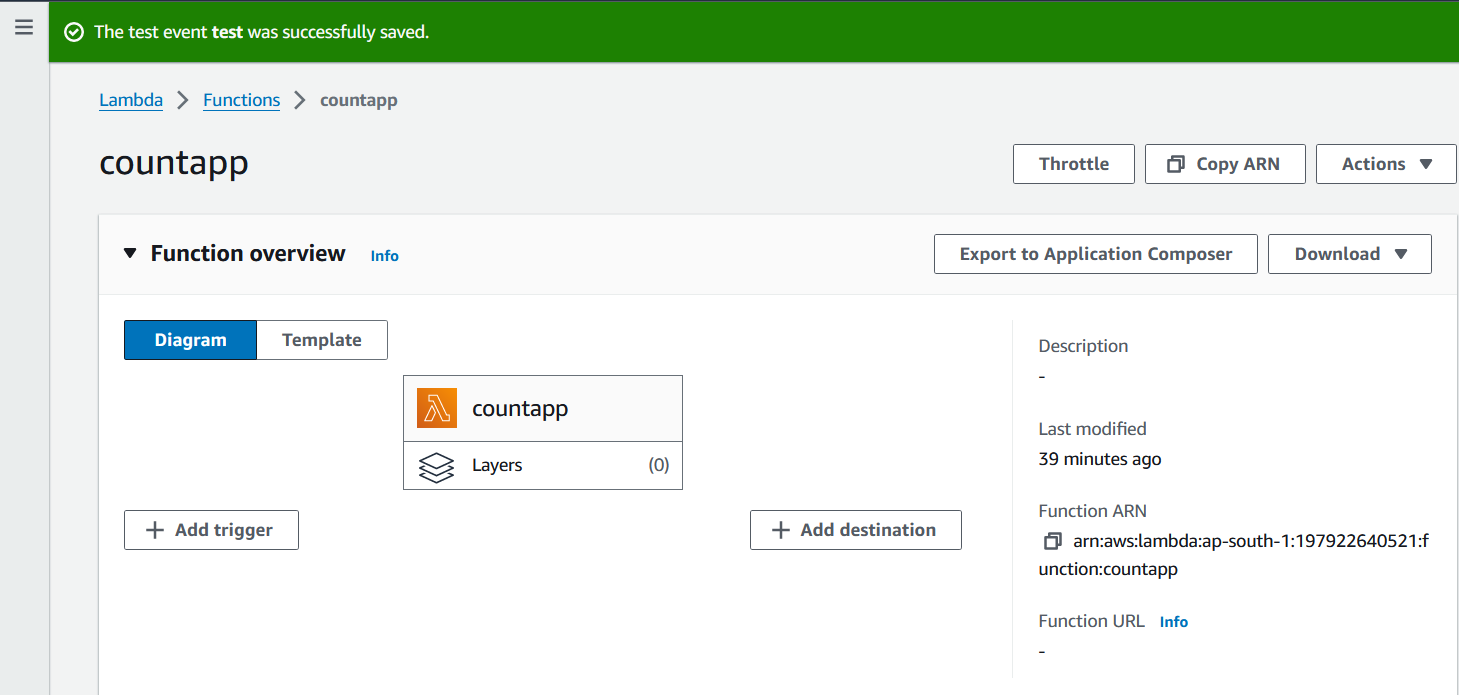


 **Create the Deployment Package**: After installing pymysql, create a ZIP file containing your Lambda function code and the pymysql module

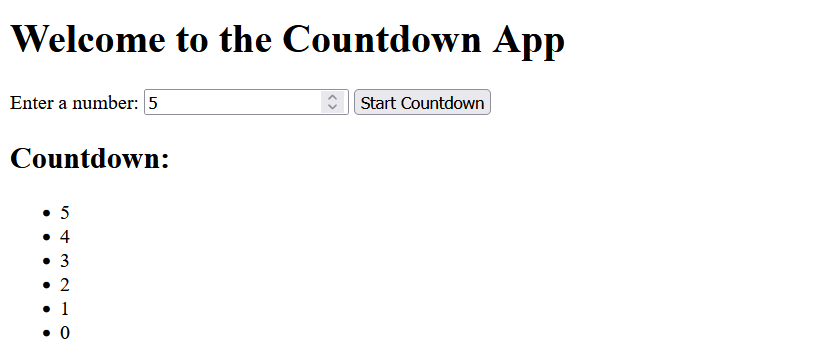


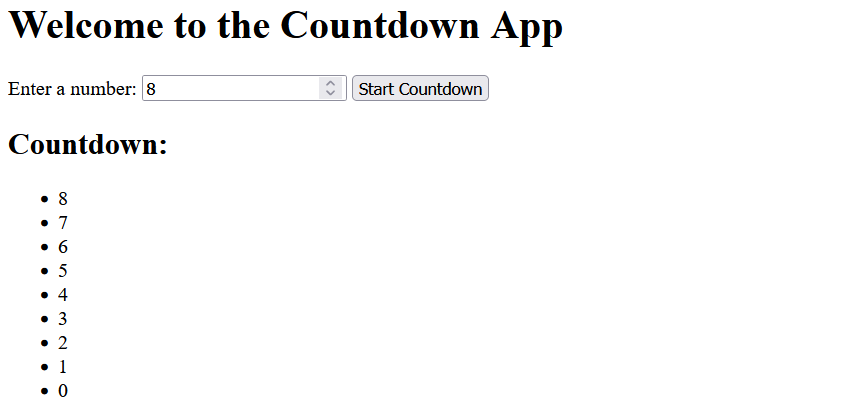
 **Upload the Deployment Package**: Upload the lambda\_function.zip file to AWS Lambda using the AWS Management Console or AWS CLI.

