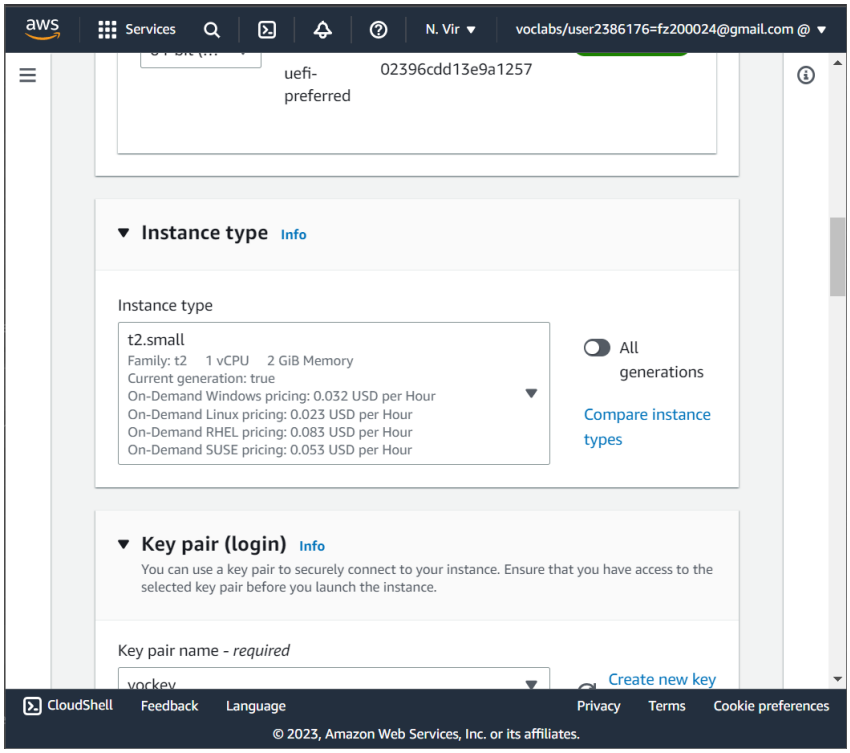
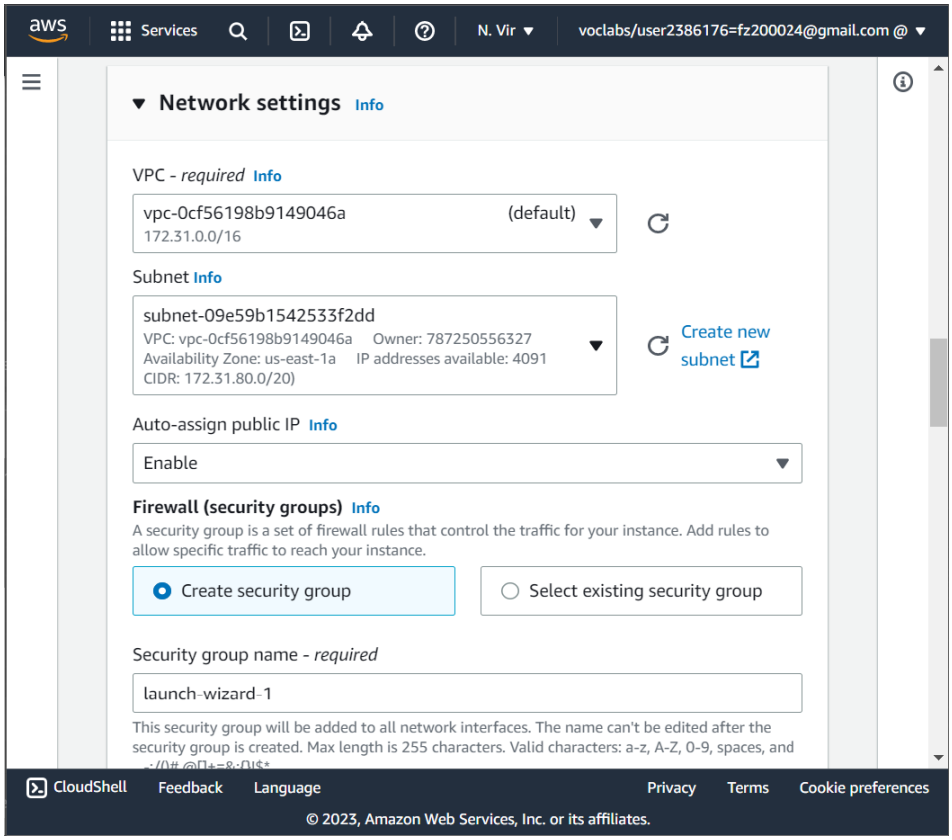
PART A:

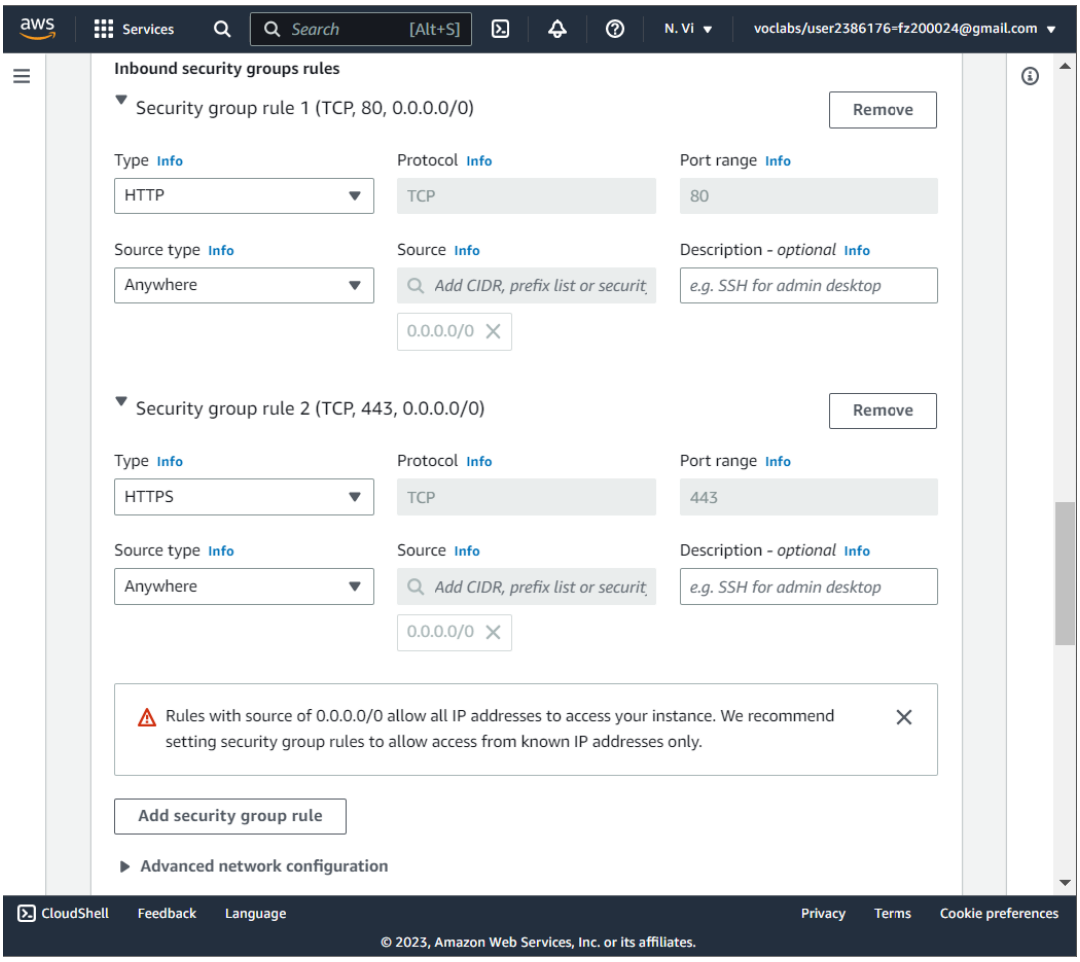
To begin the creation of the Virtual Machine (VM) with the given requirements, an instance of the EC2 server is created by the name of “Main Server” with the t2.small instance type having a single core with 2GBs of RAM.



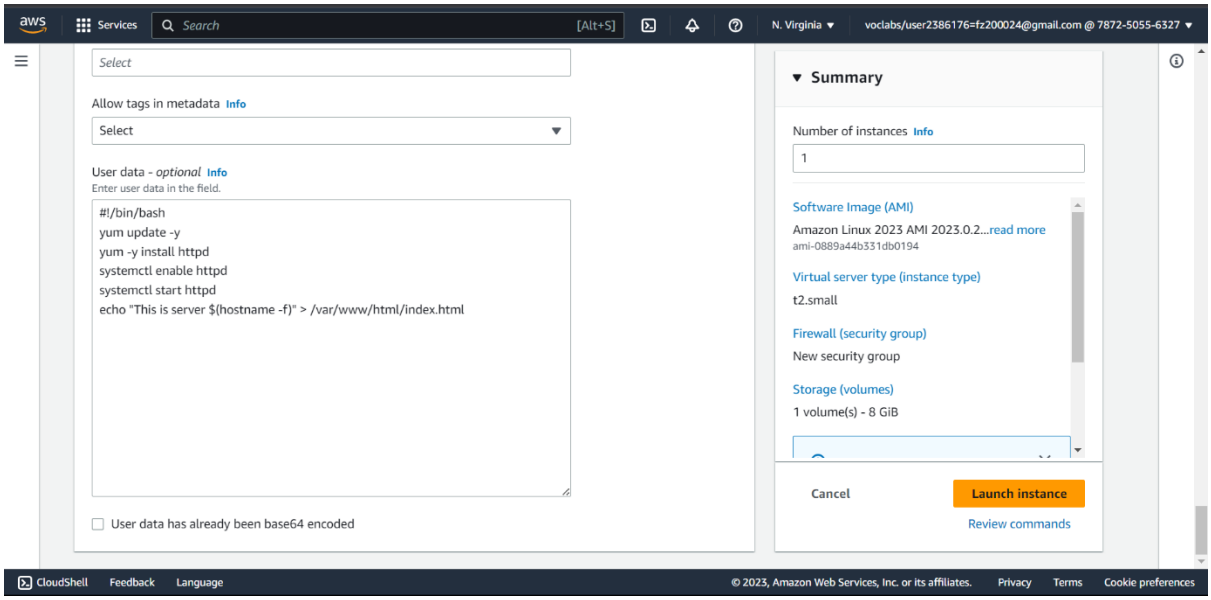
This Main Server is assigned to the us-east-1a availability zone.



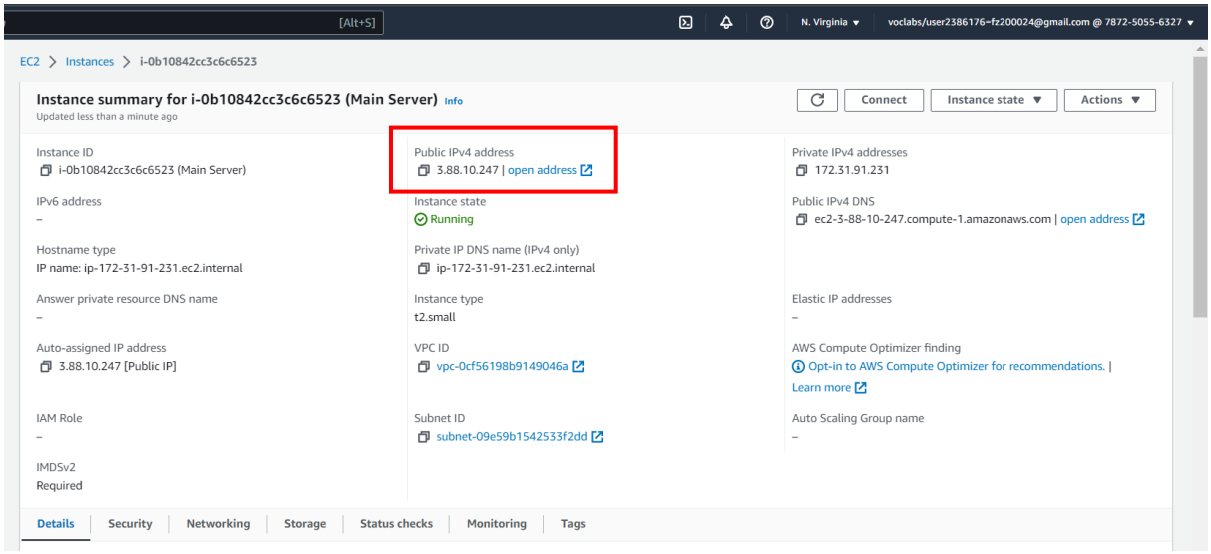
The security group rules are configured for making the server accessible through ports 80 and 443 respectively.



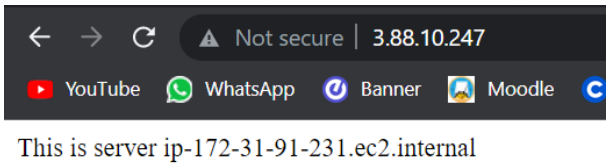
In the user data text field, the following HTML code is entered for the server to be able to render an HTML page that displays the private IP address of the EC2 instance.



The Main Server EC2 instance has been created.

