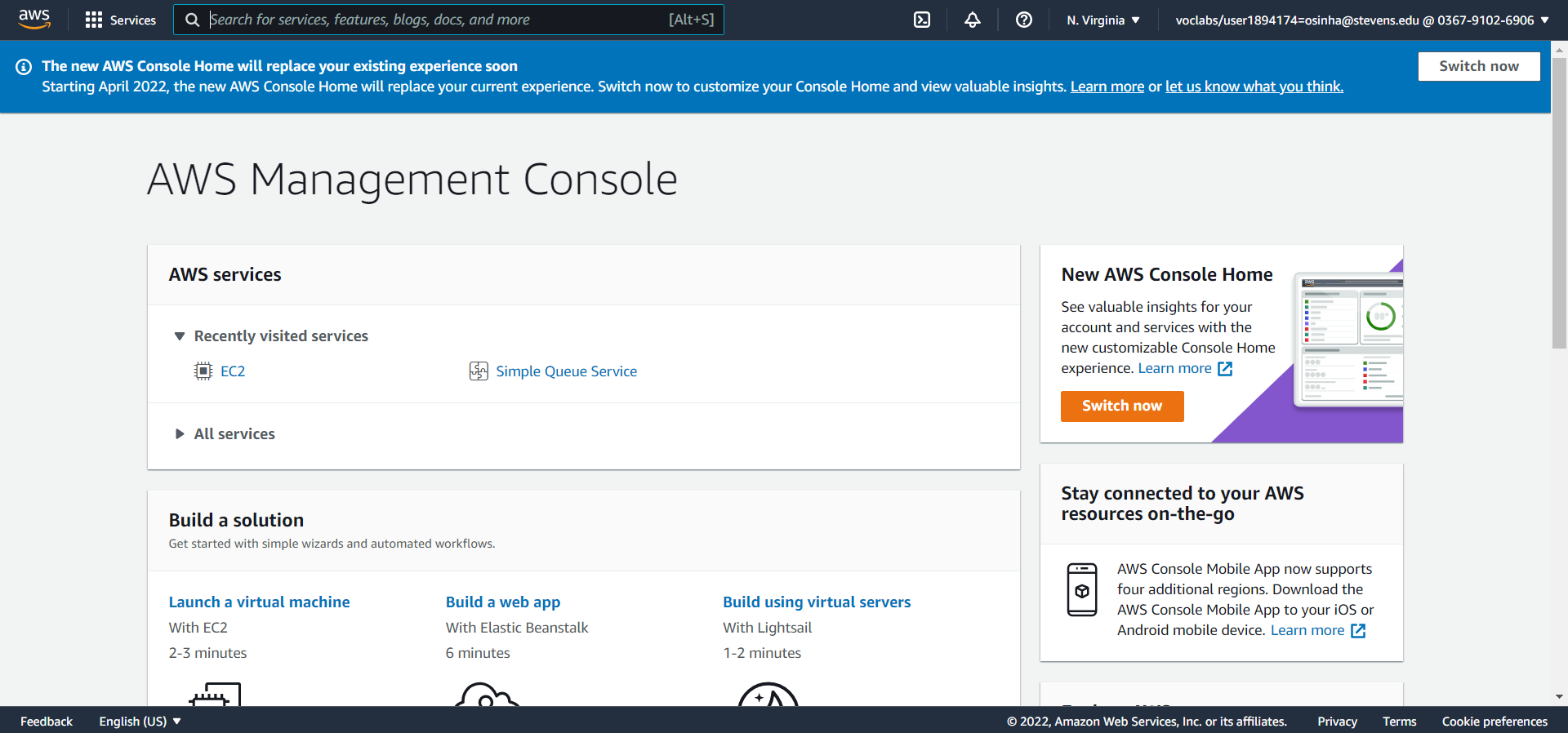
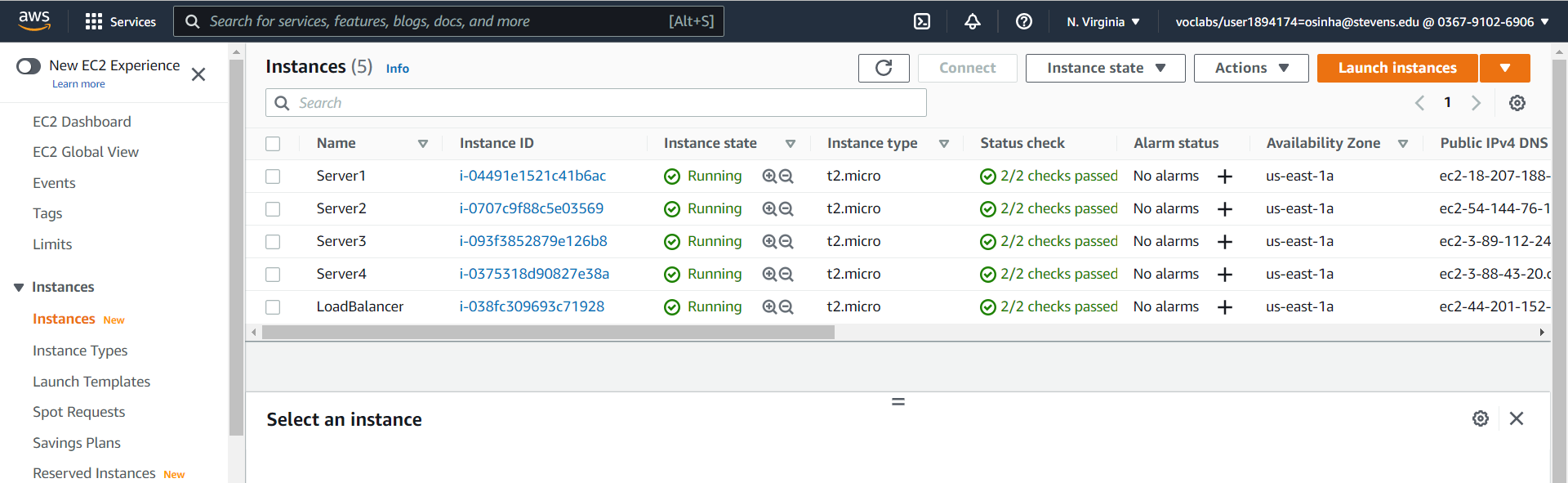
LAB – 2

**Steps used :**

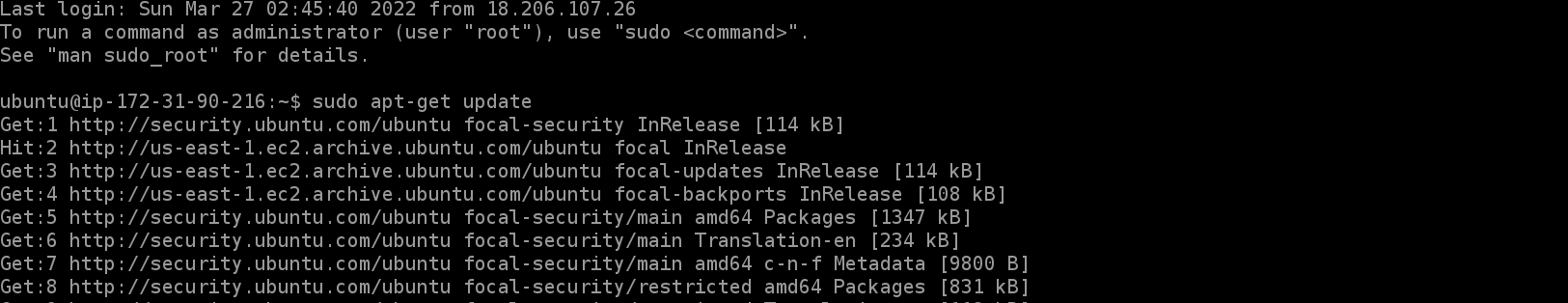
1. This is screen we encounter after starting lab:



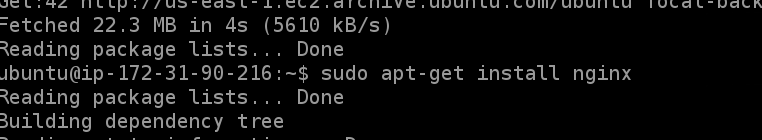
1. We create 5 “Ubuntu Server 18.04 LTS (HVM), SSD Volume Type” EC2 instances:



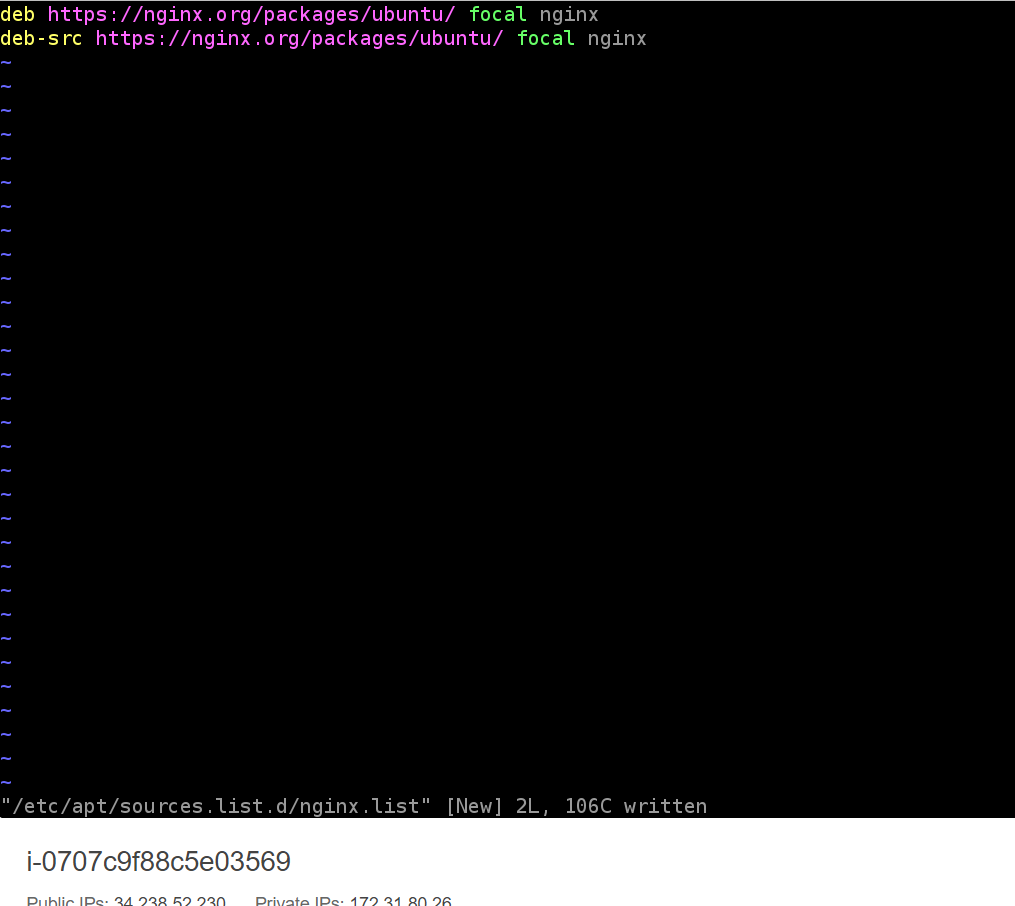
1. We install nginx on all 5 instances:
2. sudo apt-get update



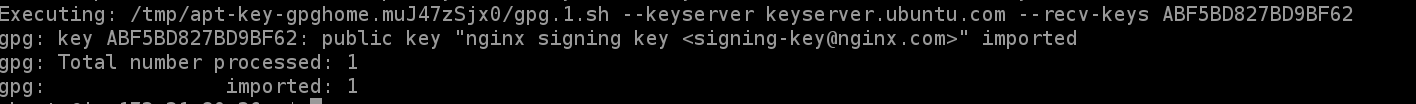
(ii) sudo apt-get install nginx



(iii) Adding required commands to */etc/apt/sources.list.d/nginx.list* file



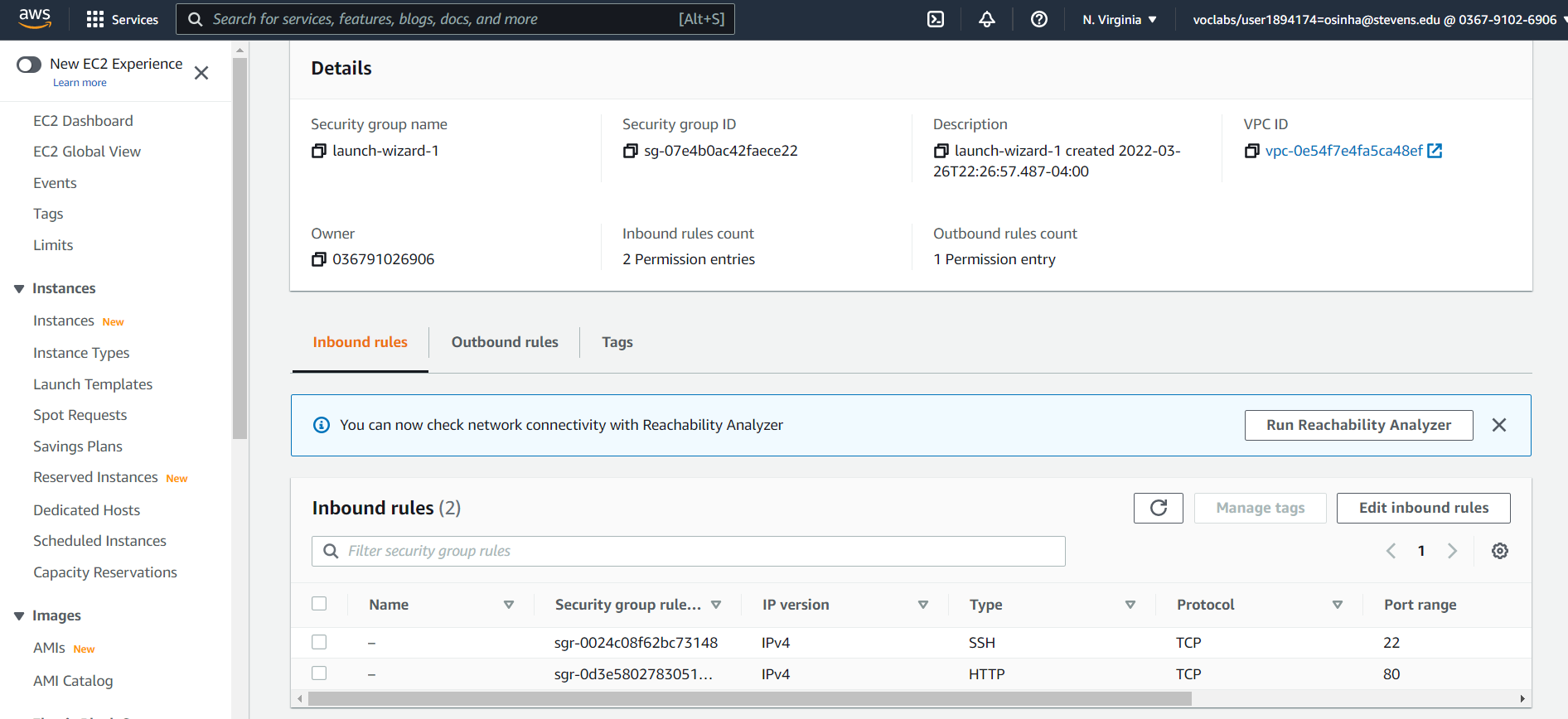
(iv) Dealing with key error:



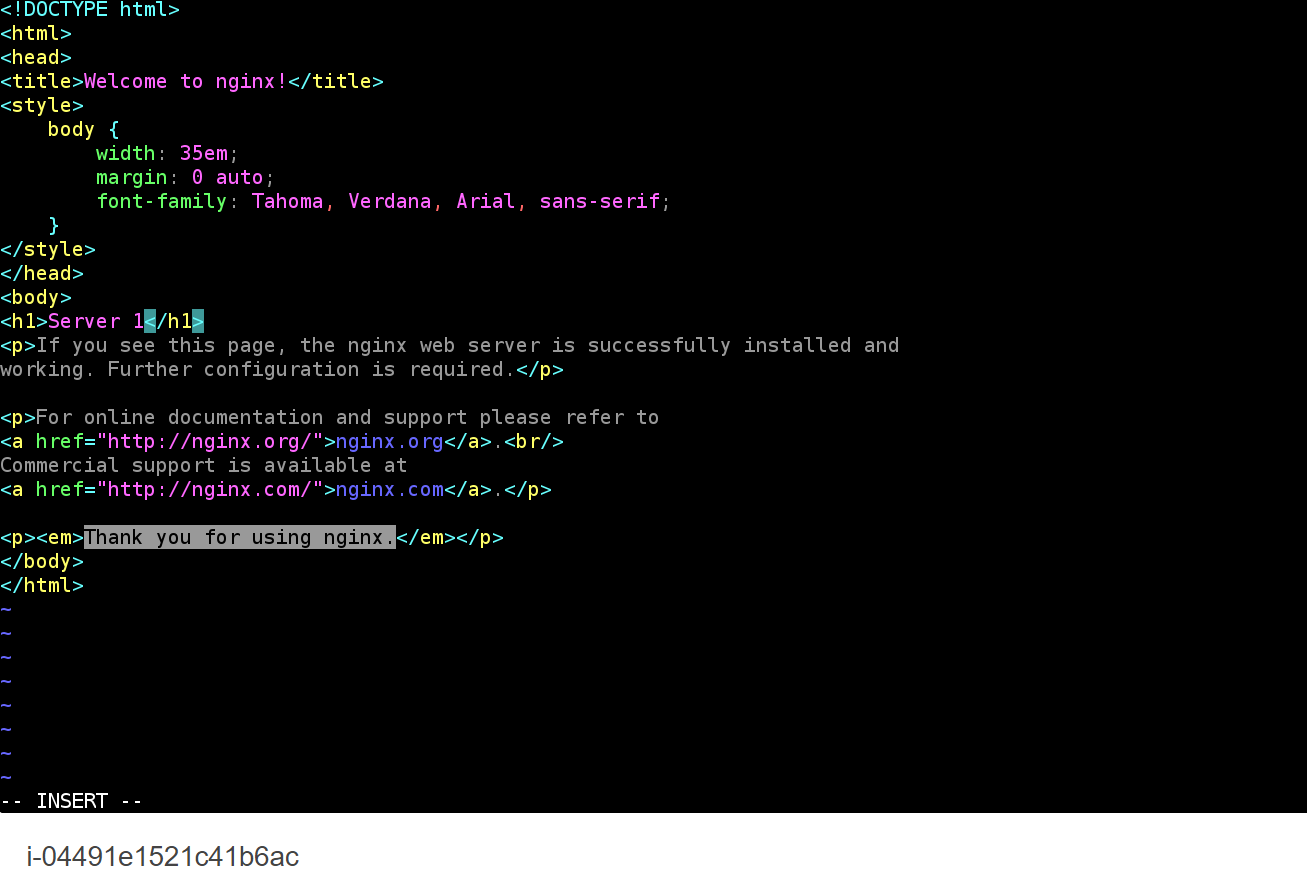
(v) sudo systemctl start nginx



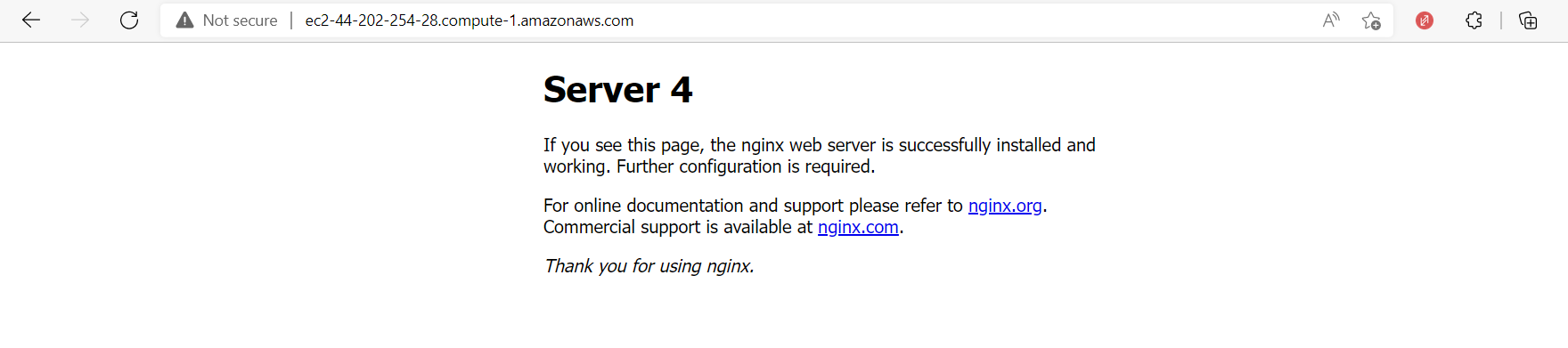
(vii) Adding http to inbound rules of group security:



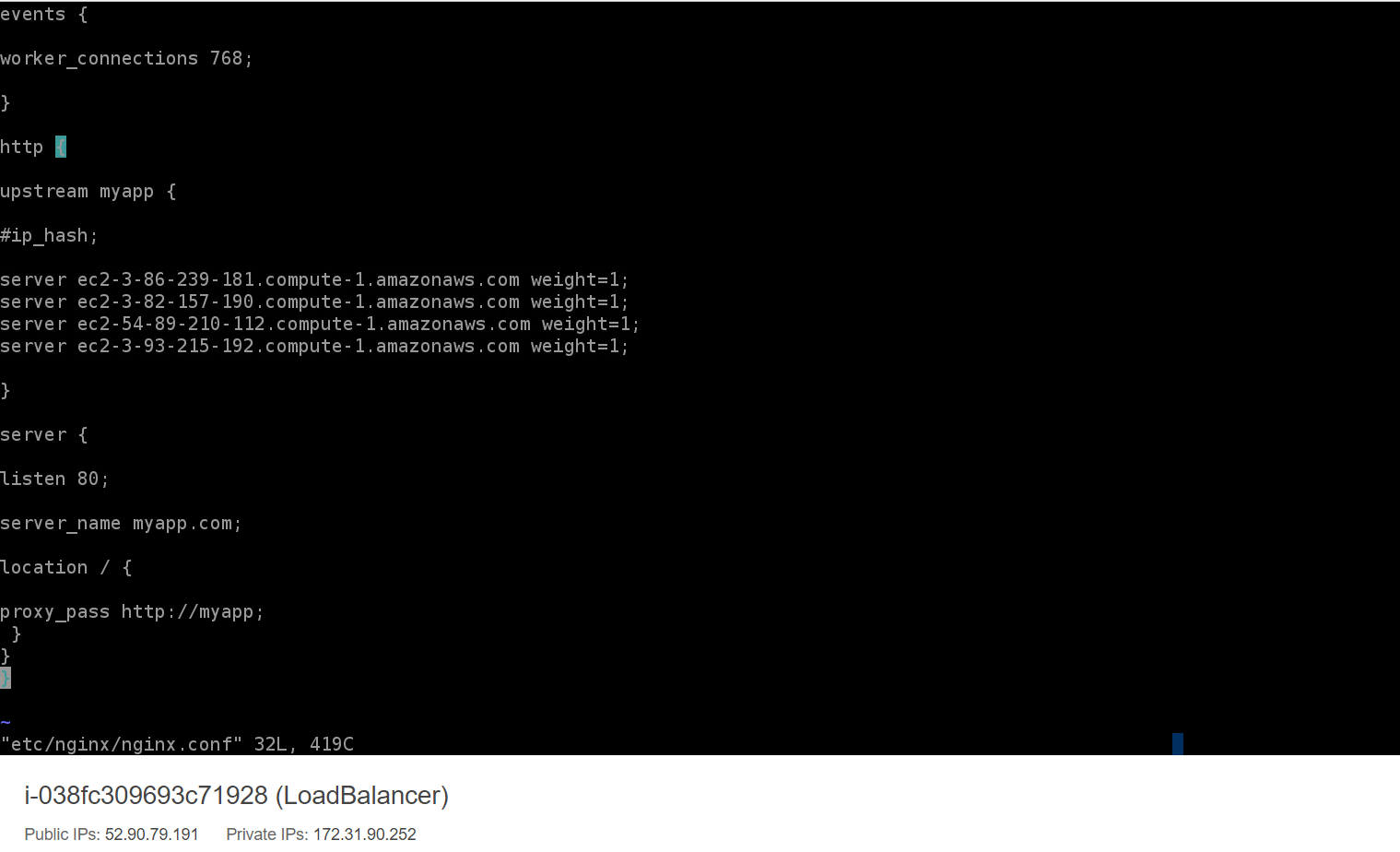
(vii) Add header Server ID in */usr/share/nginx/html/index.html*:



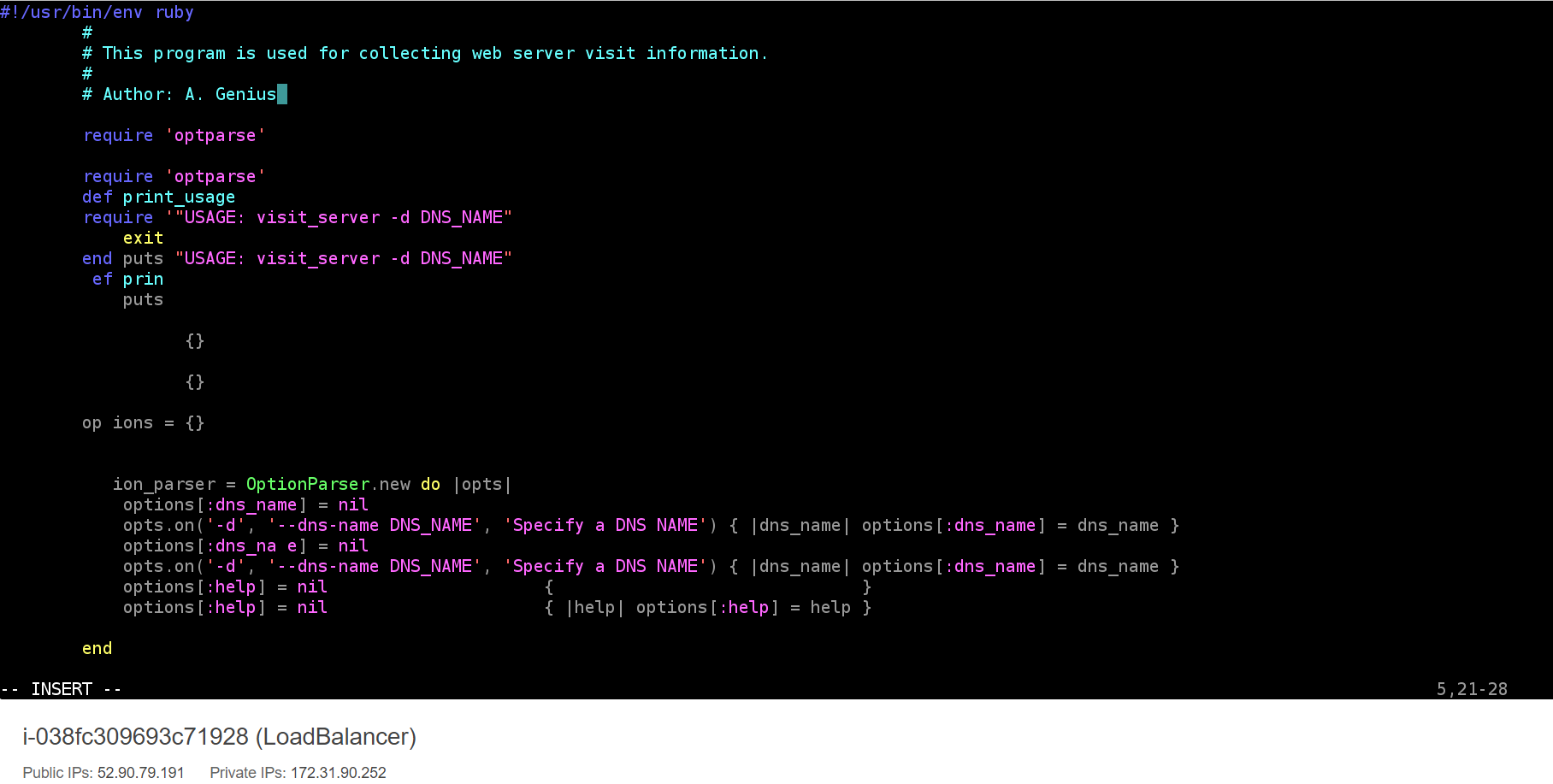
1. We open the public DNS on the instance to verify that NGINX is working:



1. We configure the load balancer by editing etc/nginx/nginx.conf file:



1. We now create a tool “visit\_server” to make 2000 requests to load balancer:



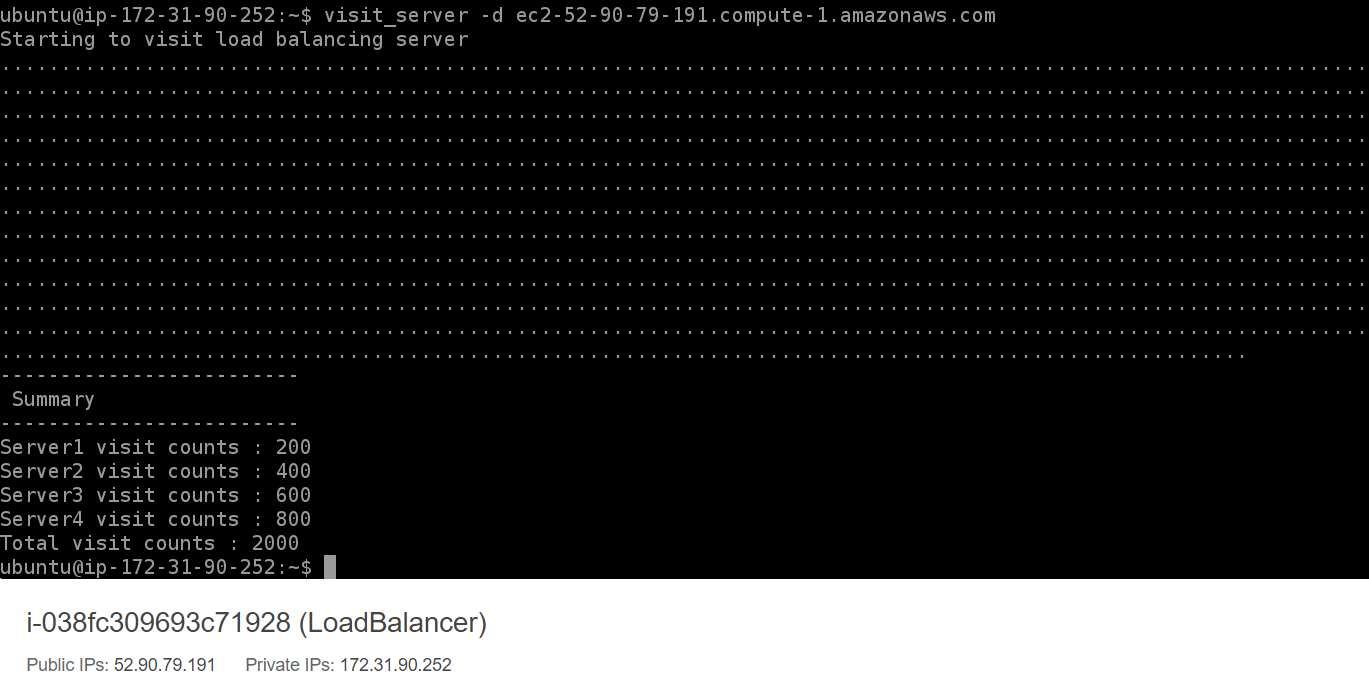
7) We check for following scenarios:

* Scenario #1 – Server1 weight=1, Server2 weight=1, Server3 weight=1, Server4 weight=1
* Scenario #2 – Server1 weight=1, Server2 weight=2, Server3 weight=3, Server4 weight=4
* Scenario #3 – Server1 weight=1, Server2 weight=2, Server3 weight=1, Server4 weight=2

(i) Scenario #1 – Server1 weight=1, Server2 weight=1, Server3 weight=1, Server4 weight=1:



(ii) Scenario #2 – Server1 weight=1, Server2 weight=2, Server3 weight=3, Server4 weight=4:



(iii) Scenario #3 – Server1 weight=1, Server2 weight=2, Server3 weight=1, Server4 weight=2:



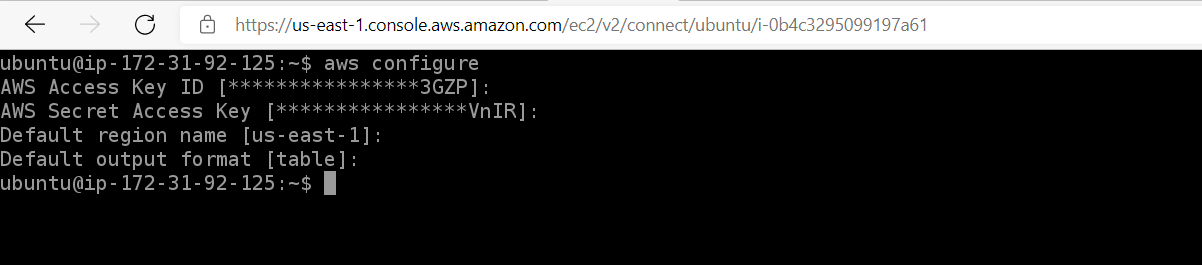
**Additional Steps:–**

**1) Create EC2 by using command line:**

Creating and Accessing EC2 instances by using command line( using Script to record):

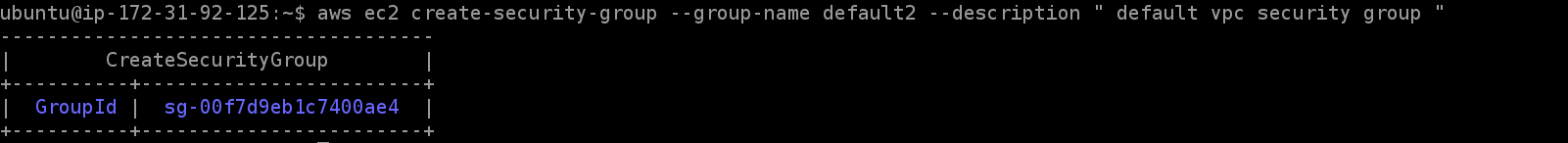
(i) Configure the AWS access key id, secret key and region details using:

$ aws configure



(ii) We create a security group by executing the following command

$ aws ec2 create-security-group --group-name default1 --description "default VPC security group”

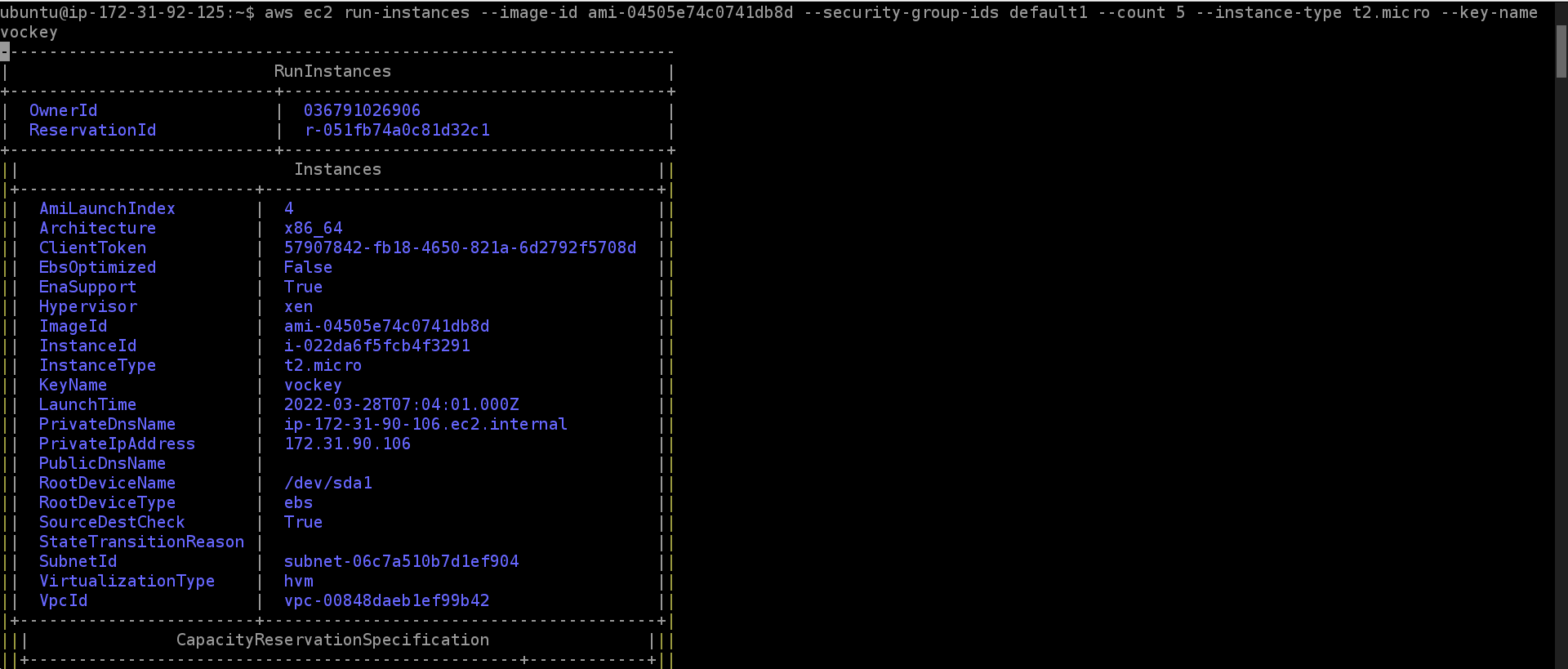


(iii) Then we can create the key-pair using the following command. Here, we use the key we have. The command is:

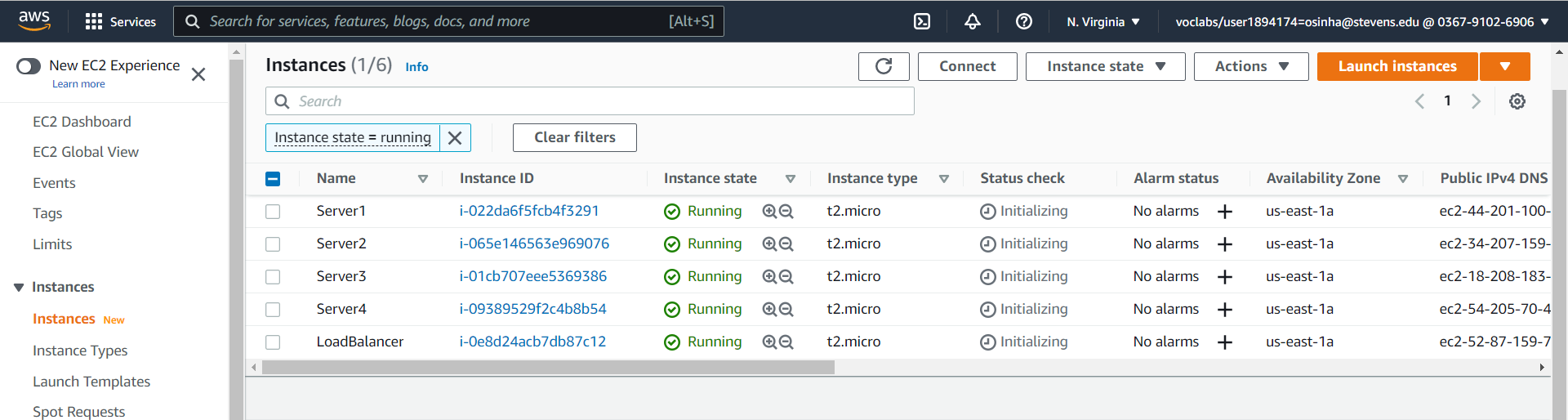
$ aws ec2 create-key-pair --key-name vockey