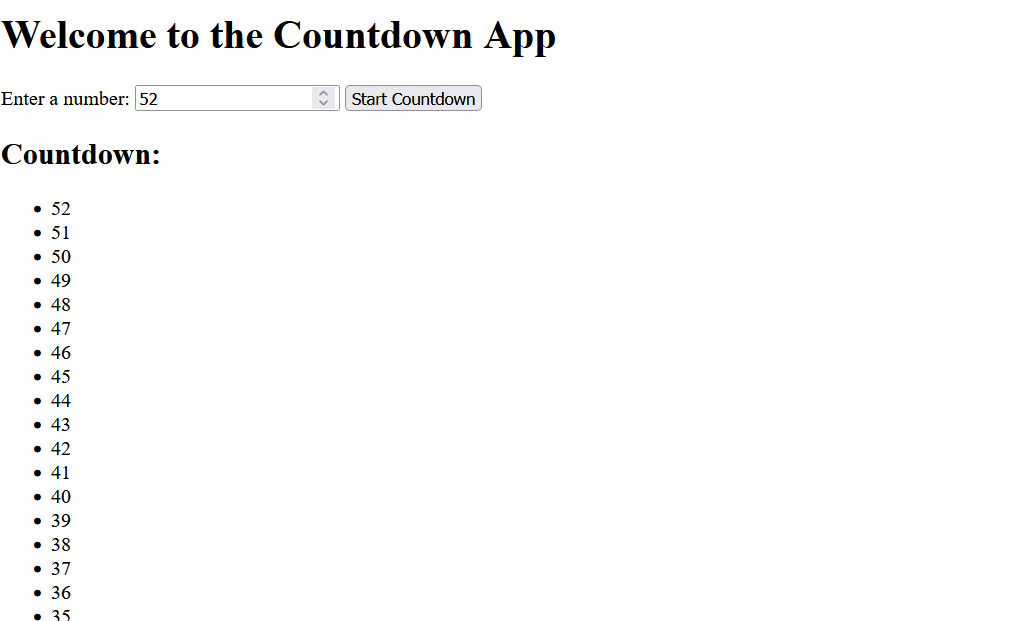




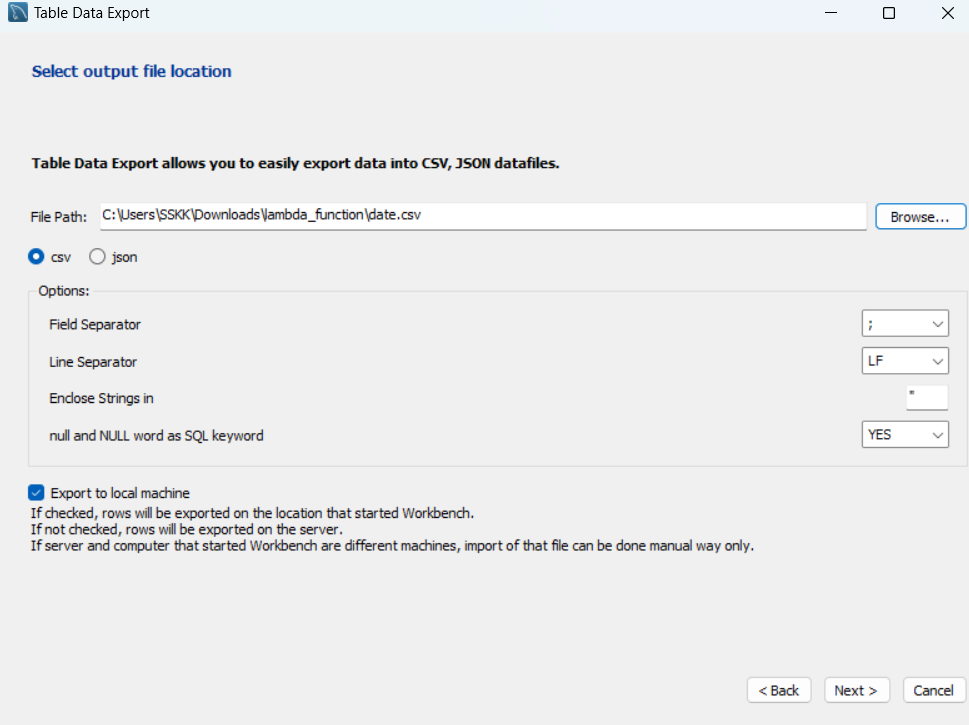
1. **Test the site**

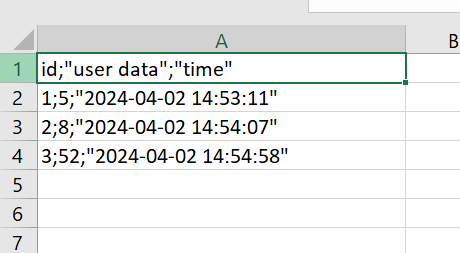






1. **Use an external export module to read the data from the database**

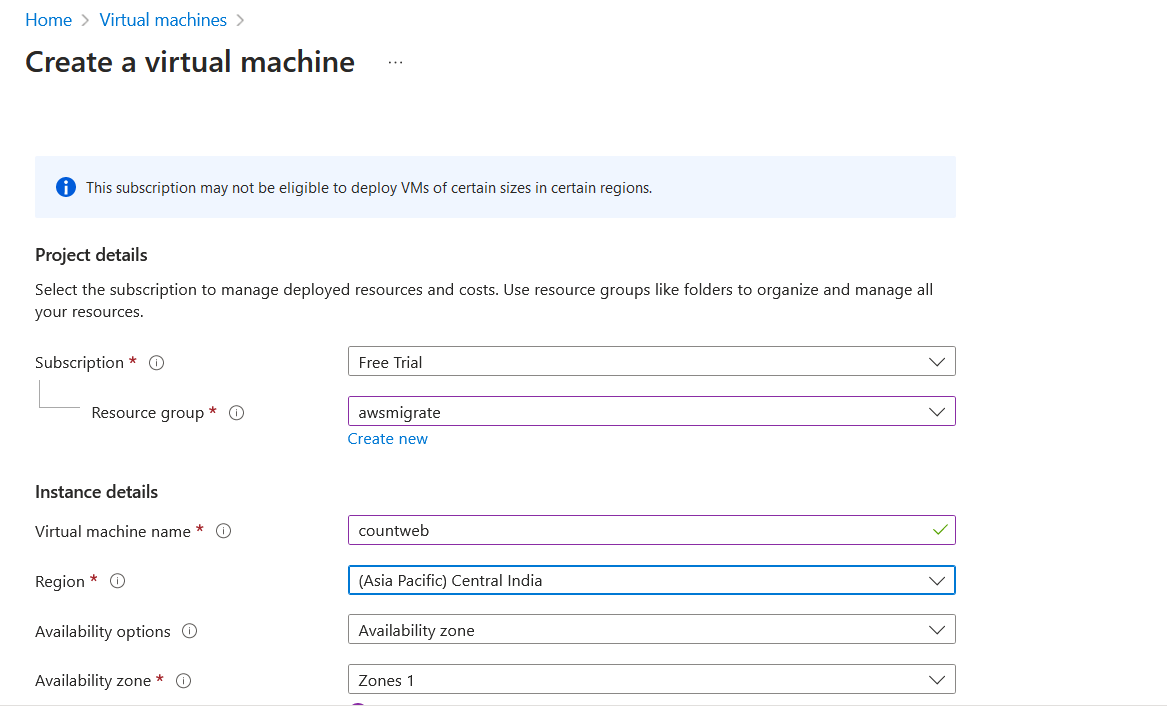


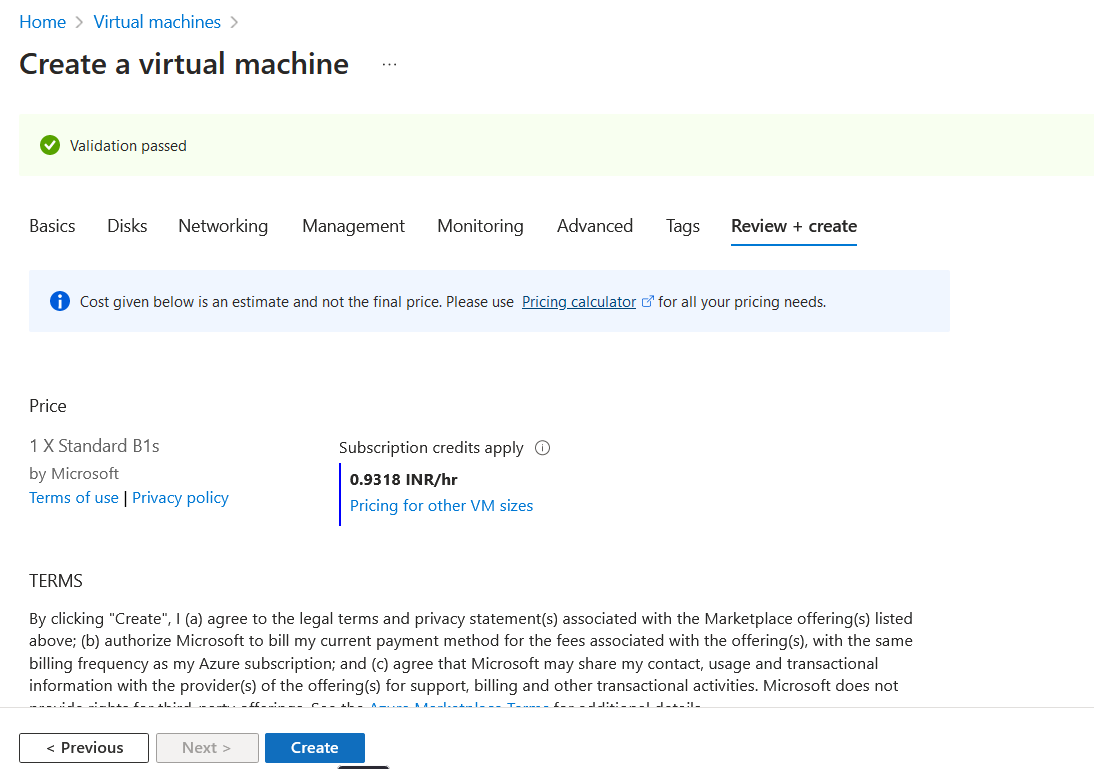


**Replicate same steps as above using azure**

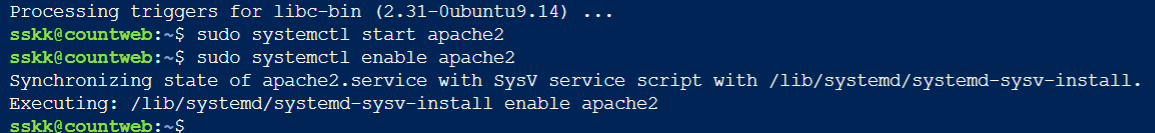
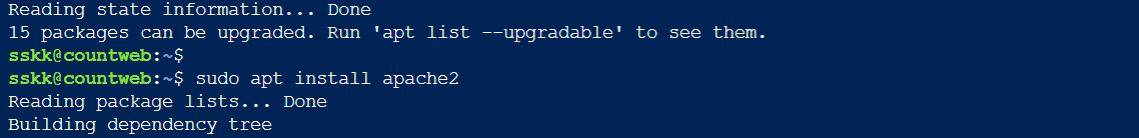
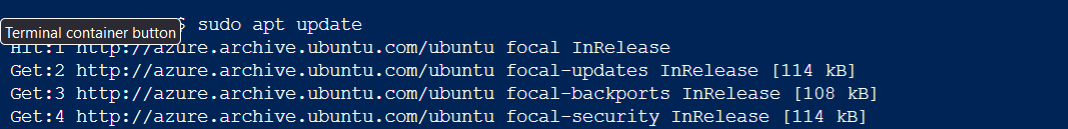
1. **Frontend Setup**:

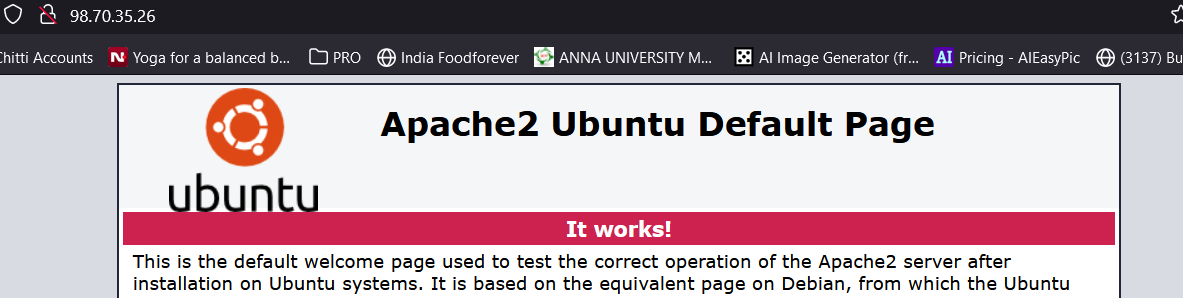