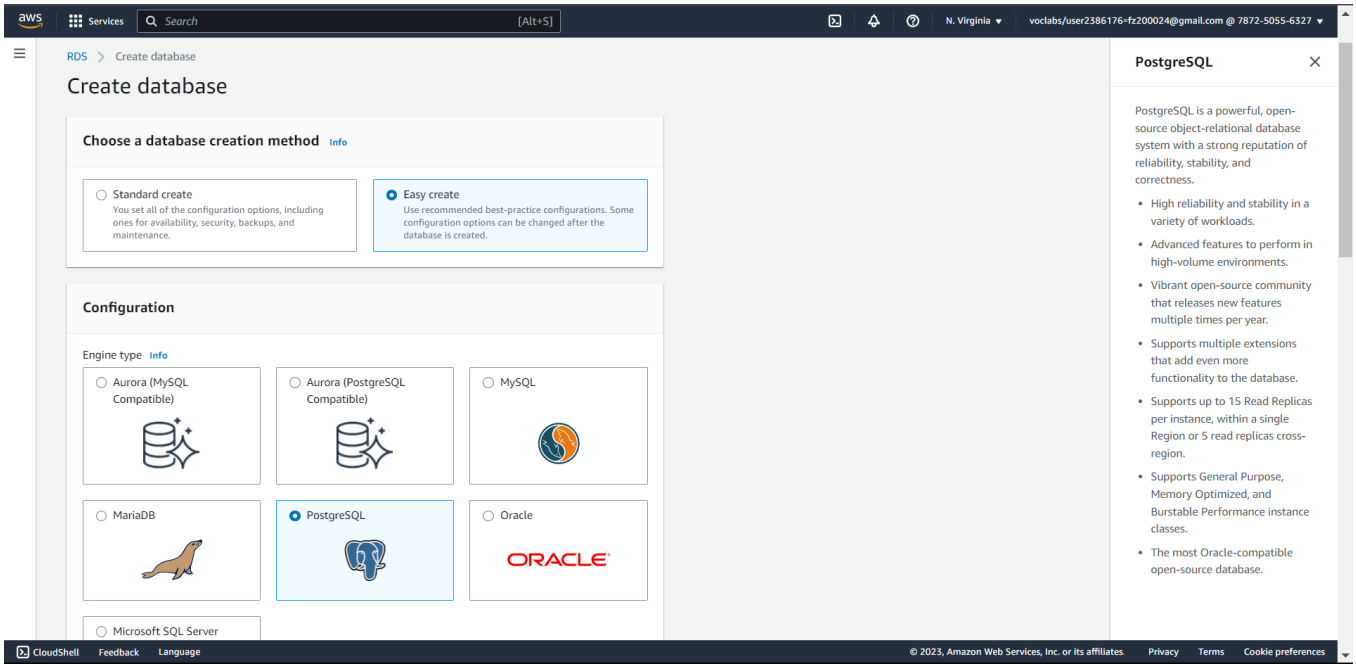


Upon entering the public IP address into the search bar as highlighted above, the following HTML page is displayed:

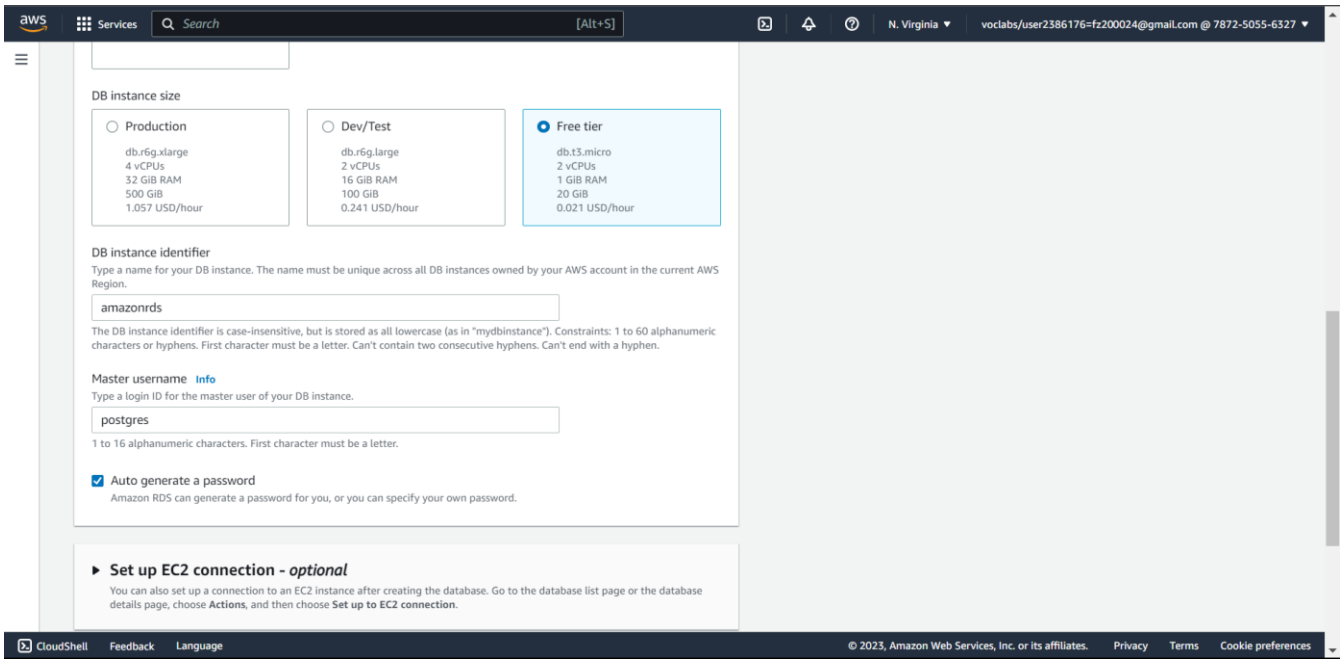


This is server ip-172-31-91-231.ec2.internal

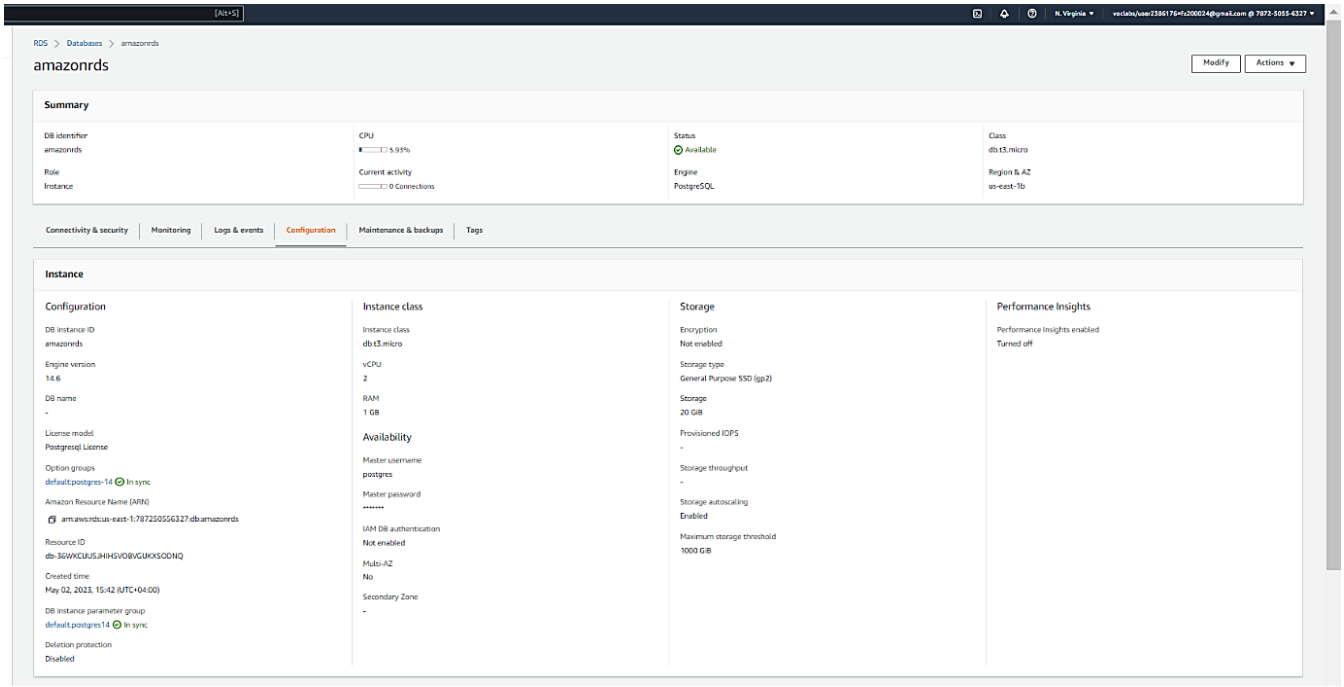
Now to create the RDS instance, the creation method will be “Easy Create” and will use the PostgreSQL configuration.



The DB instance size is selected as Free Tier and is named “amazonrds”. It is also configured to autogenerate a password upon creation of the instance.



The RDS instance has been created, with the following configurations:



After this, two DynamoDB tables are created. The first one is created with the name “Customer_Information”. The “ID” is used as partition key, which is of type Number, and the “Name” is used as sort key, which is of type String.

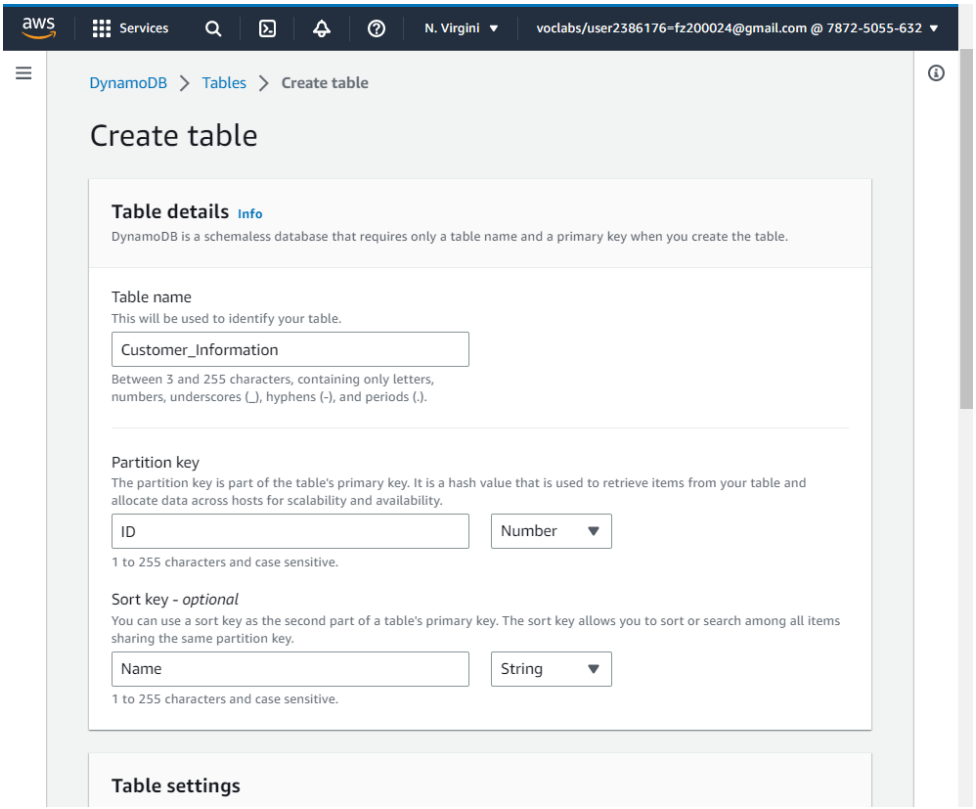
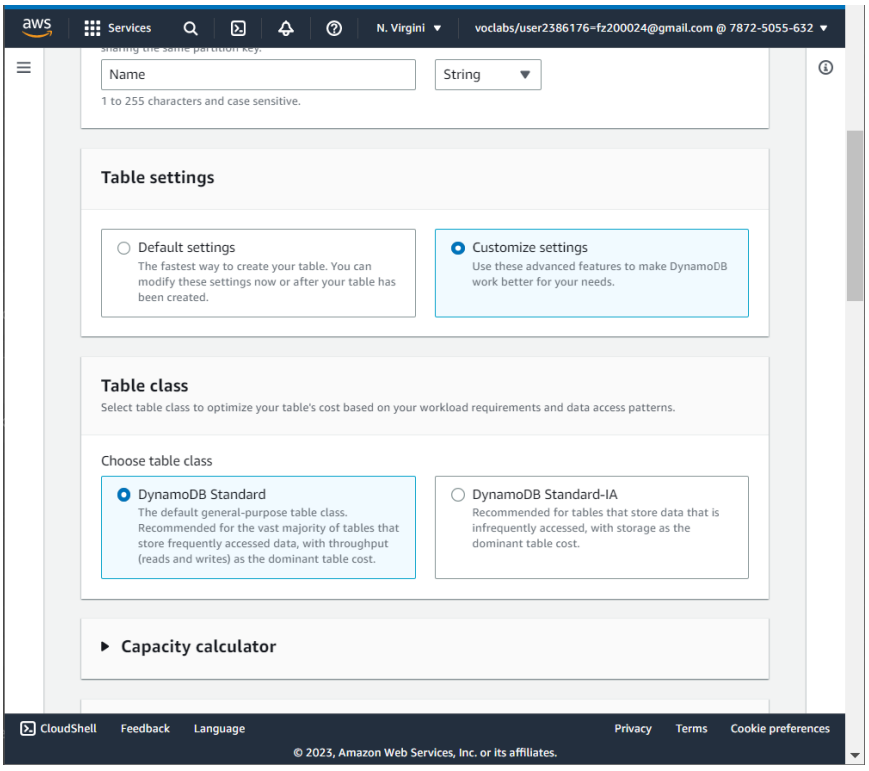
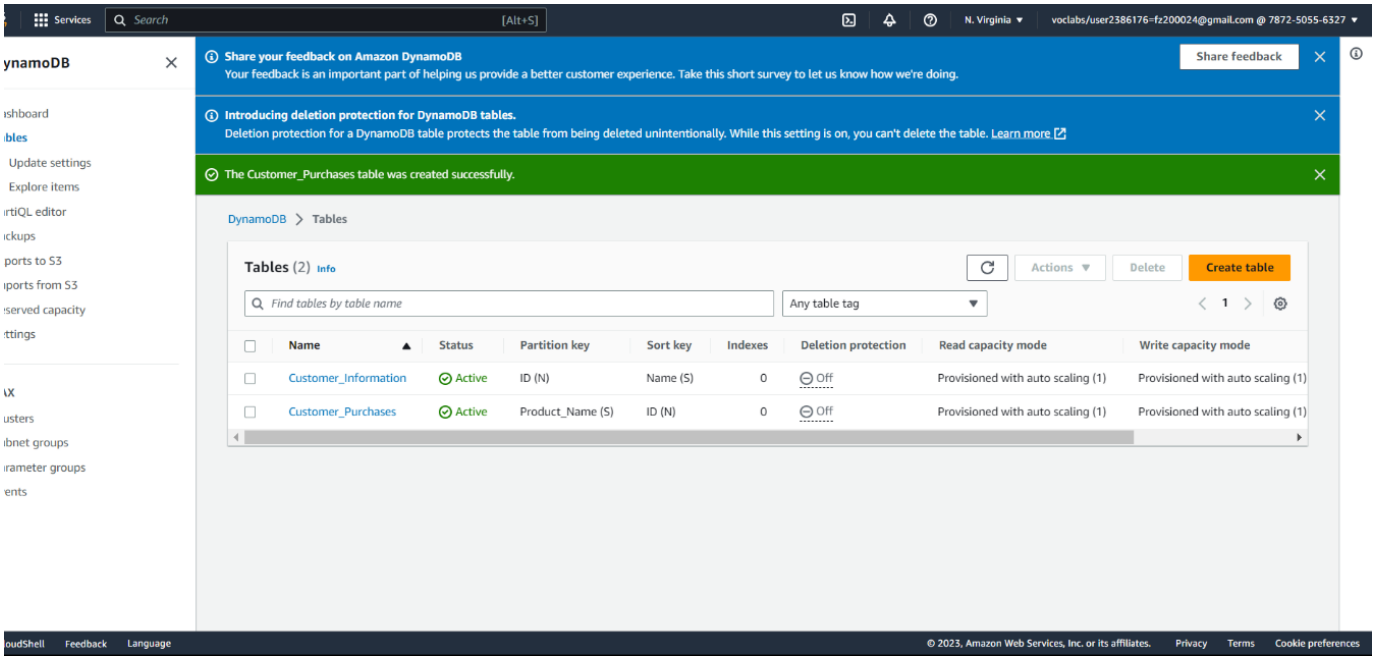