(iv) Next, we create the required 5 instances using AMI image id, security group id, key-pair name and instance type. We use the following command:

$ aws ec2 run-instances --image-id ami-04505e74c0741db8d --security-group-ids default1 --count 5 --instance-type t2.micro --key-name vockey



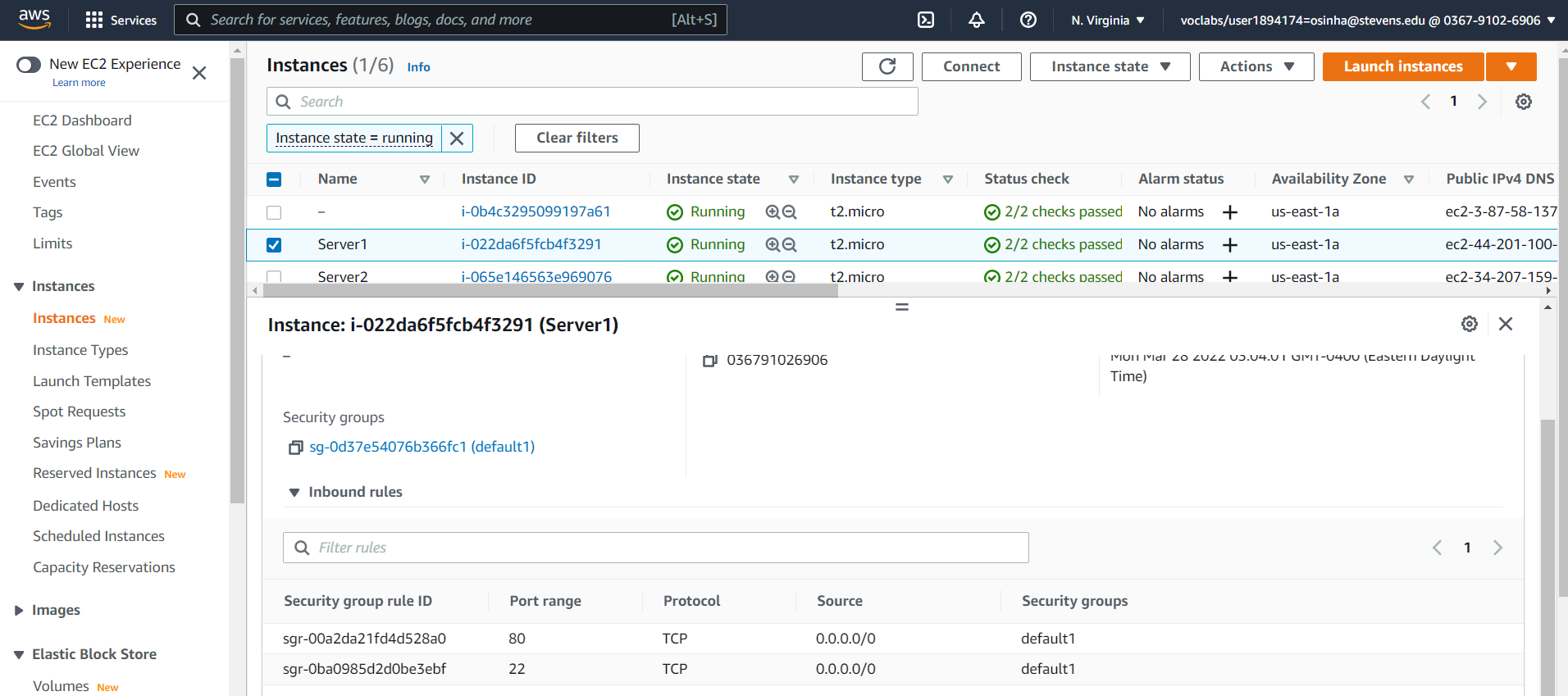
(v) Now our 5 instances are create as needed :



(vi) The last step is to configure the security group for inbound access:

aws ec2 authorize-security-group-ingress –-group-name default1 --protocol tcp –-port 22 –-cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress –-group-name default1 --protocol tcp –-port 80 –-cidr 0.0.0.0/0



(vii) We get instance IDs by following command:

$ aws ec2 describe-instances

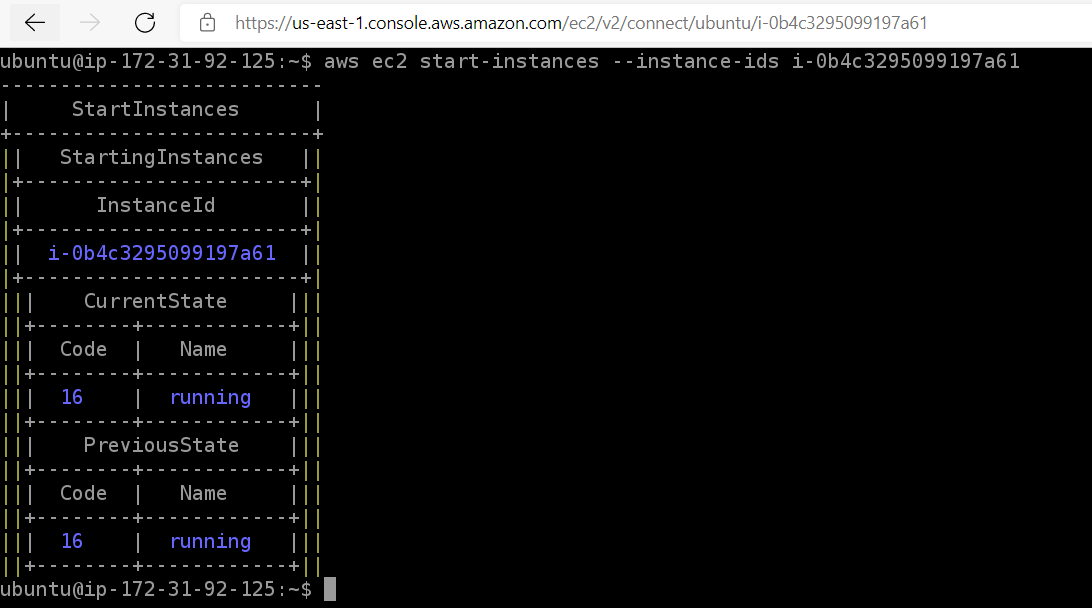




*(image cut due to size)*

(viii) Start the instances by executing the following command for all the instances by replacing instances id