* Create a Virtual Machine (VM) in Azure to host your frontend application. Choose the appropriate VM size and operating system based on your requirements.

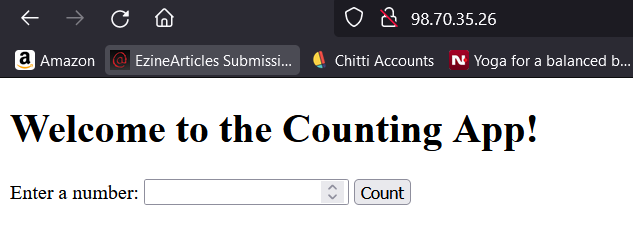




* Install and configure the necessary software and dependencies for your frontend application on the VM.





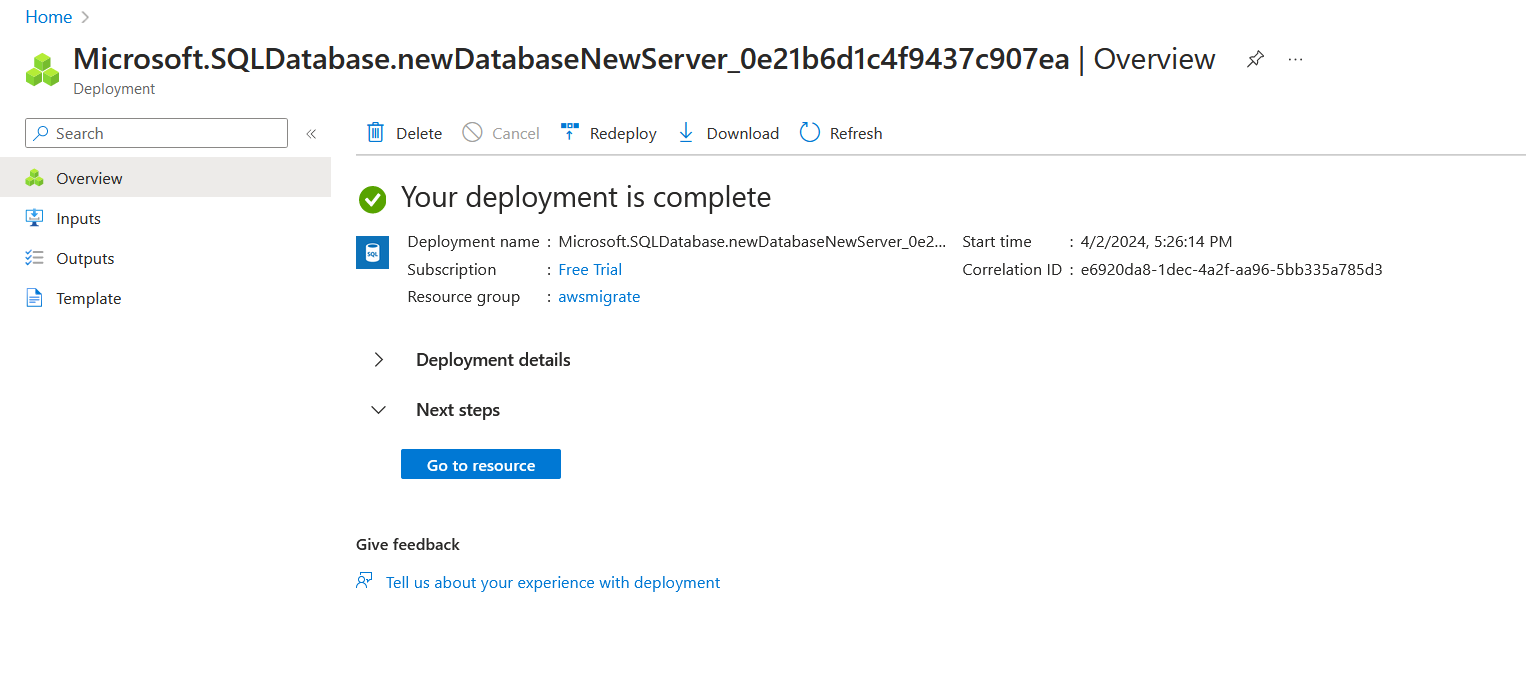


* Deploy your frontend application code to the VM.

1. **Database Setup:**

* Create an Azure SQL Database instance to serve as your data tier. Choose the appropriate pricing tier and configuration based on your database requirements.