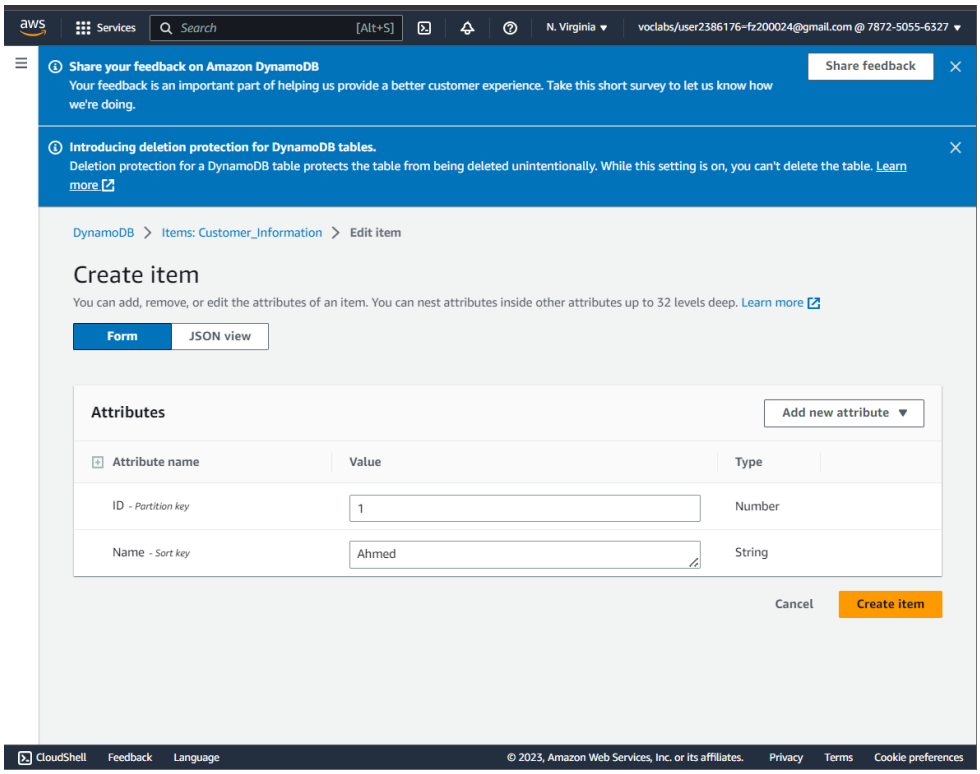
Table settings are customized to have the DynamoDB Standard Table Class, and the table is created.

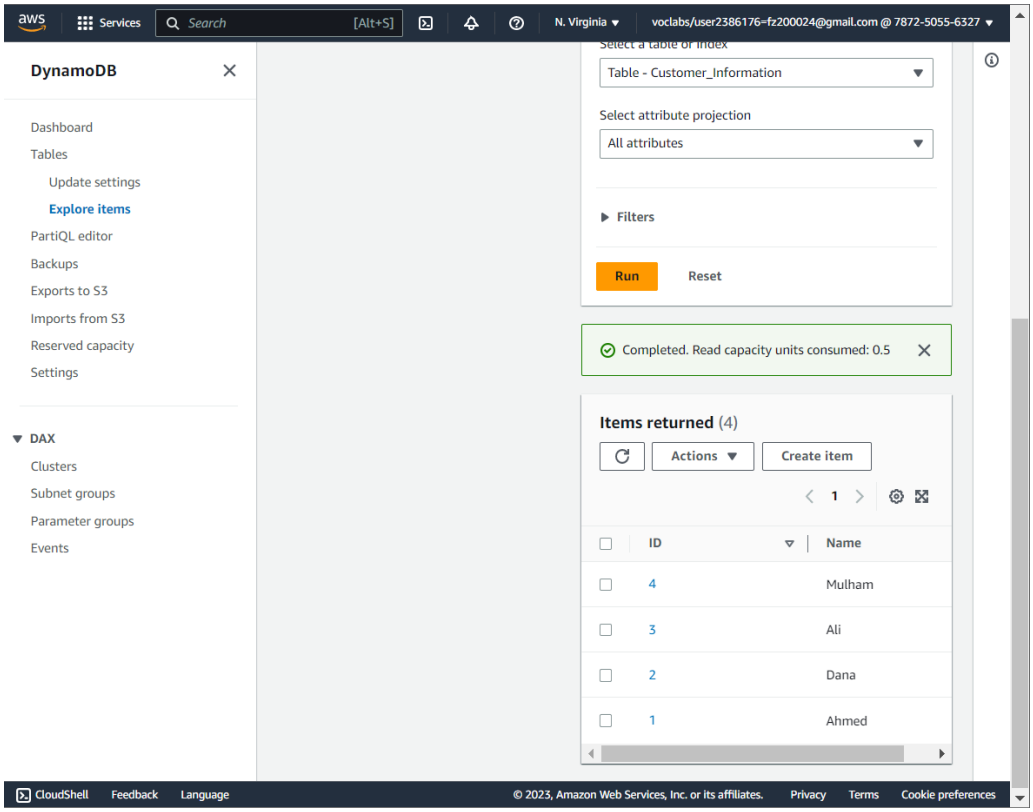


The same steps are followed for the creation of the second table i.e., the “Customer_Purchases” table, with the “Product_Name” of type String used as partition key, and “ID” of type Number used as sort key. Following is the result of the creation of the two DynamoDB tables:

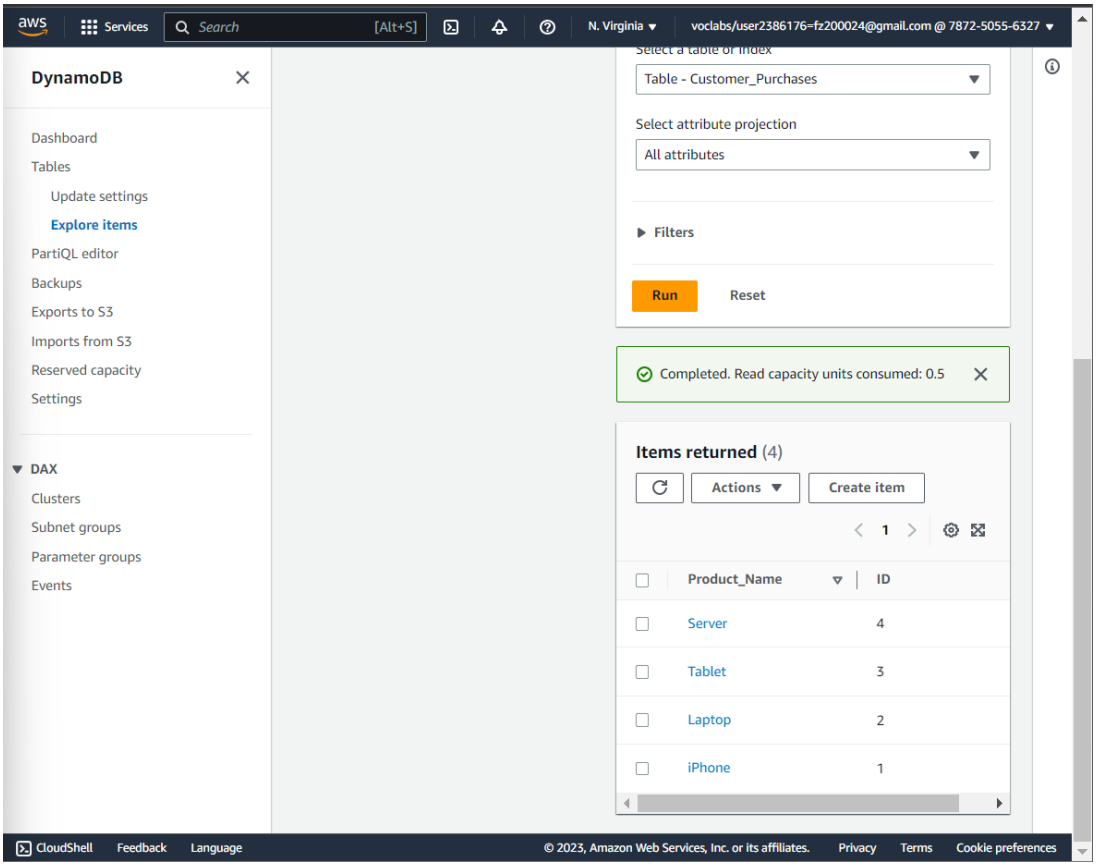


To store the customer data in the DynamoDB tables according to the given schema, the respective items in each database are entered individually. For the Customer_Information table, the items are entered as follows:



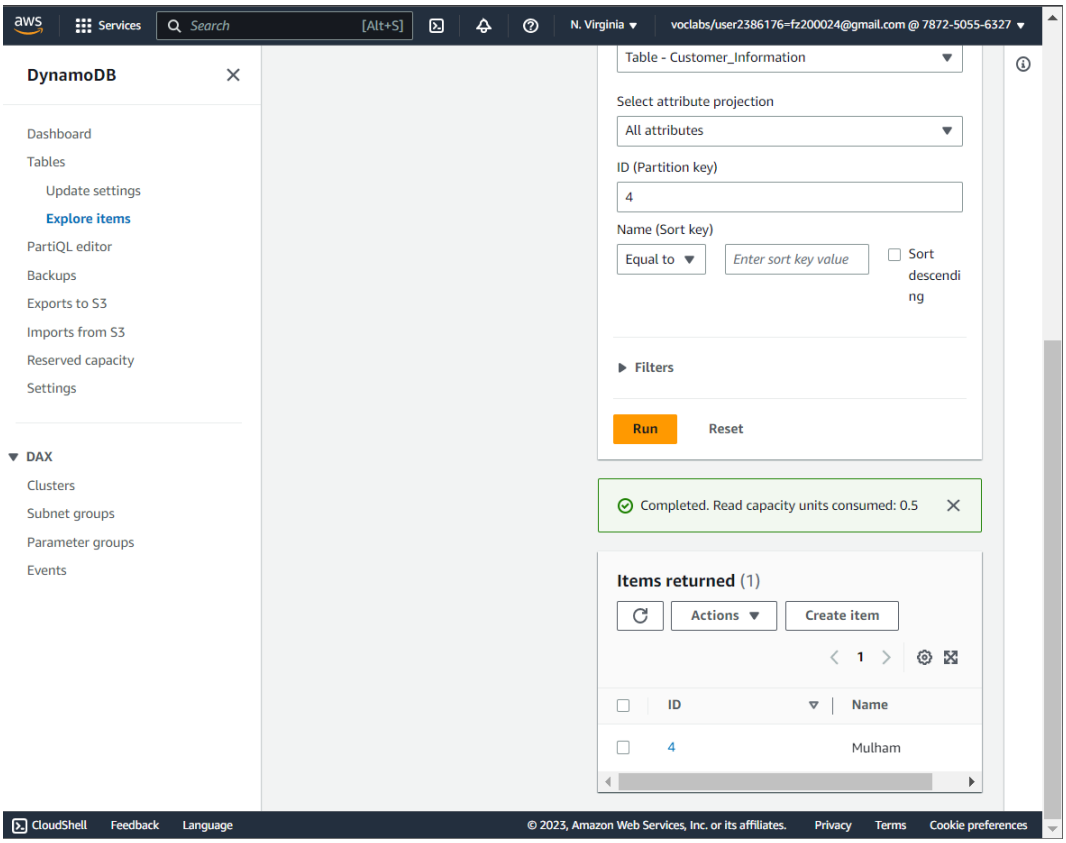


The same is done to add items and store items in the Customer_Purchases table as follows:

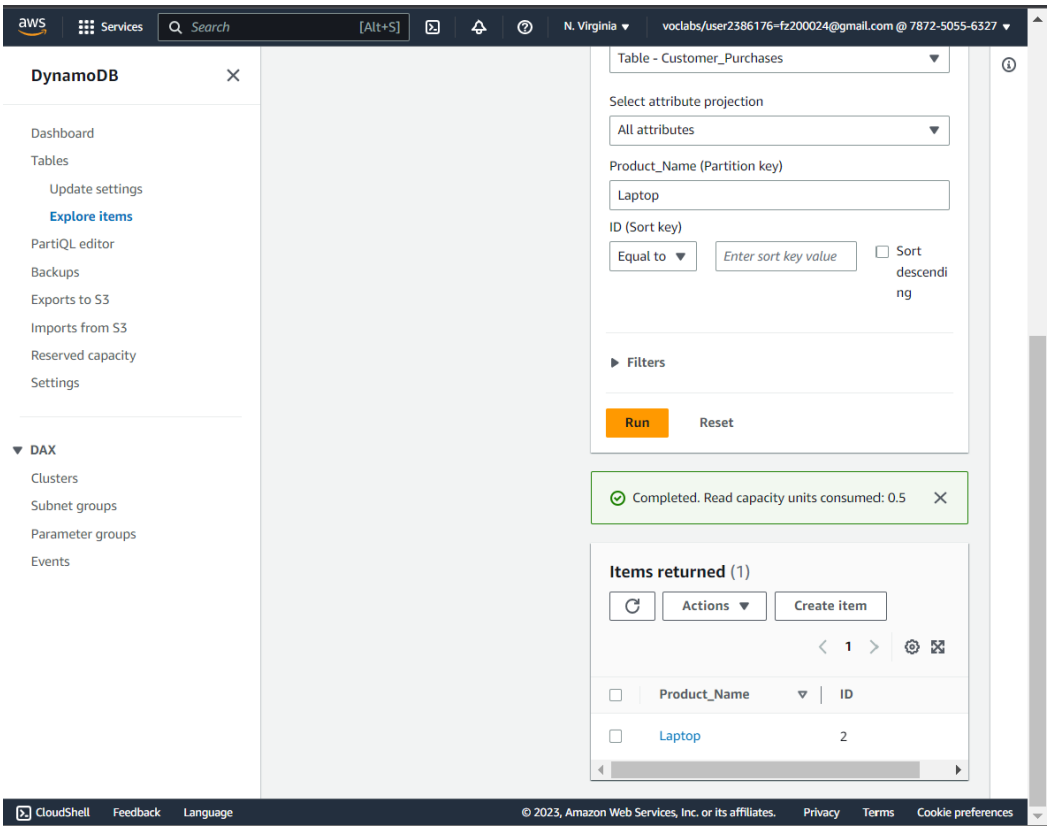


Some sample queries are run on the tables with the following results:

e.g. Item from table Customer_Information with ID 4 is retrieved:



e.g. Item from Customer_Information table with Product_Name “Laptop” is retrieved:

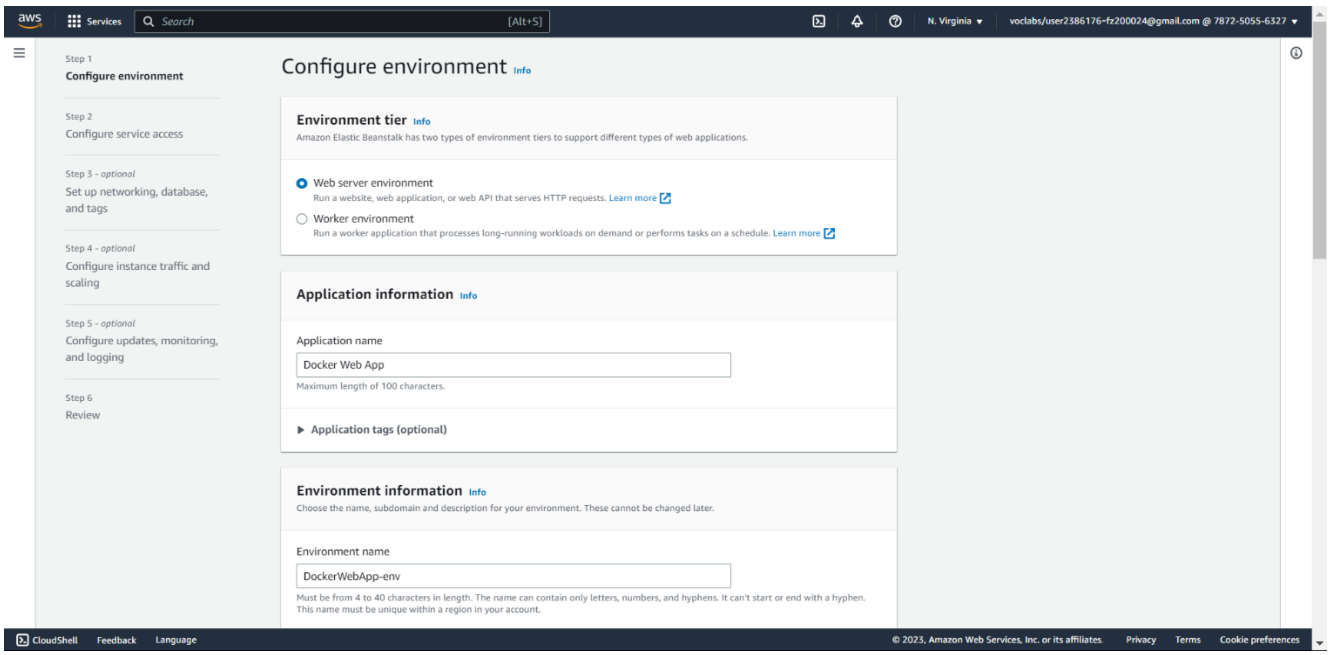


QUESTION: Can you perform a table join in DynamoDB?

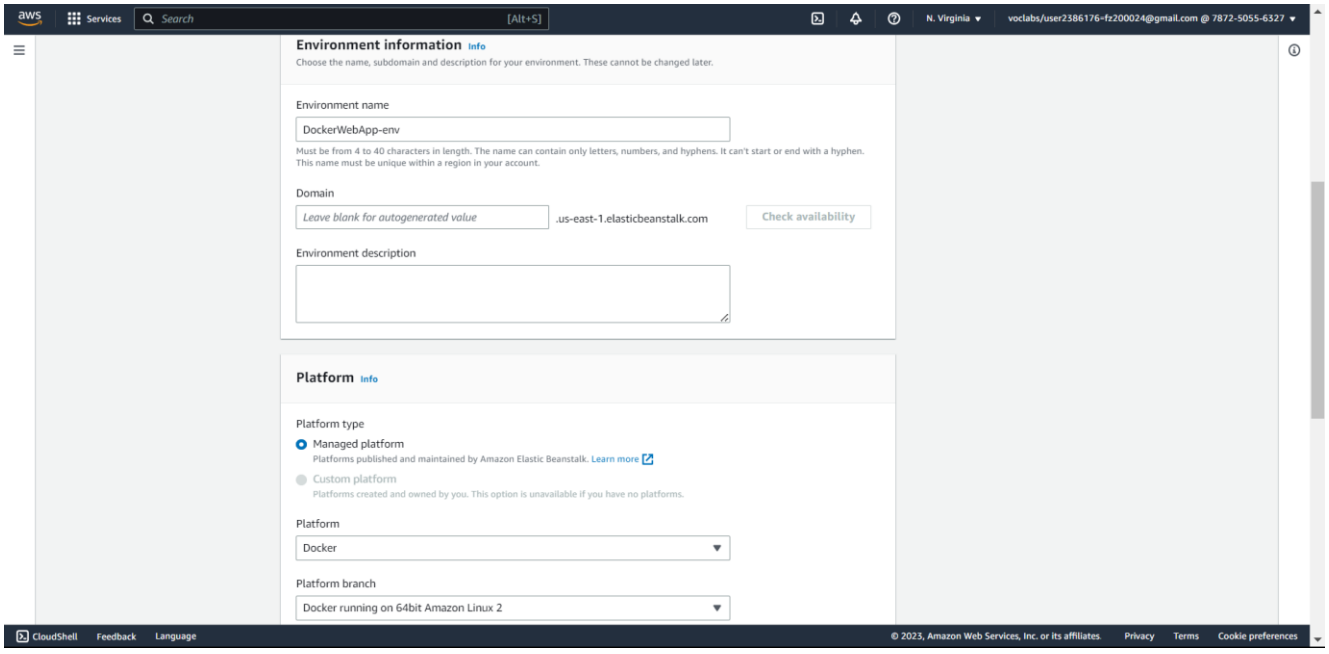
ANSWER: NO, a table join cannot be performed in DynamoDB. DynamoDB is a NoSQL database and unlike the traditional relational databases, there are no relations between tables and it does not support table join operations. It has standalone capabilities and also, no such querying option is made available on the Amazon Web Services Platform.

PART B:

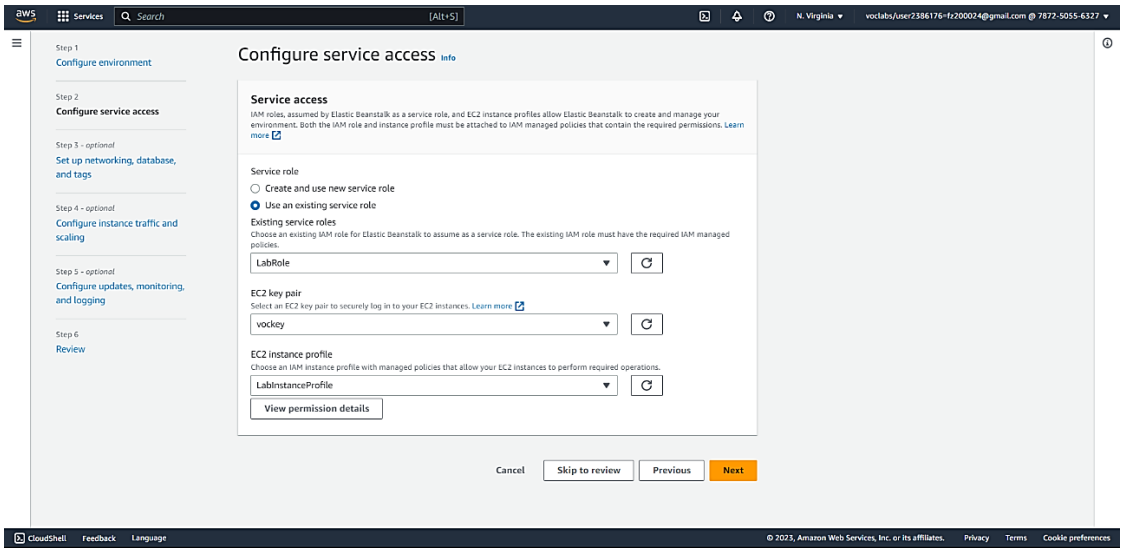
Now to deploy web applications and implement load balancing, web server environments are configured using Elastic Beanstalk to deploy the two web applications. The first one is configured for the “Docker Web App” with the “Web server environment” tier.



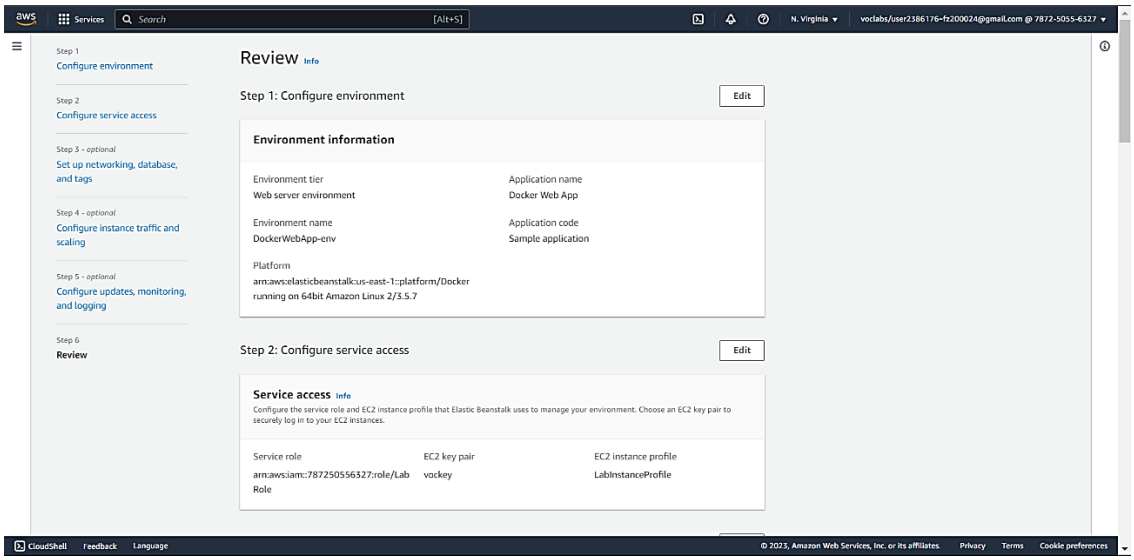
The platform is a Docker platform of type Managed Platform and sample application (shown clearly in upcoming review screenshot).