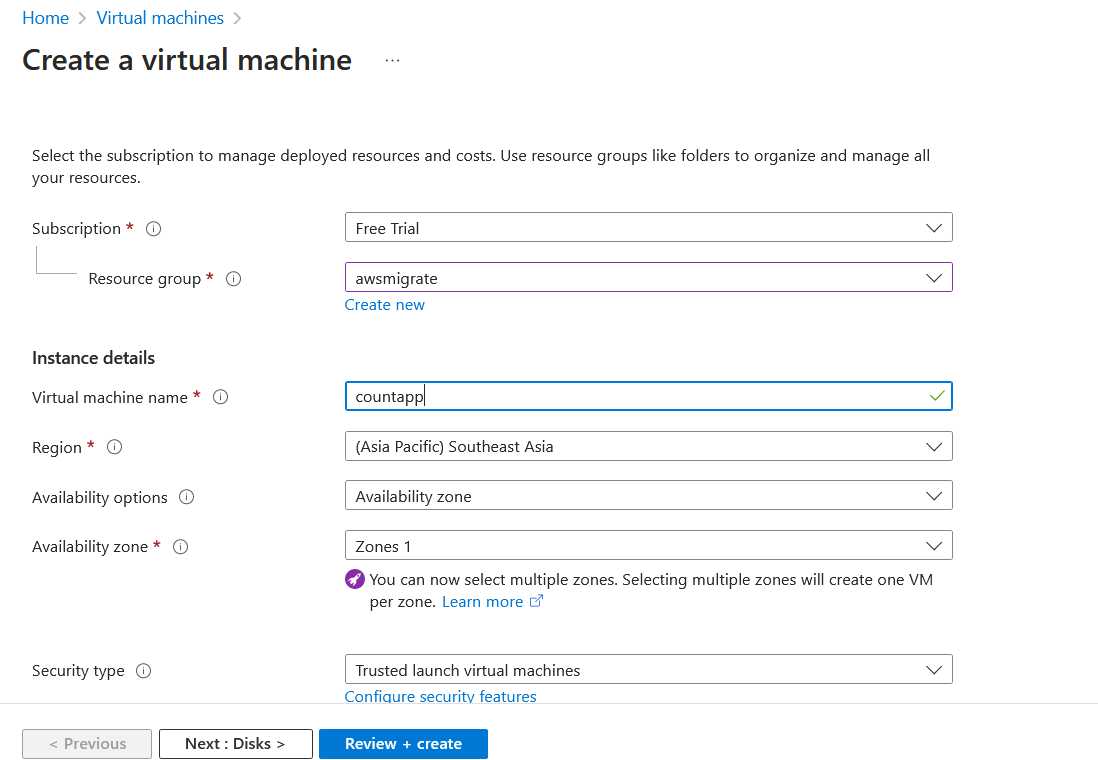


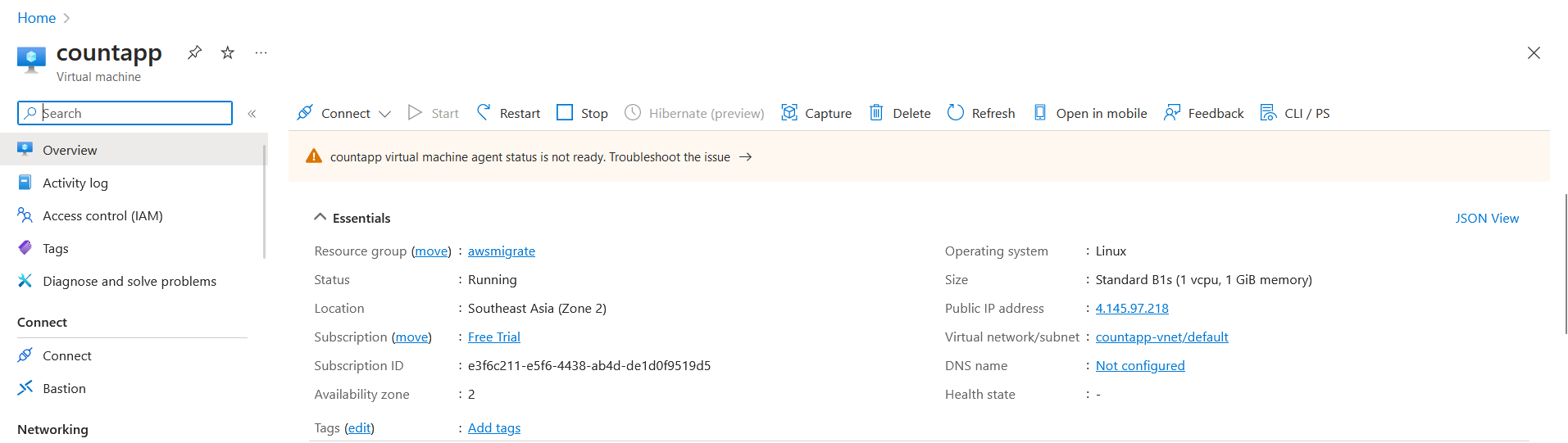
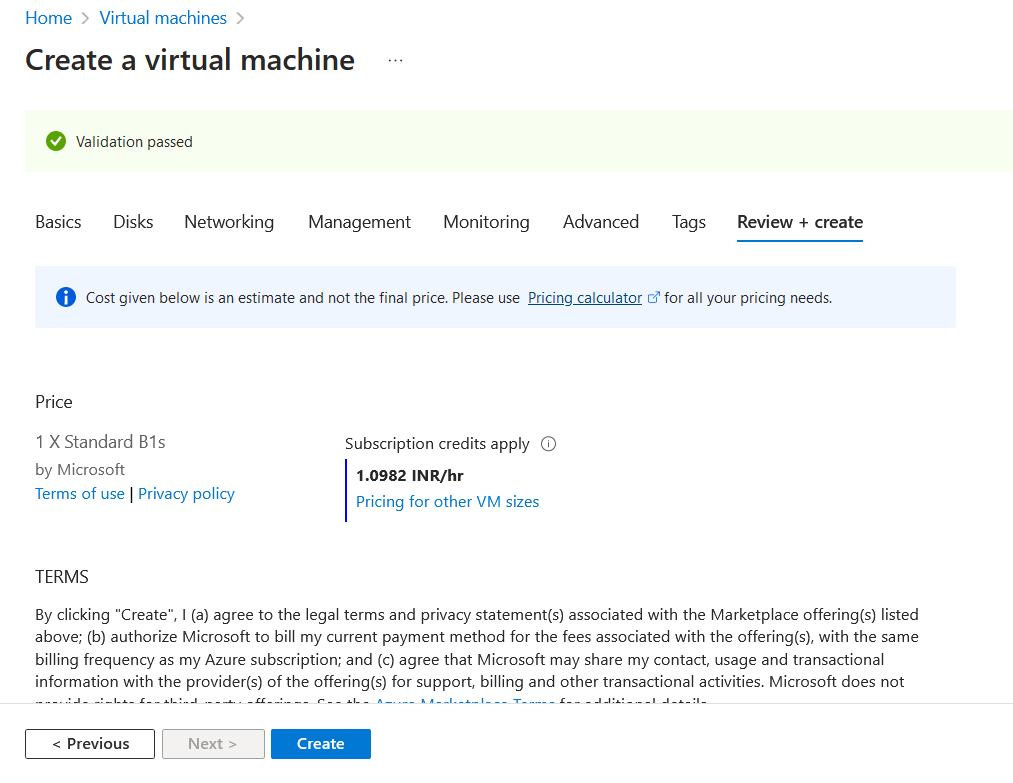


* Use Azure Database Migration Service if you need to migrate data from an existing database to Azure SQL Database.
* Connect your backend application to the Azure SQL Database instance and configure access permissions.

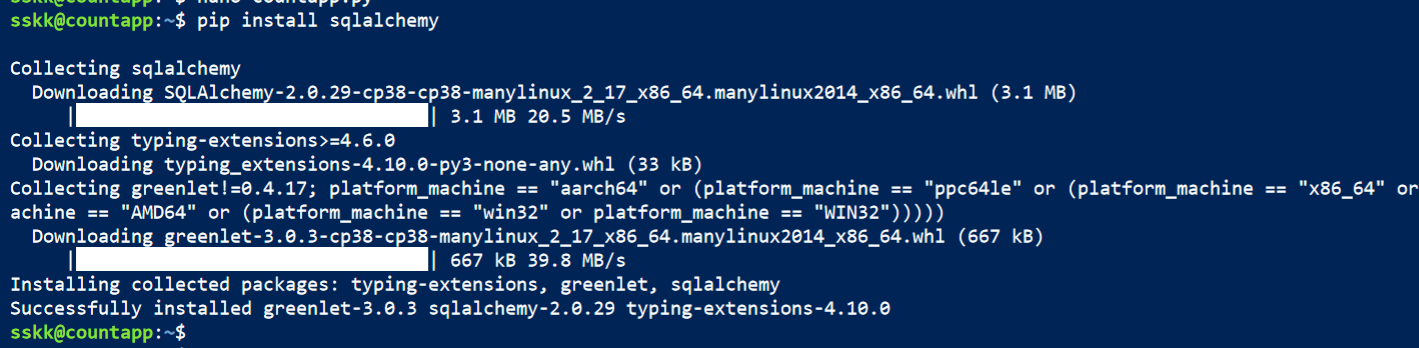
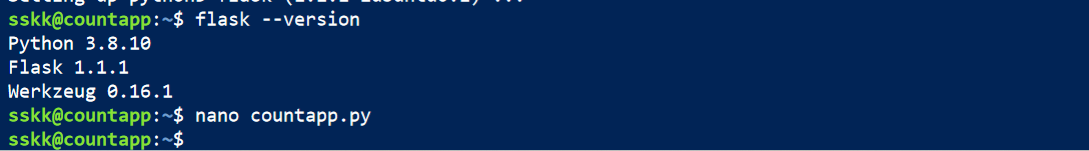
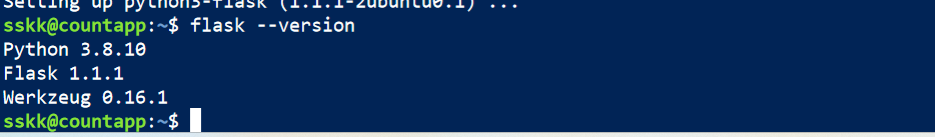
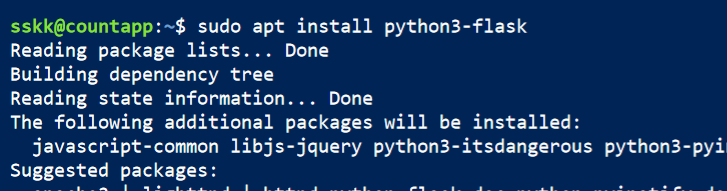
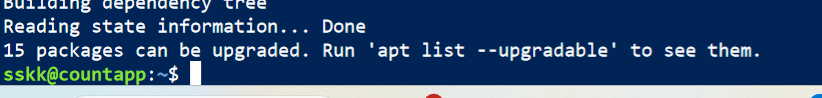
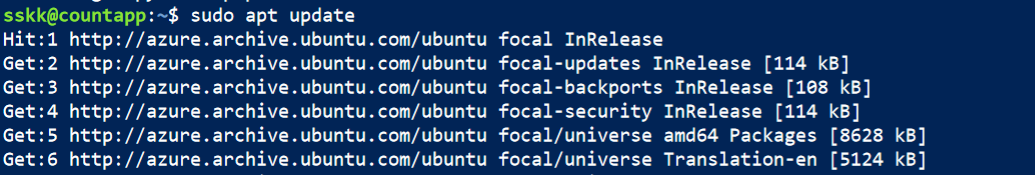
1. **Backend Setup:**

* Similar to the frontend, create another Virtual Machine in Azure to host your backend application.





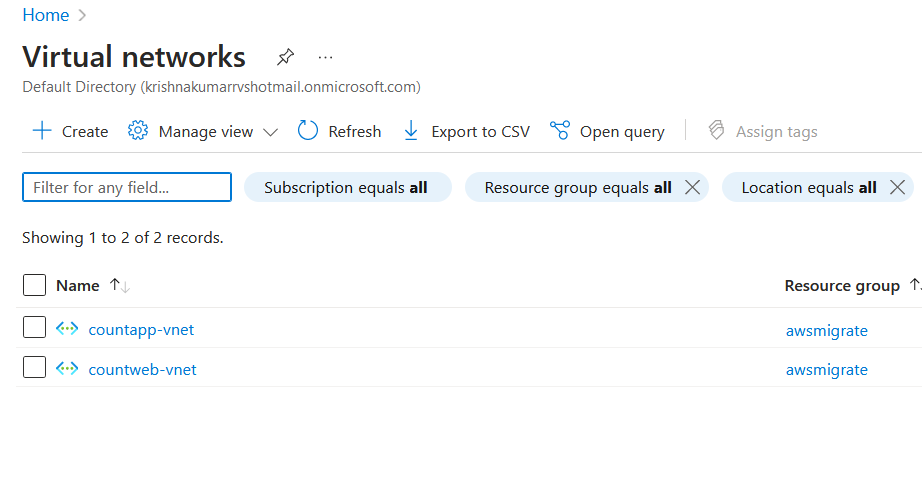
* Install and configure the necessary software stack (e.g., Flask, Django, Node.js) for your backend application on the VM.

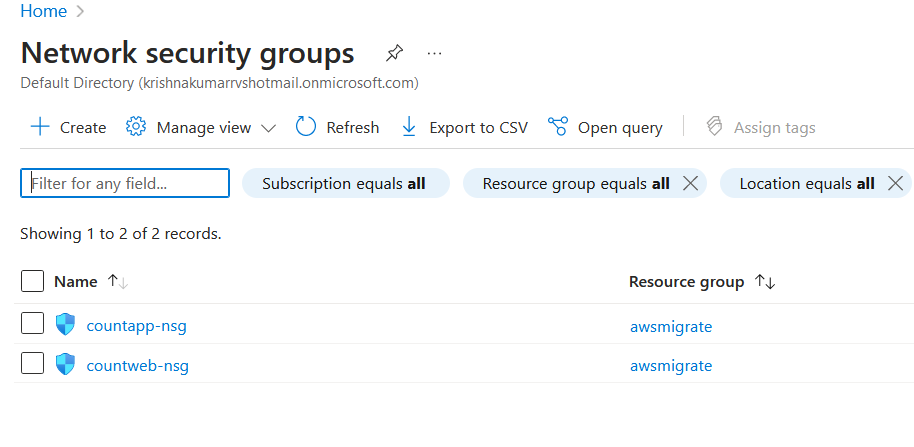


* Deploy your backend application code to the VM.

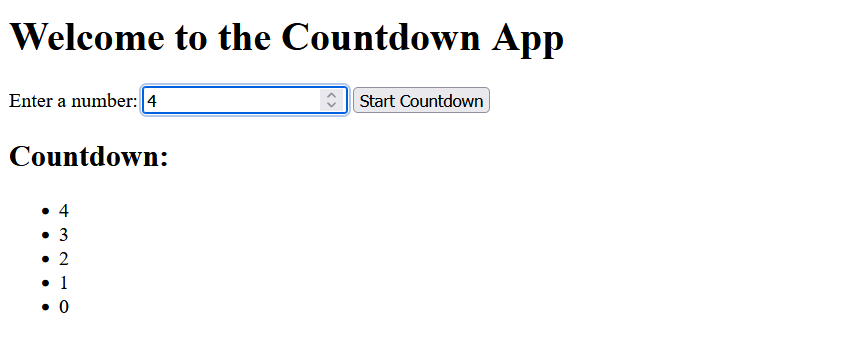
 **Networking and Security**:

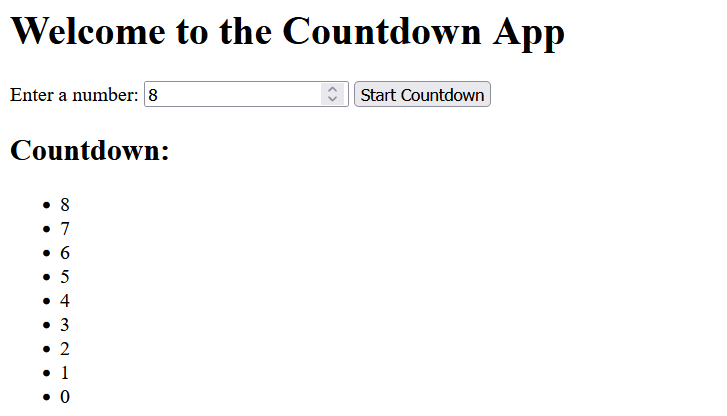
* Set up Virtual Networks (VNets) in Azure to isolate and securely connect your frontend, backend, and database components.