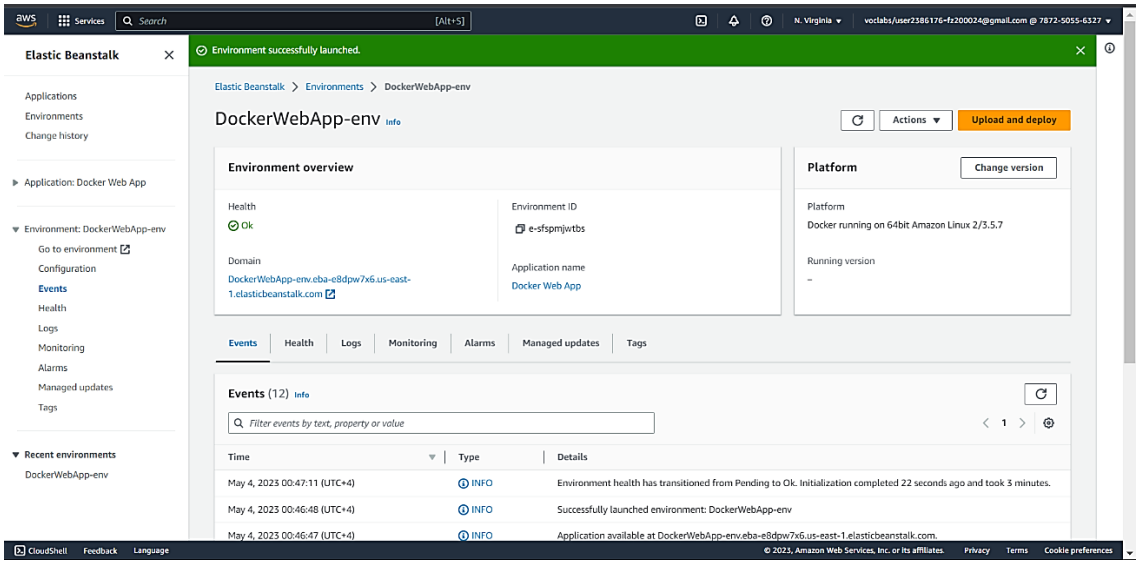


The service access for the Docker app is configured using the existing service role of “LabRole”, with the “vockey” key pair, and EC2 instance profile of “LabInstanceProfile”.

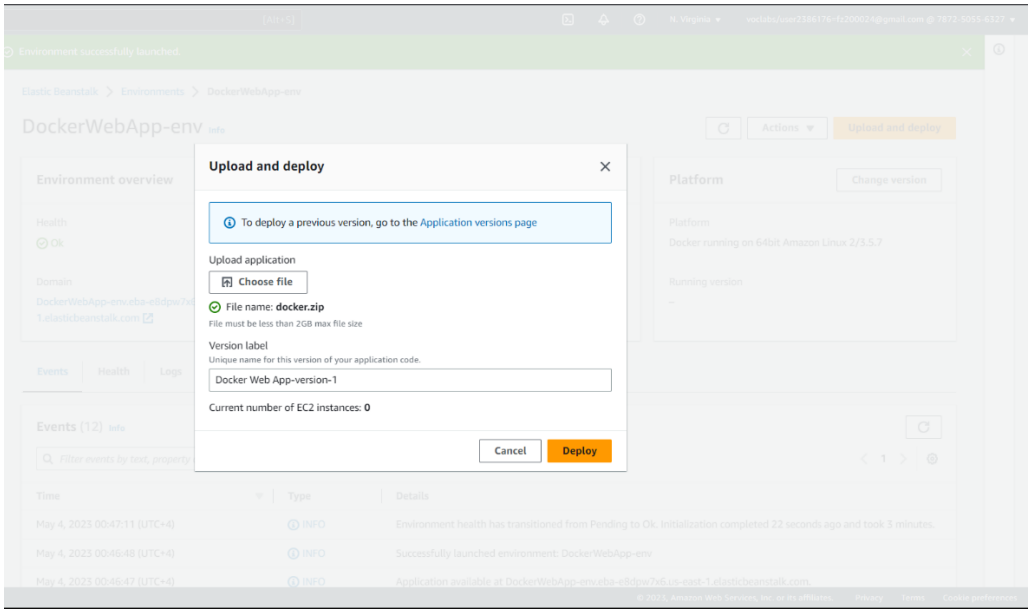


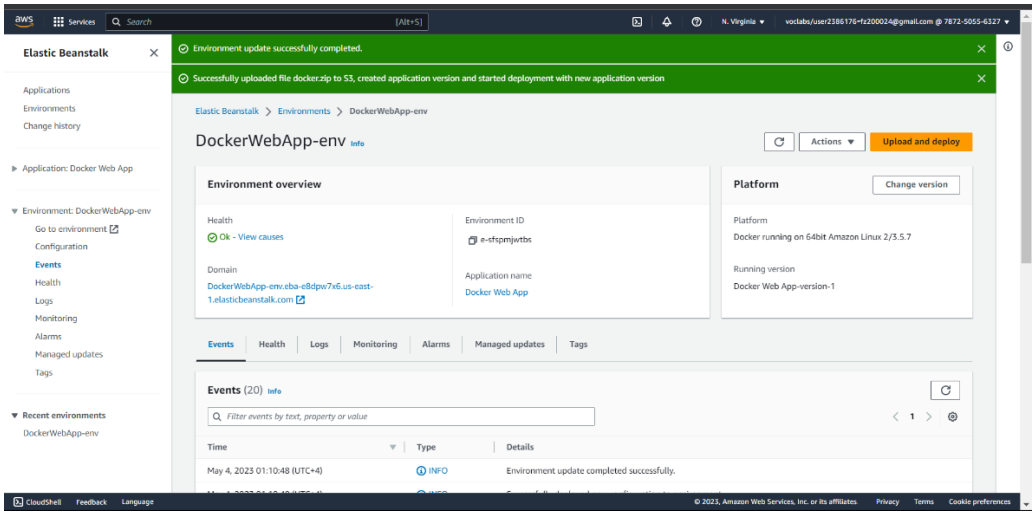
The environment for the Docker Web App has been configured and created.



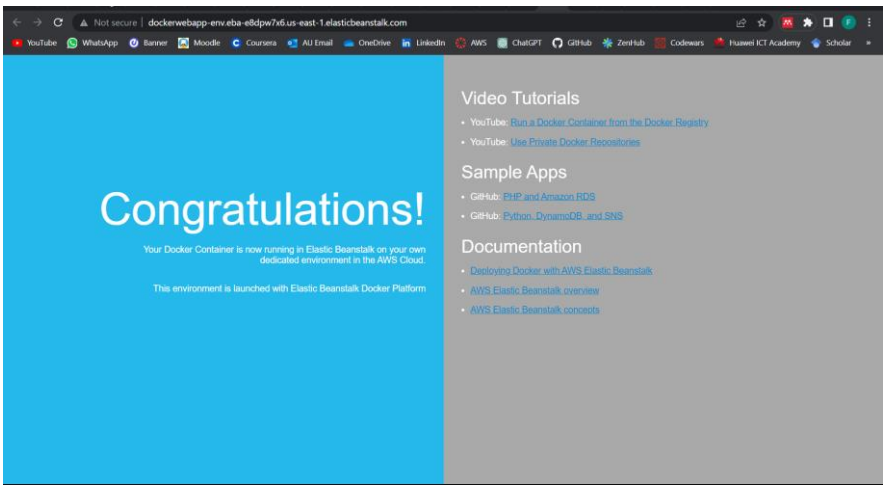


The Docker web application is then uploaded and deployed.

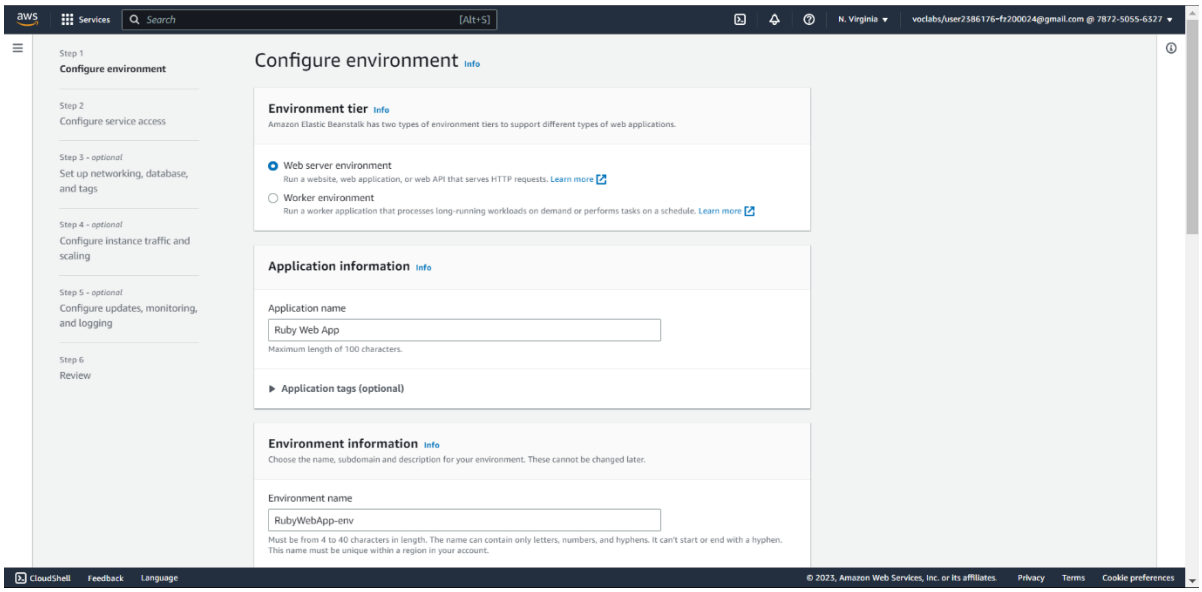


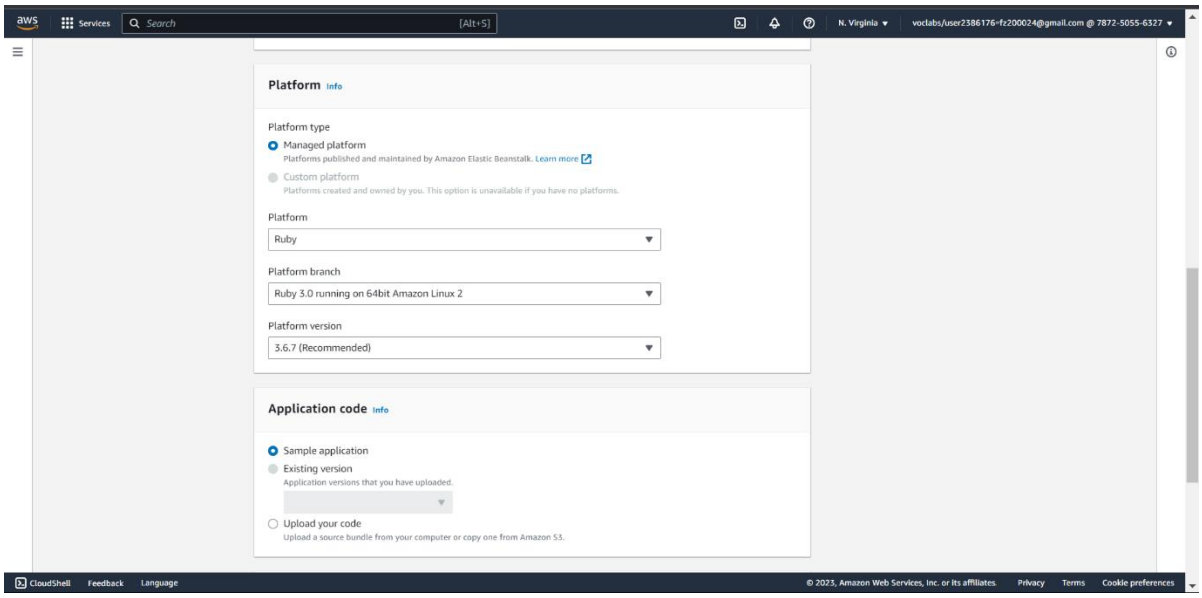


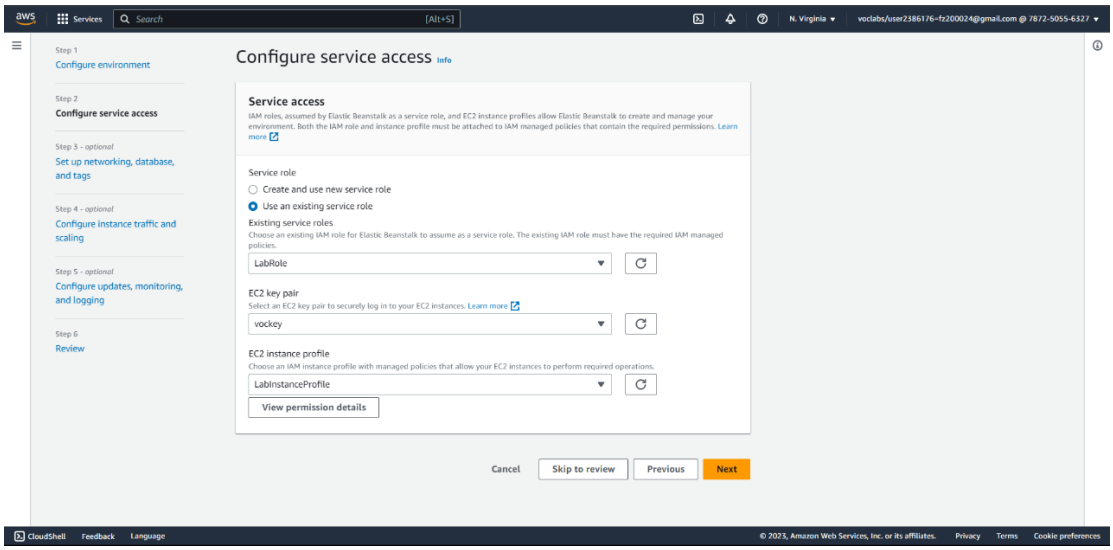
The given domain link is copied and pasted into the search bar to check the working of the deployed web app.