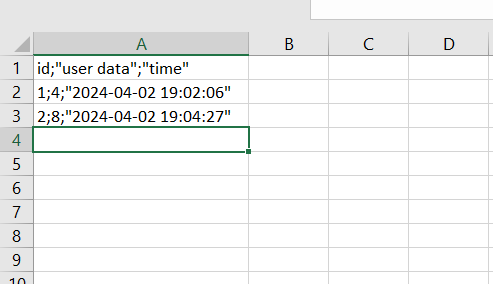


* Configure Network Security Groups (NSGs) to control inbound and outbound traffic to your VMs and database. 

1. **Test the site and database**







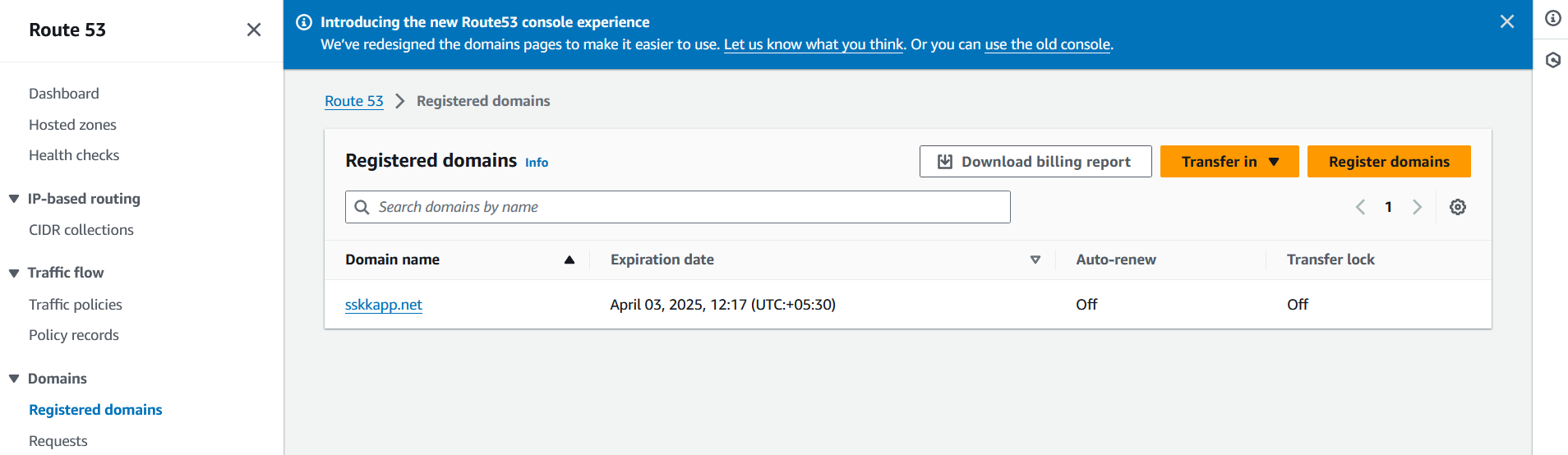
**Steps for global peering using both servers**

To implement global traffic management between AWS and Azure using AWS Route 53 or Azure Traffic Manager, you can follow these steps:

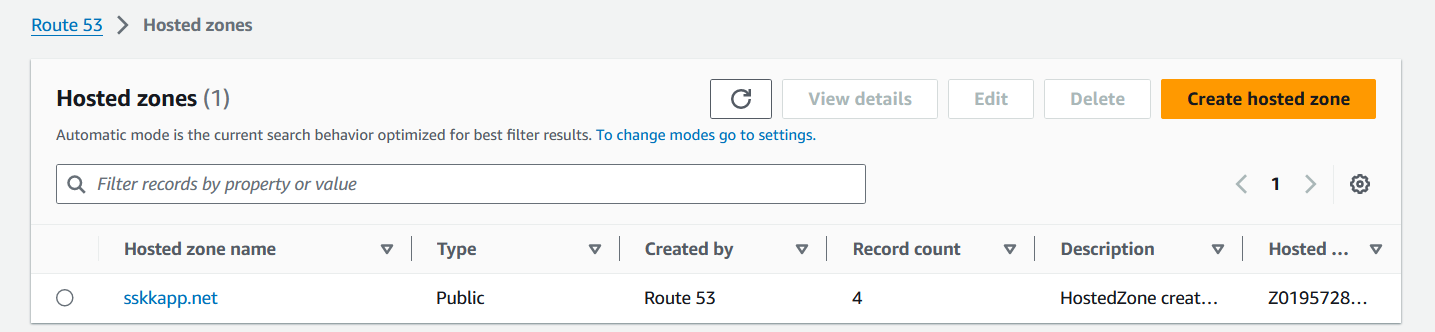
To implement global traffic management between AWS and Azure using AWS Route 53, you can follow these steps:

* 1. **Navigate to Route 53 and register a domain**

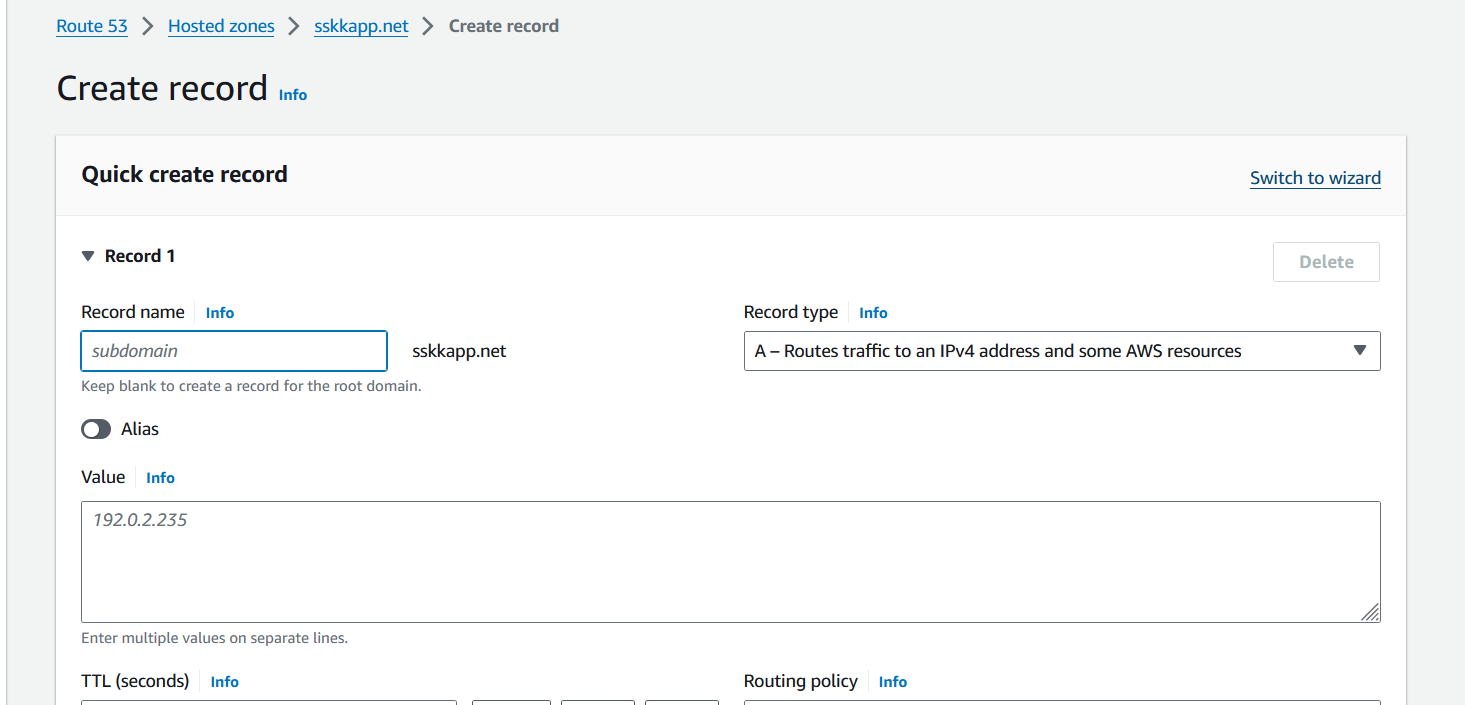


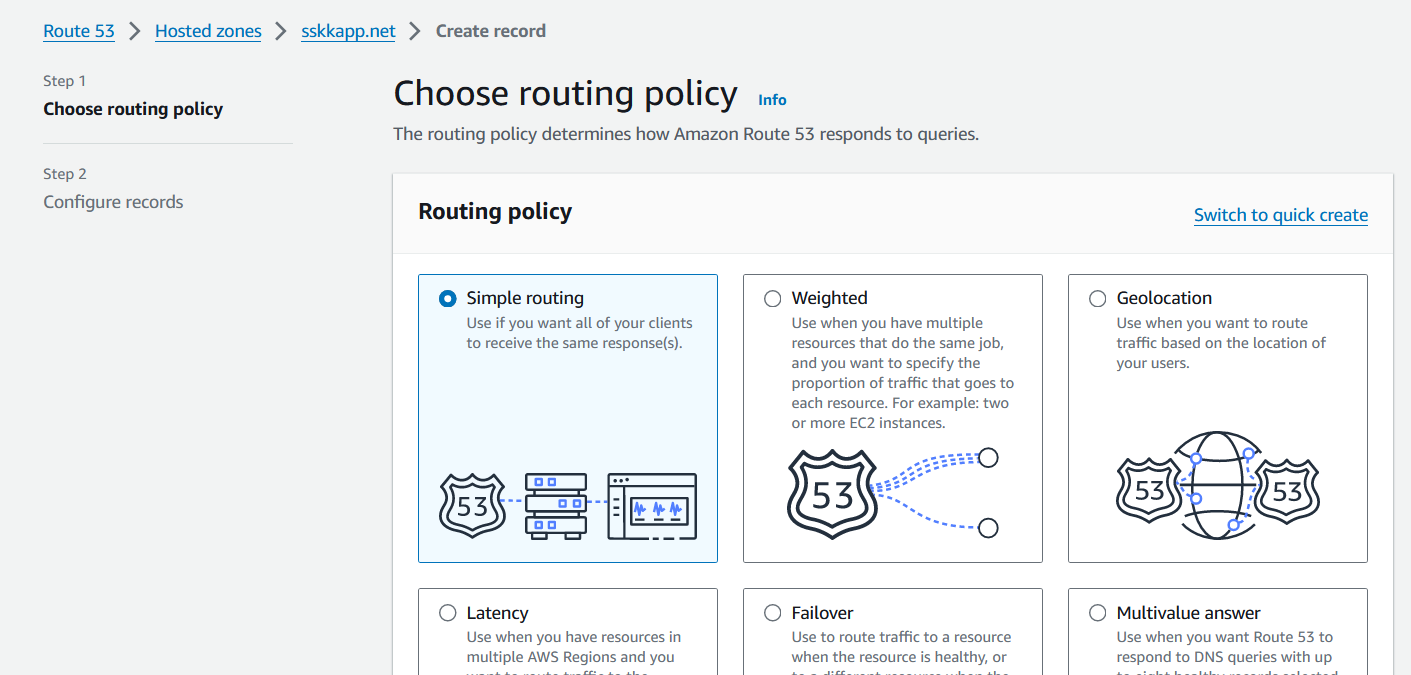


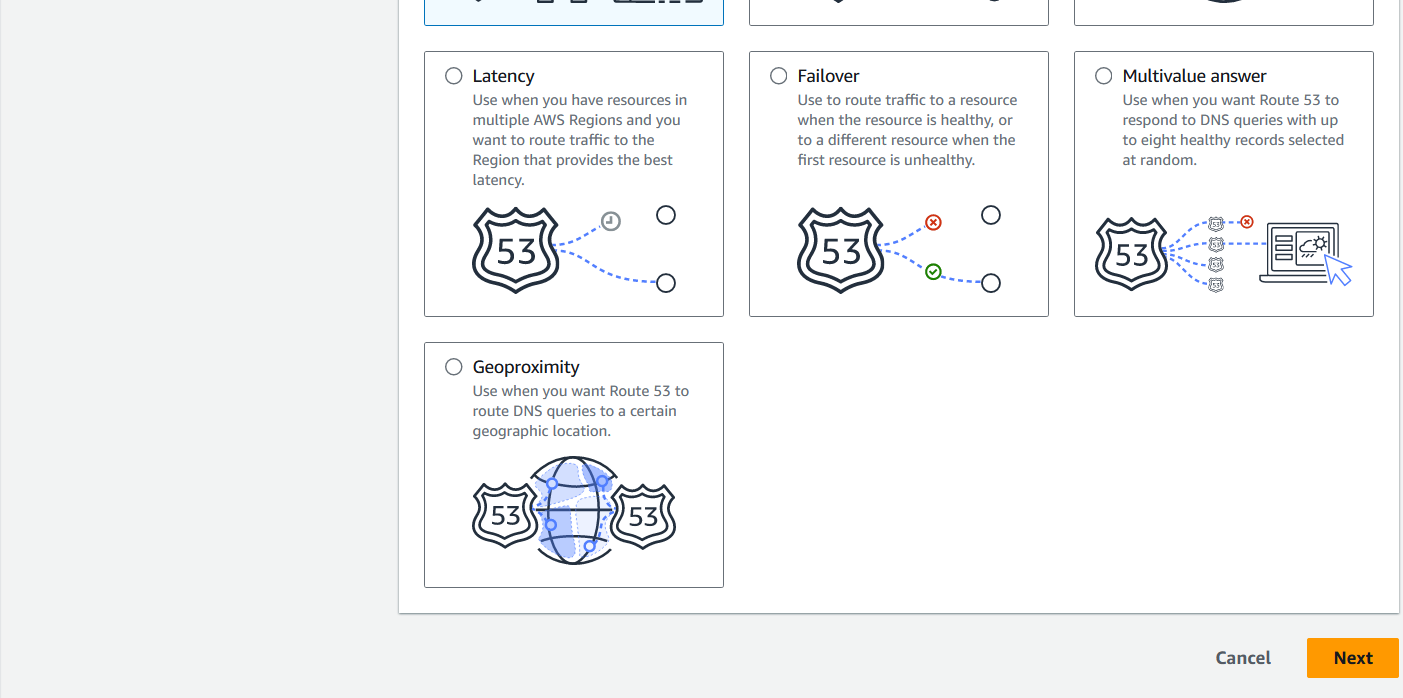
* 1. **Create a Hosted Zone**

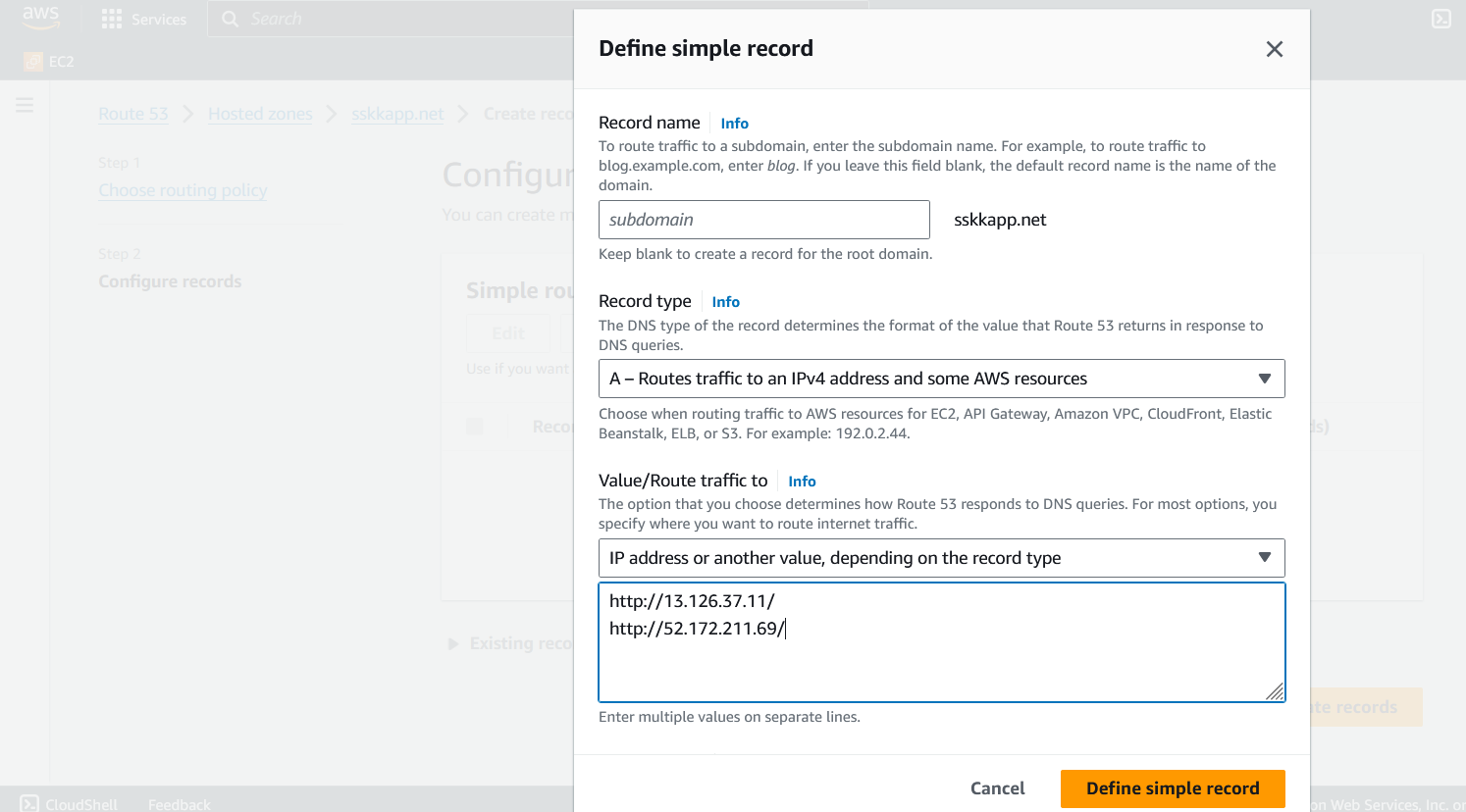
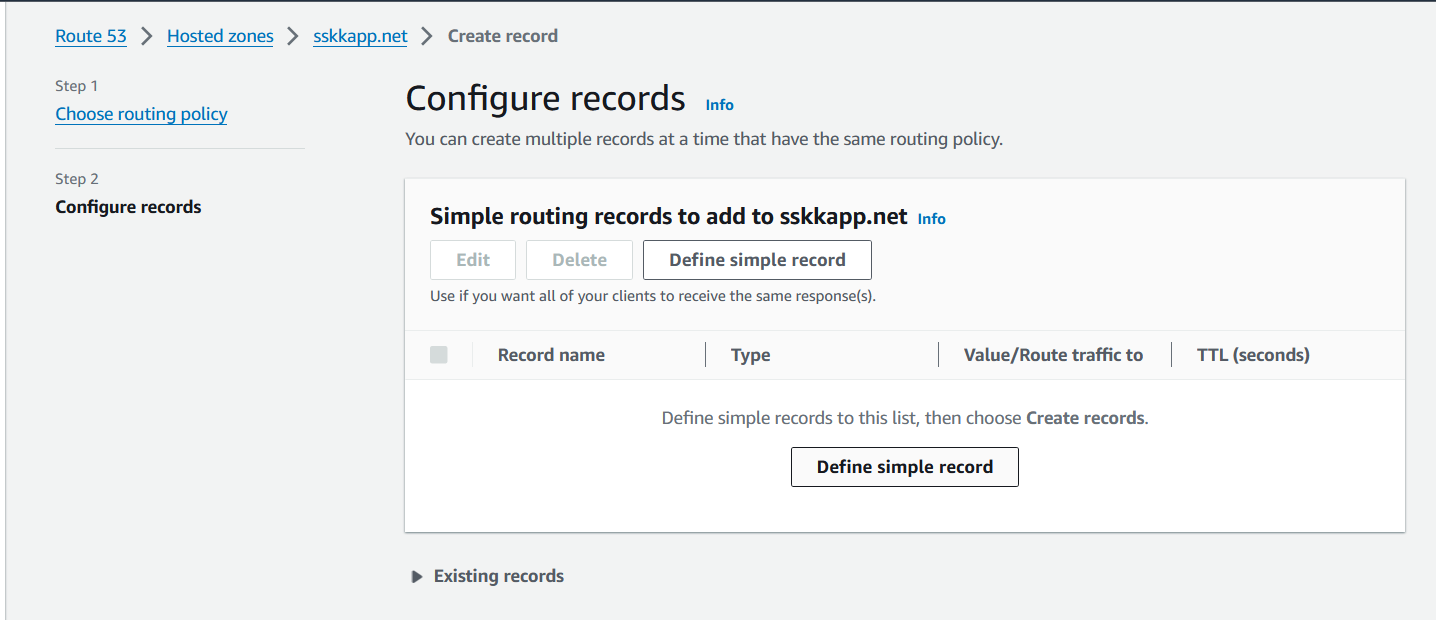


When we register a domain using the route53 domain registration, a default hosted zone is created by the aws.

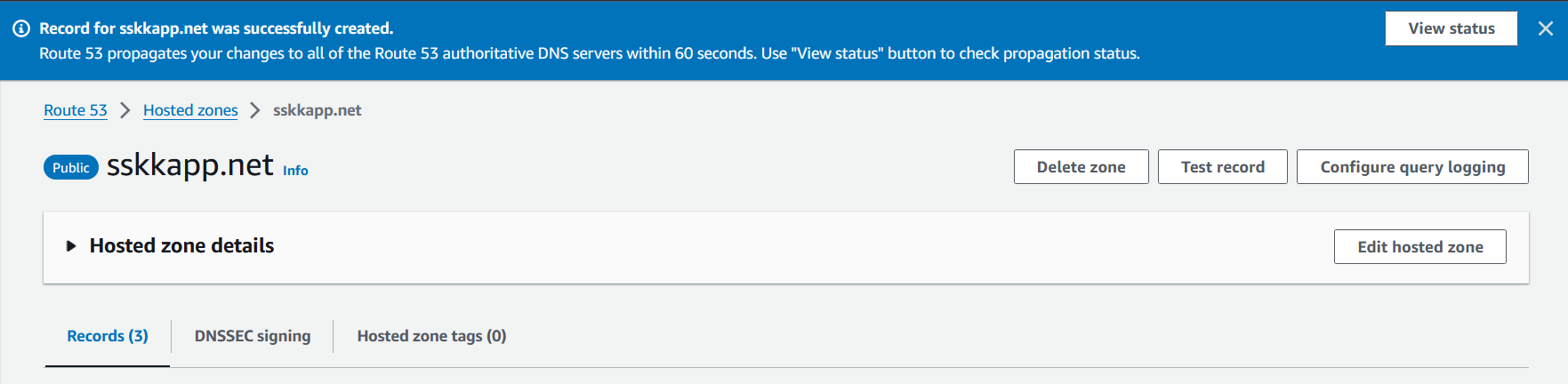
* 1. **Create Record Sets**: Inside your hosted zone, create record sets to define the DNS records for your domain. You'll typically create an A record for each endpoint in AWS and Azure that you want to route traffic t











* 1. **Test and Monitor**: After configuring the record sets and routing policies, thoroughly test your setup to ensure that traffic is correctly routed between AWS and Azure based on your chosen policies. Monitor the performance and availability of your endpoints to ensure that failover mechanisms work as expected. Since we have chosen simple routing, the traffic is not routed to any specific servers for specific conditions, so refresh the page repeatedly to observe the traffic routing. For easy observation, the landing page on both servers are modified to show their origin server.