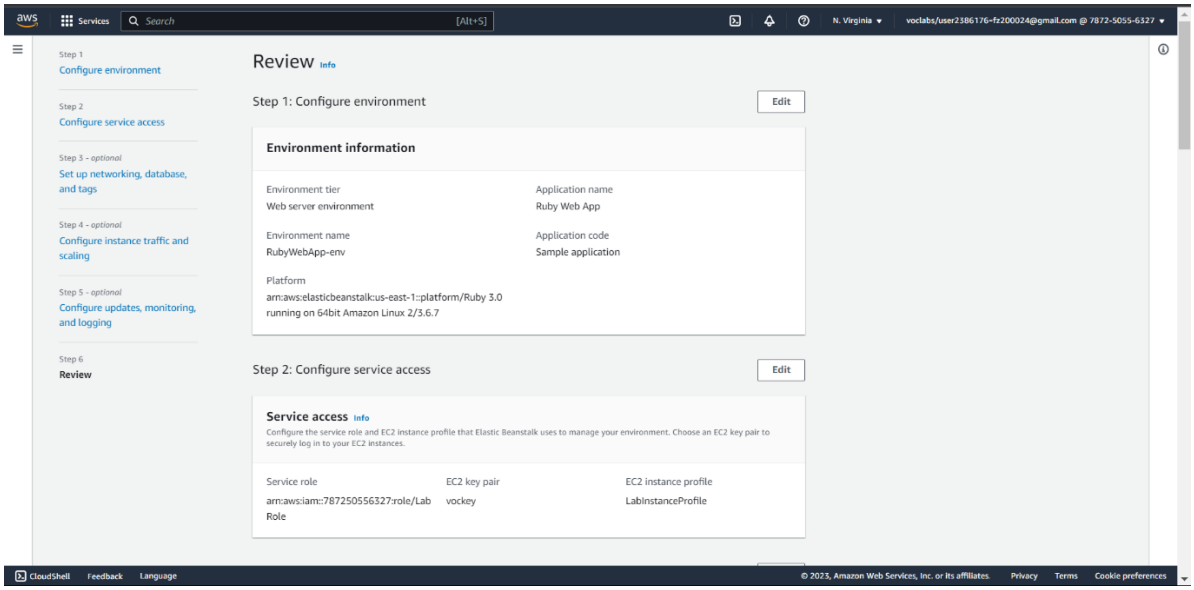


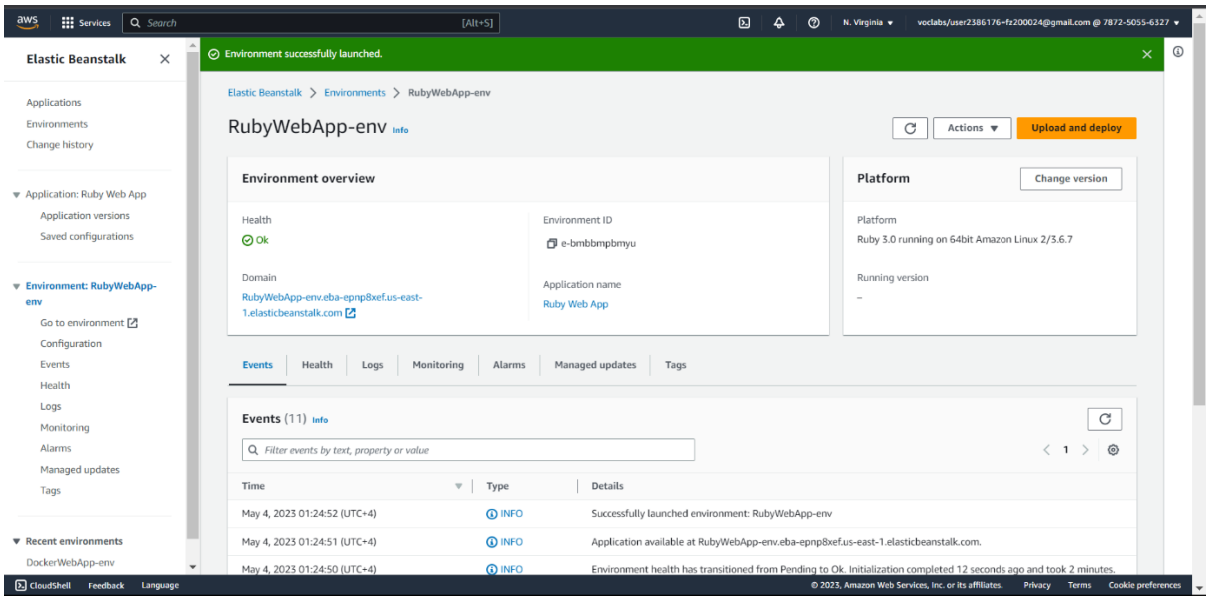
Similar set of steps are followed to configure the web environment and deploy the Ruby web application with the name of “Ruby Web App”.

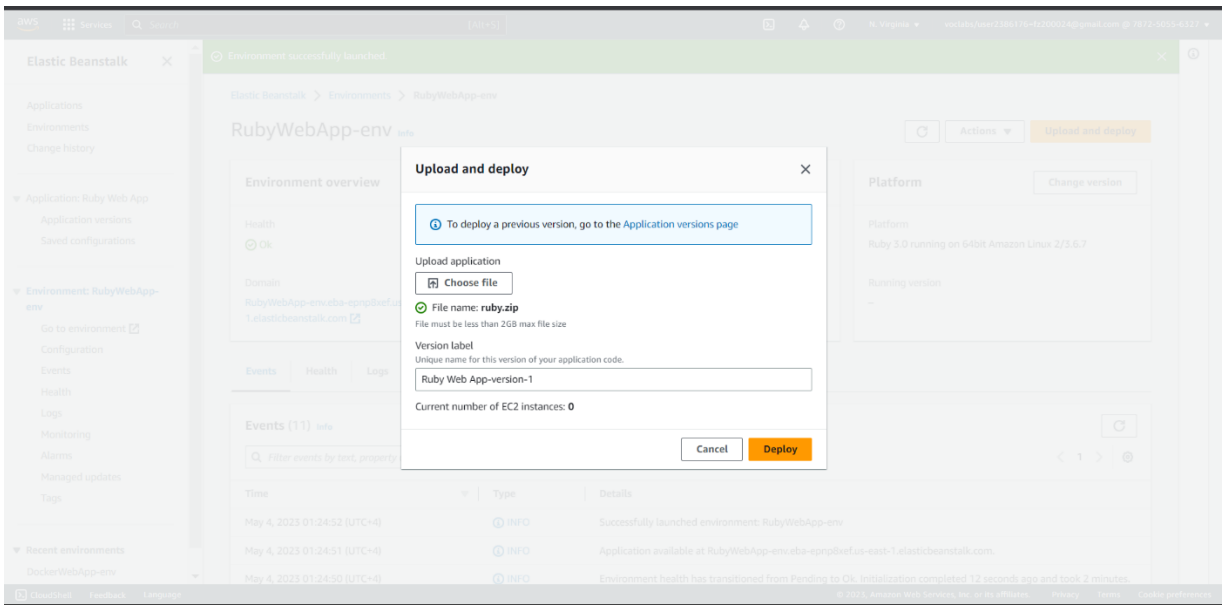


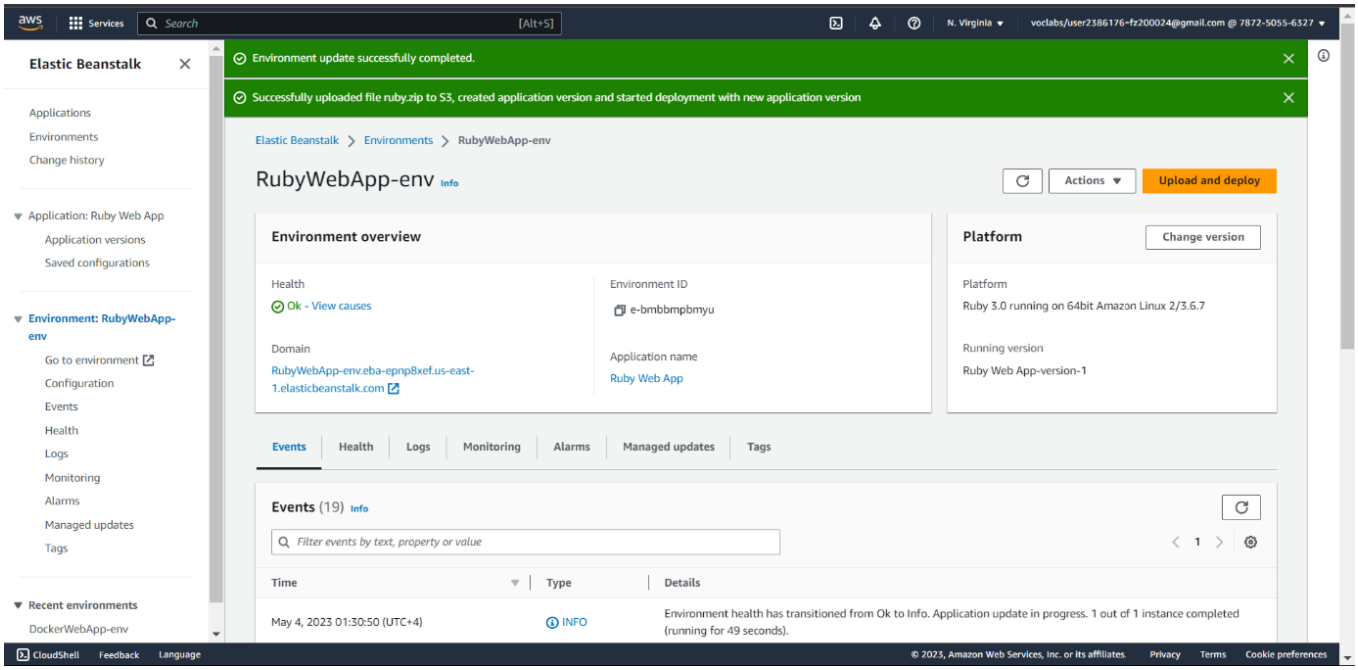




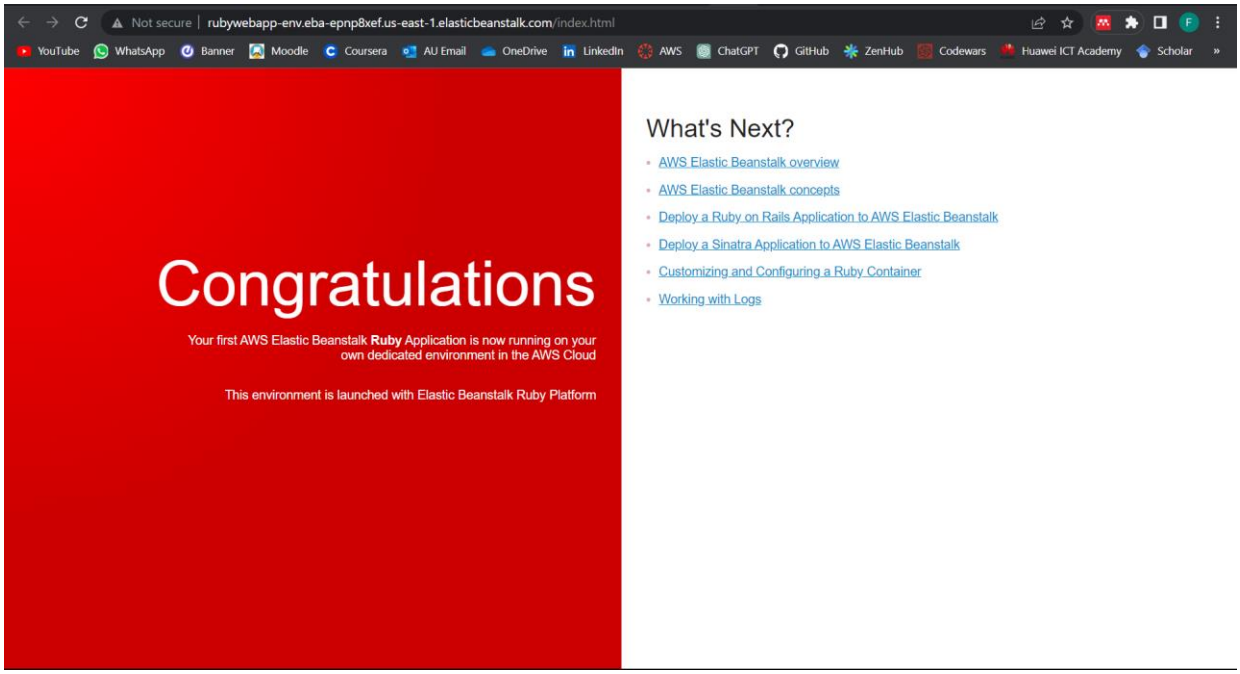

Review Info
Step 1: Configure environment
Edit
Environment information
Environment tier
Web server environment
Application name
Ruby Web App
Environment name
RubyWebApp-env
Application code
Sample application
Platform
arn:aws:elasticbeanstalk:us-east-1::platform/Ruby 3.0 running on 64bit Amazon Linux 2/3.6.7
Step 2: Configure service access
Edit
Service access Info
Service role
arn:aws:iam::787250556327:role/LabRole
EC2 key pair
vockey
EC2 instance profile
LabInstanceProfile


Environment successfully launched.
Elastic Beanstalk > Environments > RubyWebApp-env
RubyWebApp-env Info
Actions
Upload and deploy
Environment overview
Health
Ok
Environment ID
e-bmbbmpbmyu
Domain
RubyWebApp-env.eba-epnp8xef.us-east-1.elasticbeanstalk.com
Application name
Ruby Web App
Platform
Change version
Platform
Ruby 3.0 running on 64bit Amazon Linux 2/3.6.7
Running version
Events
Health
Logs
Monitoring
Alarms
Managed updates
Tags
Events (11) Info
Time
Type
Details
May 4, 2023 01:24:52 (UTC+4)
INFO
Successfully launched environment: RubyWebApp-env
May 4, 2023 01:24:51 (UTC+4)
INFO
Application available at RubyWebApp-env.eba-epnp8xef.us-east-1.elasticbeanstalk.com.
May 4, 2023 01:24:50 (UTC+4)
INFO
Environment health has transitioned from Pending to Ok. Initialization completed 12 seconds ago and took 2 minutes.

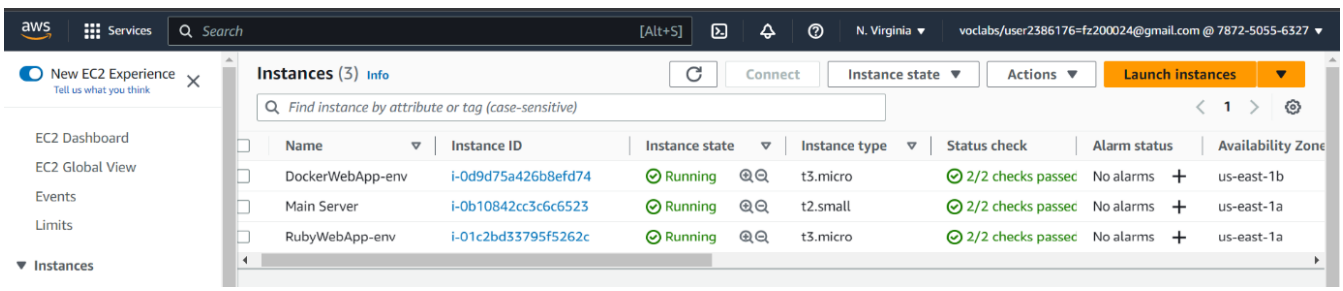

Upload and deploy
To deploy a previous version, go to the Application versions page
Upload application
Choose file
File name: ruby.zip
File must be less than 2GB max file size
Version label
Unique name for this version of your application code.
Ruby Web App-version-1
Current number of EC2 instances: 0
Cancel
Deploy



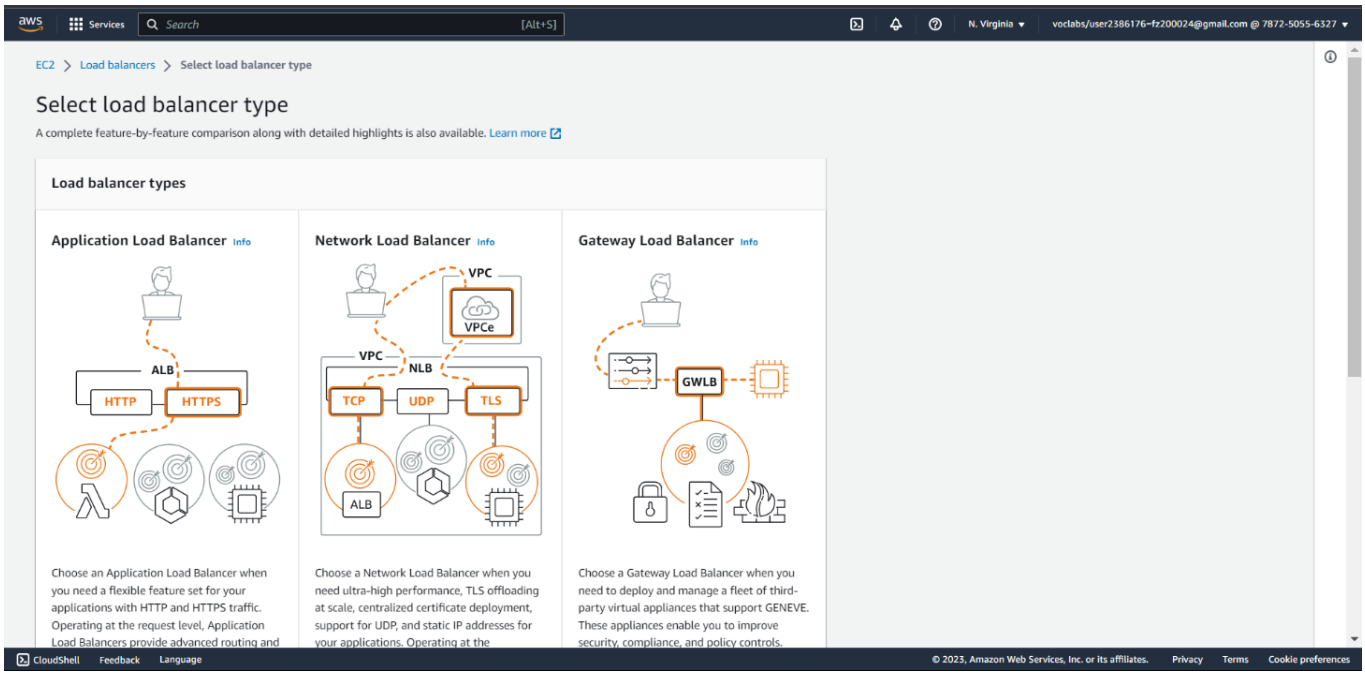
The given domain link is copied and pasted into the search bar to check the working of the deployed web app.

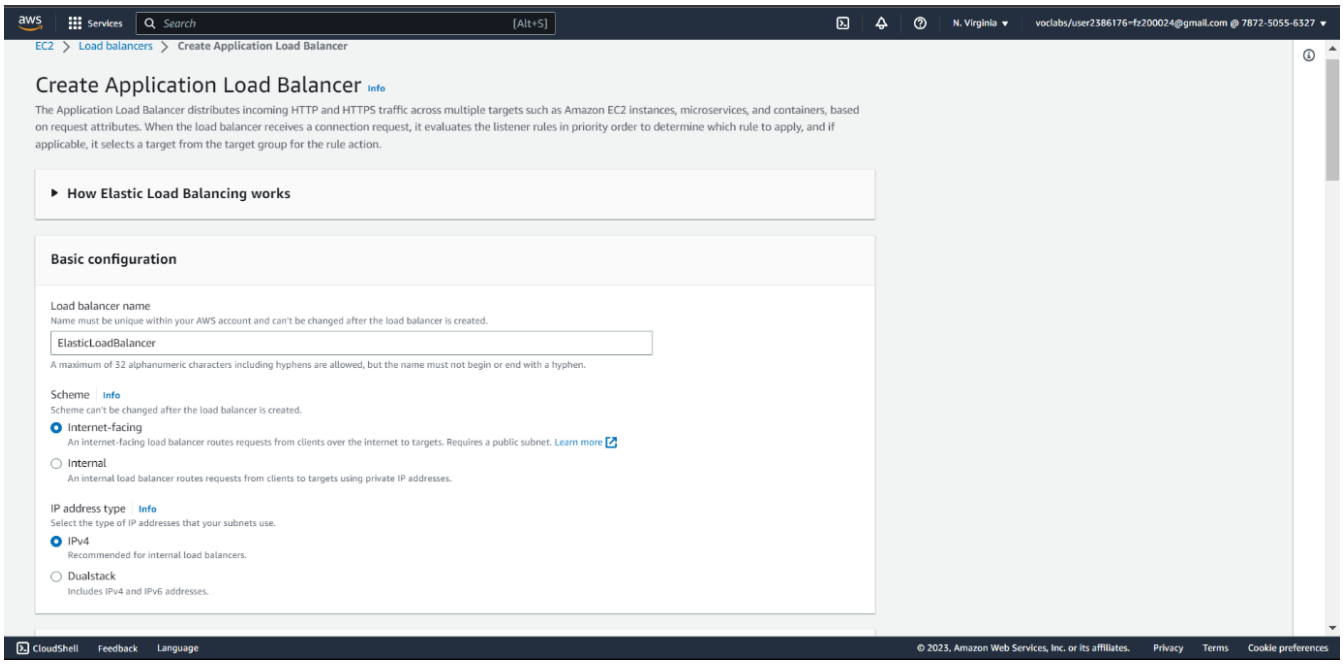


As a result of deploying the two web applications using Elastic Beanstalk, two new EC2 instances are also created with them:

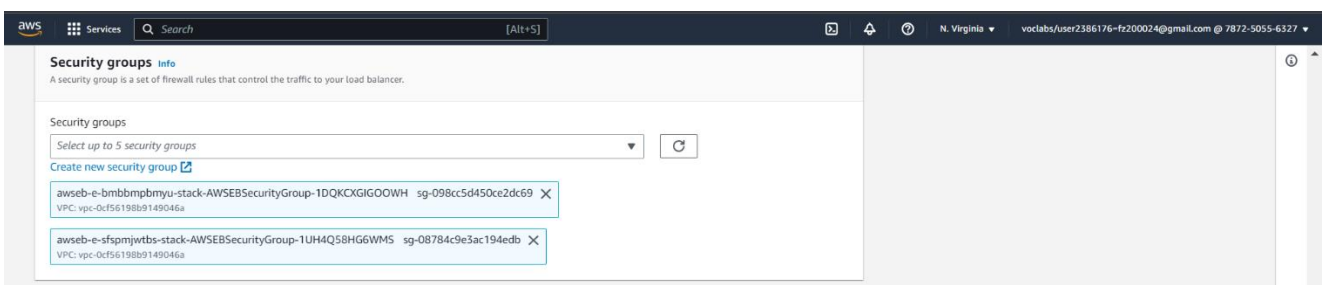


Now a load balancer of name “ElasticLoadBalancer” is created to load balance between the two deployed web applications. An application load balancer is selected to fulfil this purpose.

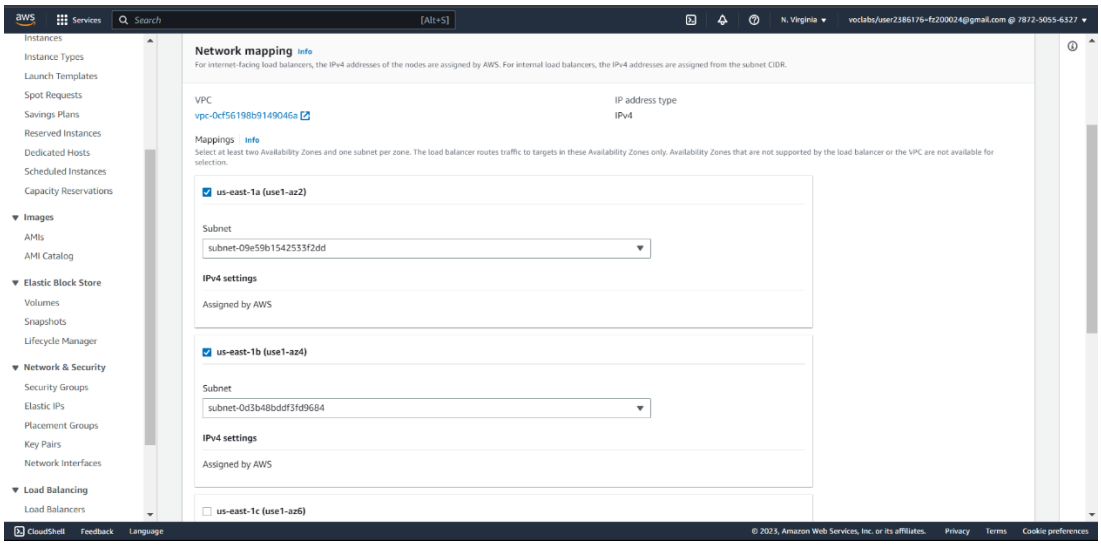




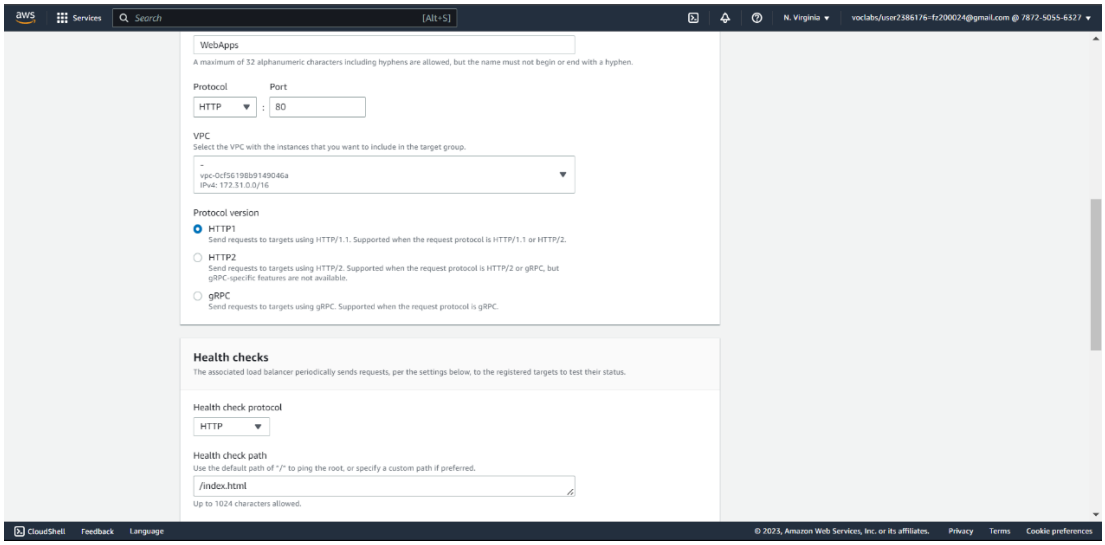
The security groups are selected as the ones that were created with the creation of the web applications.



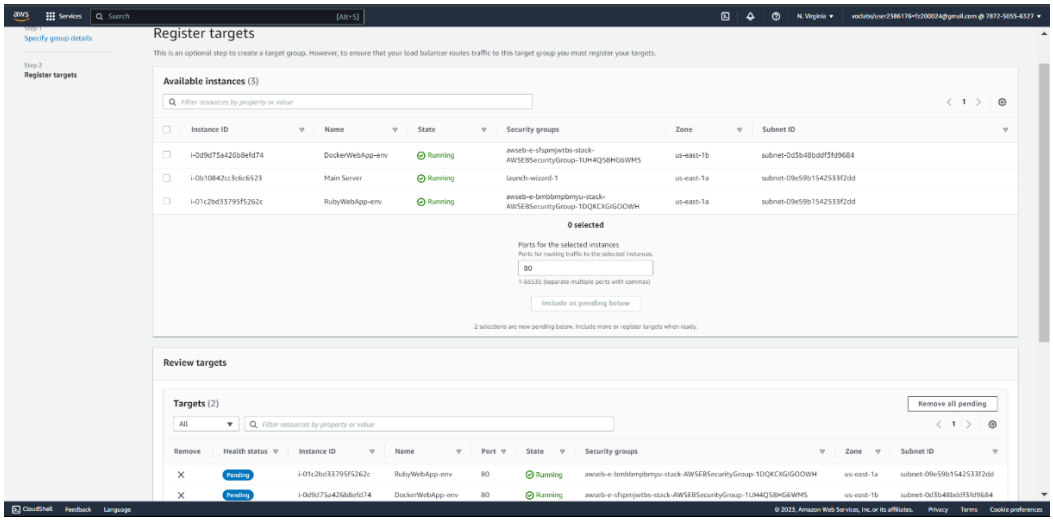
The required subnets i.e. us-east-1a and us-east-1b are specified for load balancing.



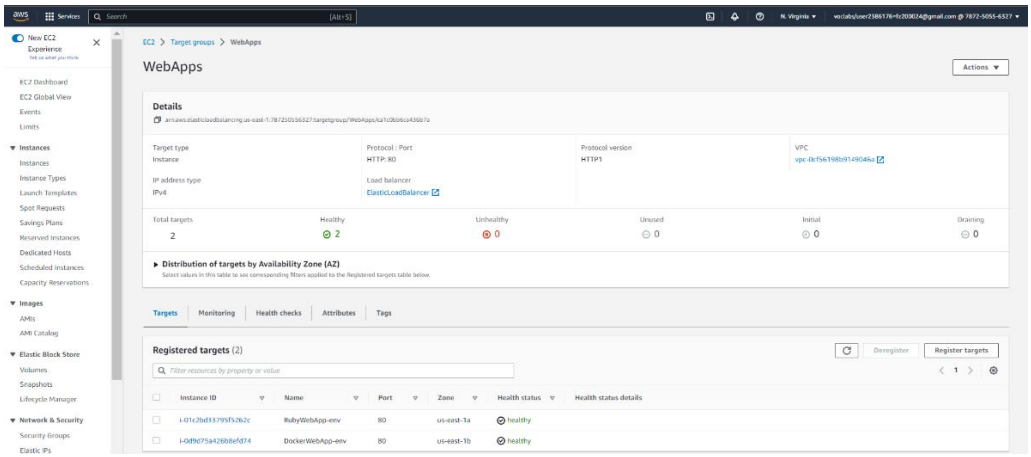
A target group by the name of “WebApps” is created to be used for the load balancer. HTTP is chosen as the protocol for target health check with the health check path as “/index.html”.



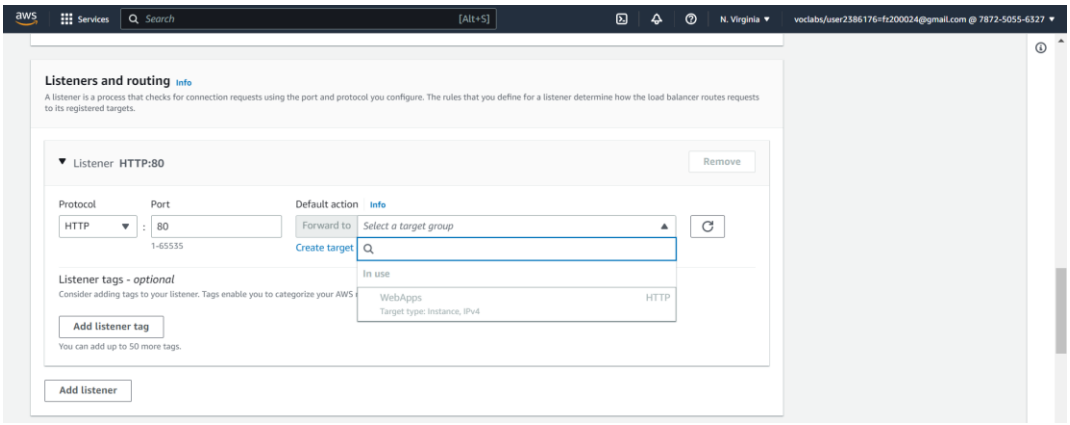
The two instances for the deployed web applications are registered in the target group.



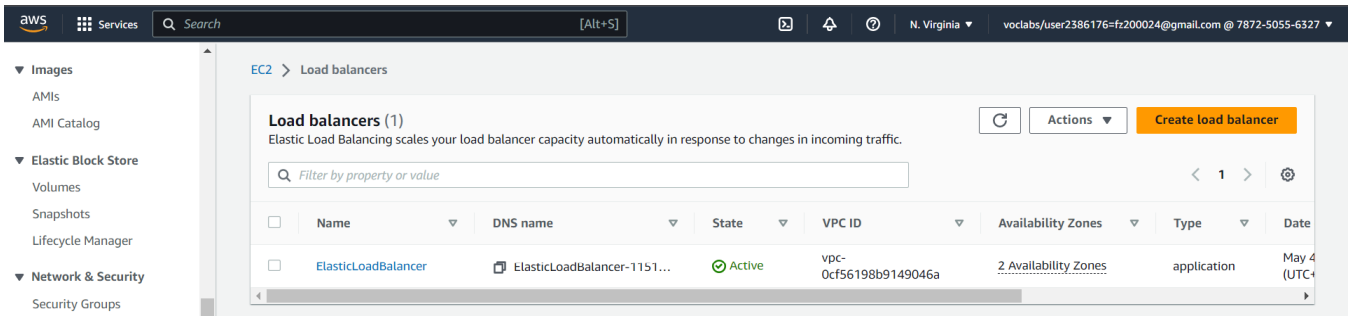
The “WebApps” target group has been created.

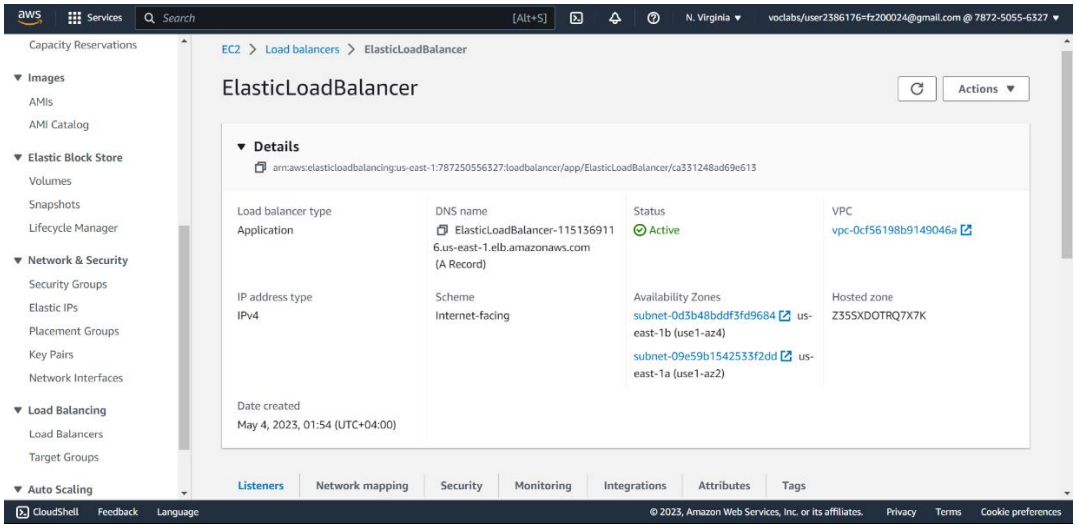


The WebApps target group is selected for the load balancer. (I forgot to screenshot this step at the time of actually adding the target group, so I added one screenshot for demonstration below, hence the target group in the dropdown list is greyed out).

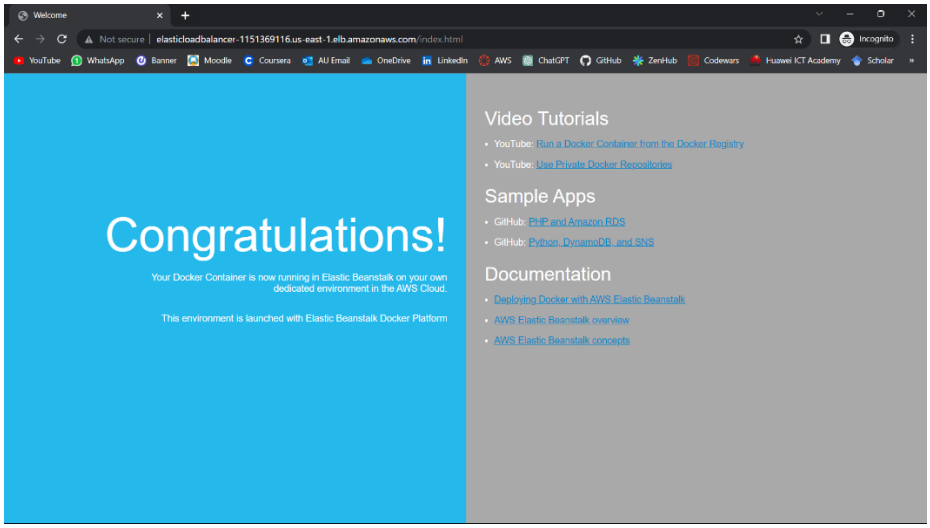


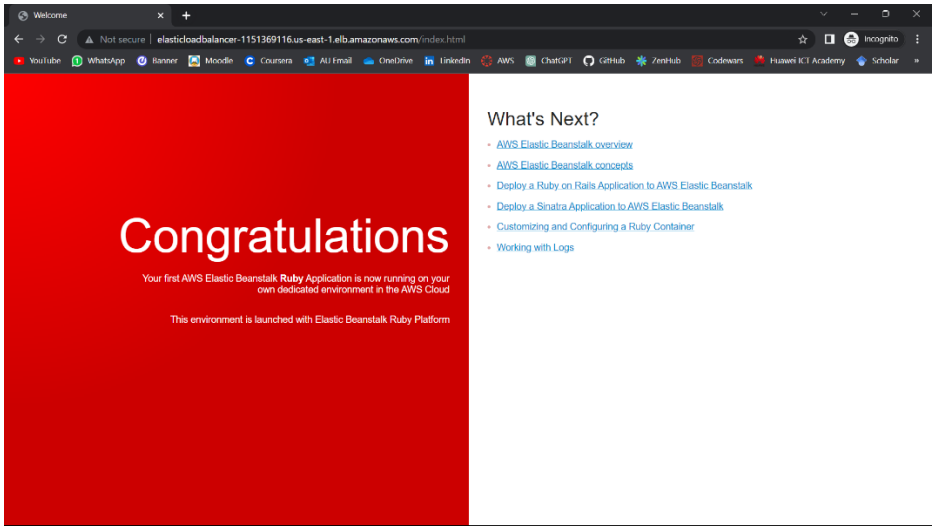
The load balancer has been created after making the required configurations.





The DNS name link is copied and pasted into the search bar to check the working of the load balancer. Upon refreshing, the screen shows the webpage changing between the docker web app and the ruby web app.





The required cloud architecture has been designed and completed.