$ aws ec2 start-instances --instance-ids i-0b4c3295099197a61



**2) Tcpdump Analysis**

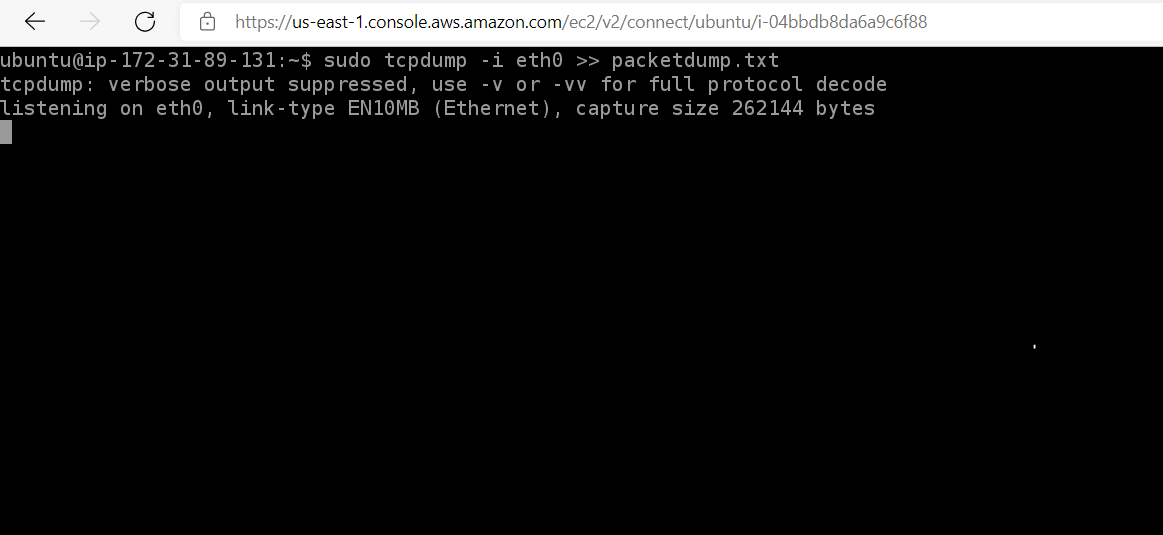
We use following steps in this analysis on ubuntu:

(i) Installing the relevant packages:

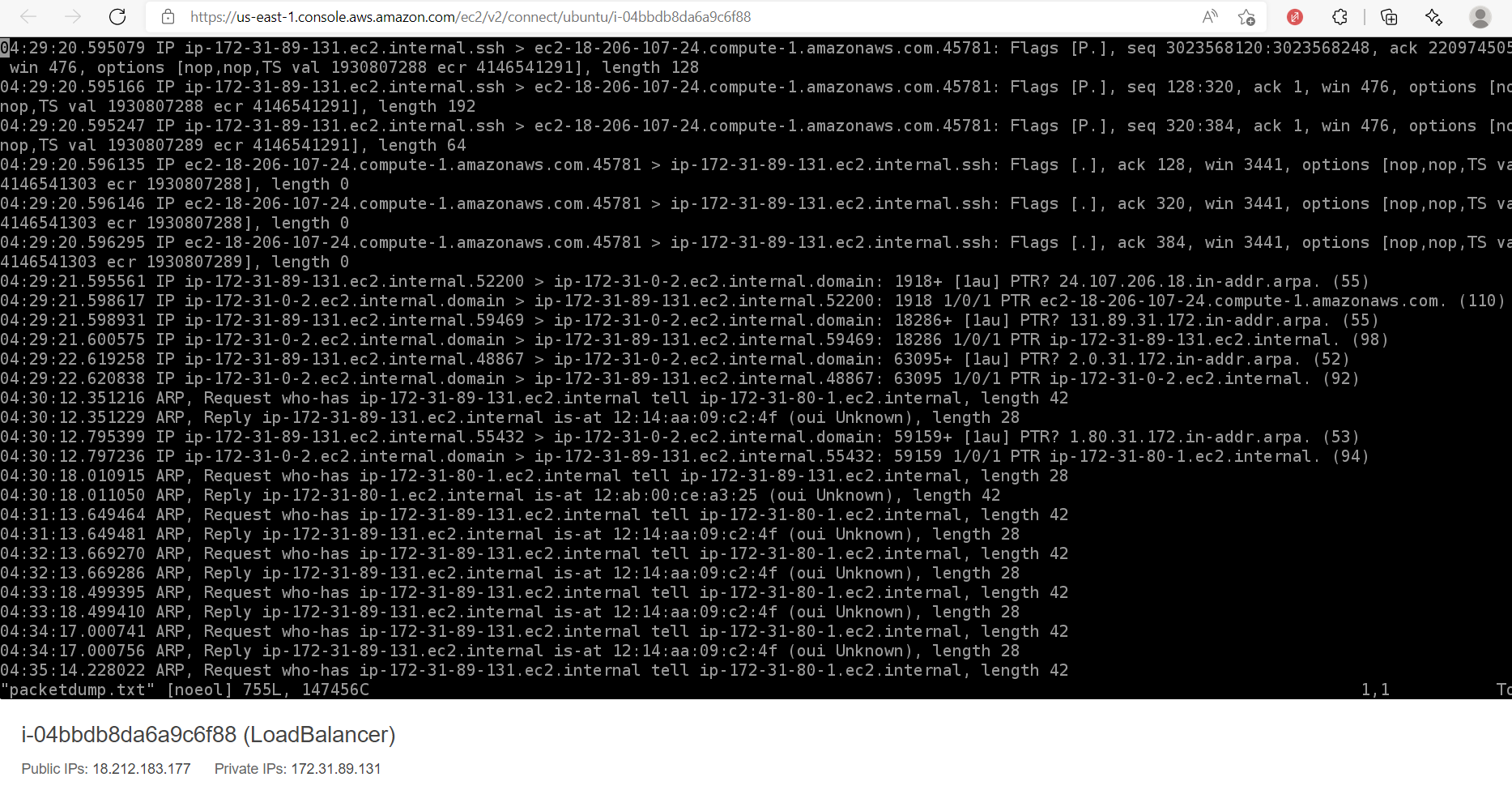
$ sudo apt install tcpdump

(ii) We run the tcpdump command and store result in packetdymp.txt file

$ sudo tcpdump -i eth0 >> packetdump.txt



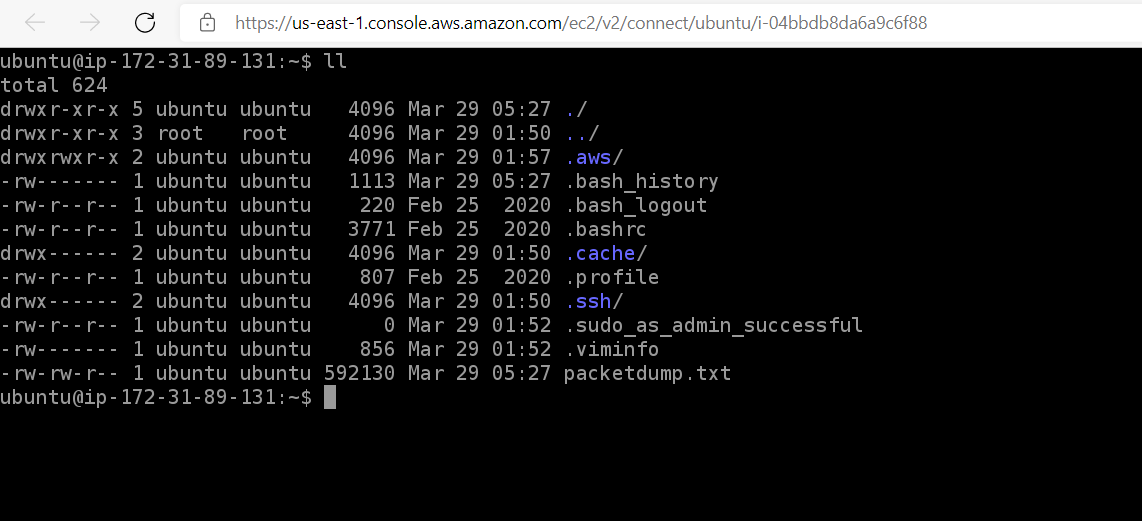
(iii) tcpdump data is stored into packetdump.txt file:



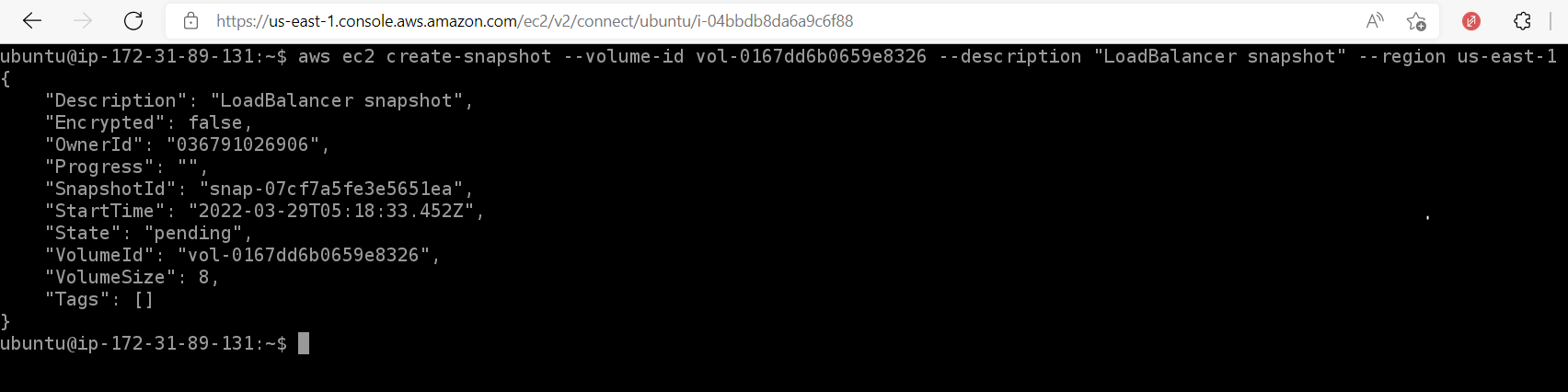
(iv) When we analyze the file contents, we see the first few lines being sent by the remote host to my system. Then the remote host issues an ARP request to get its own address. Since I had made a http request to the load balancer while tcpdump was running (on the load balancer), there are packet information from my local computer to the load balancer, then from the load balancer to one of the servers, and finally back all the way to my local system.

**3) EC2 backup and restore:**

(i) We note down the file currently present in Load Balance. This is done so that later( in step viii) we use this to verify if our volume has been transferred to new instance that we create based on it’s AMI.



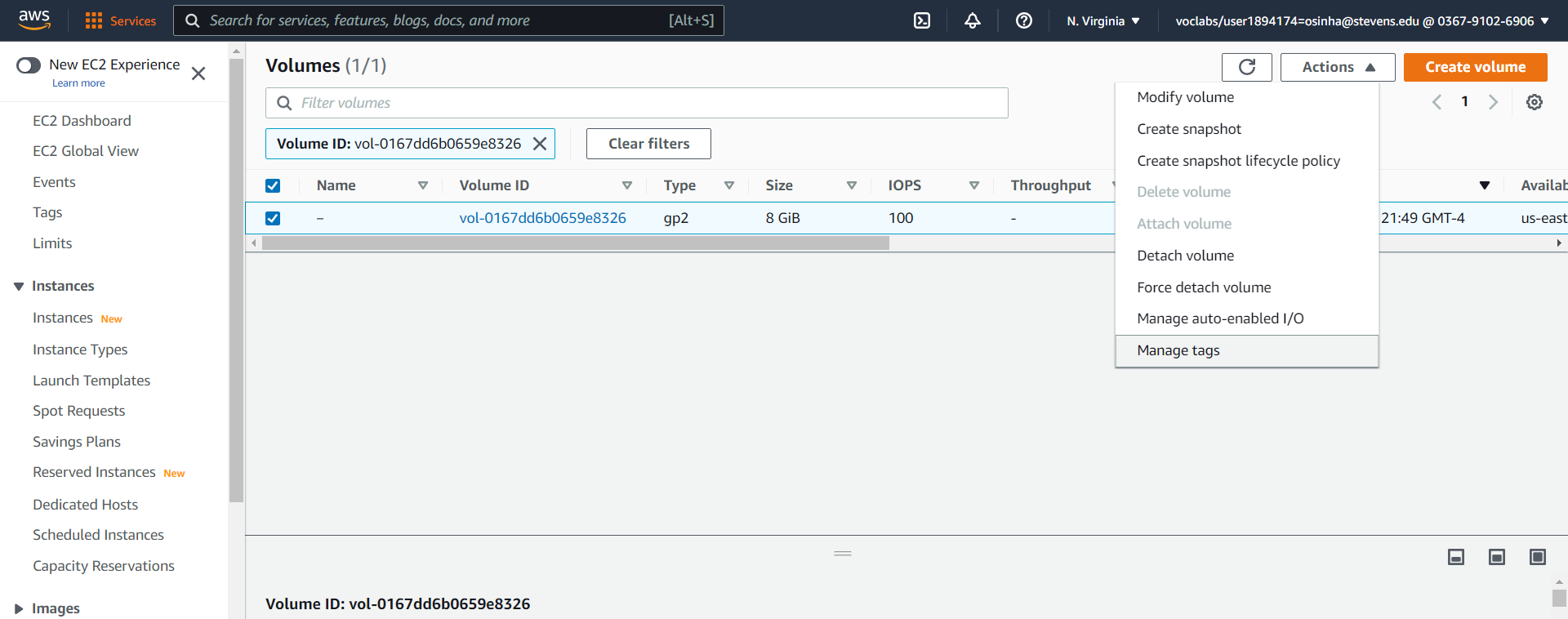
(ii) Taking snapshot of Load balancer, using volume-id(from instance storage):



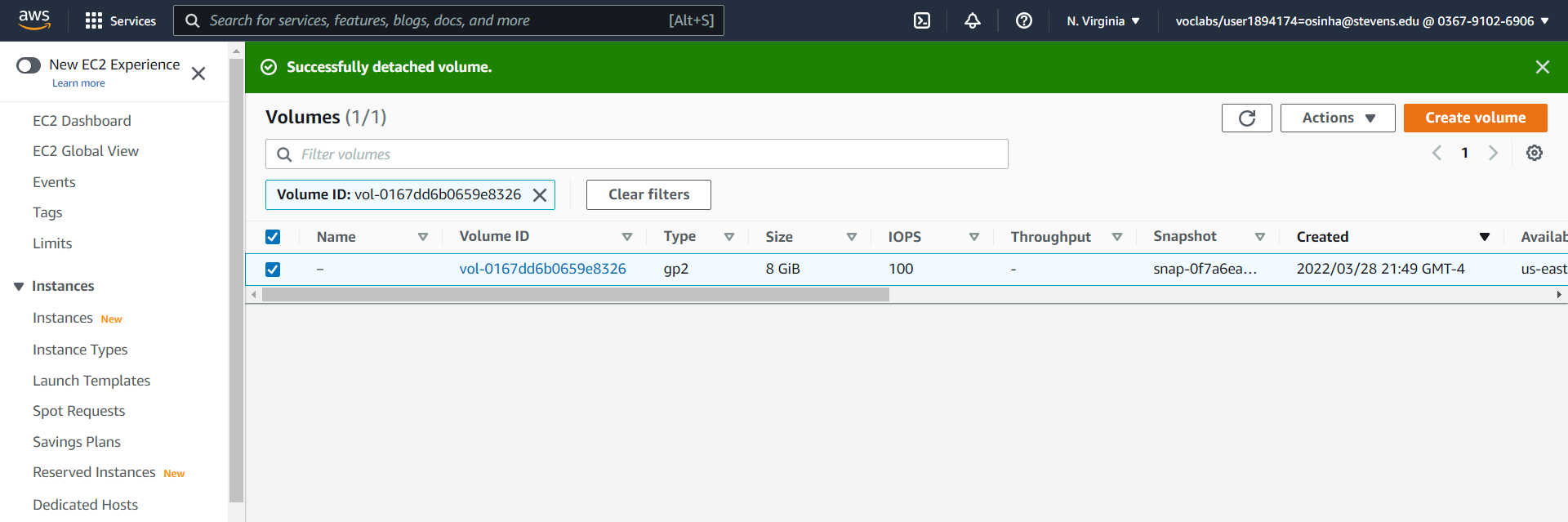
(iii) Stopping the instance to detach the volume:

$ aws ec2 stop-instances --instance-ids i-04bbdb8da6a9c6f88

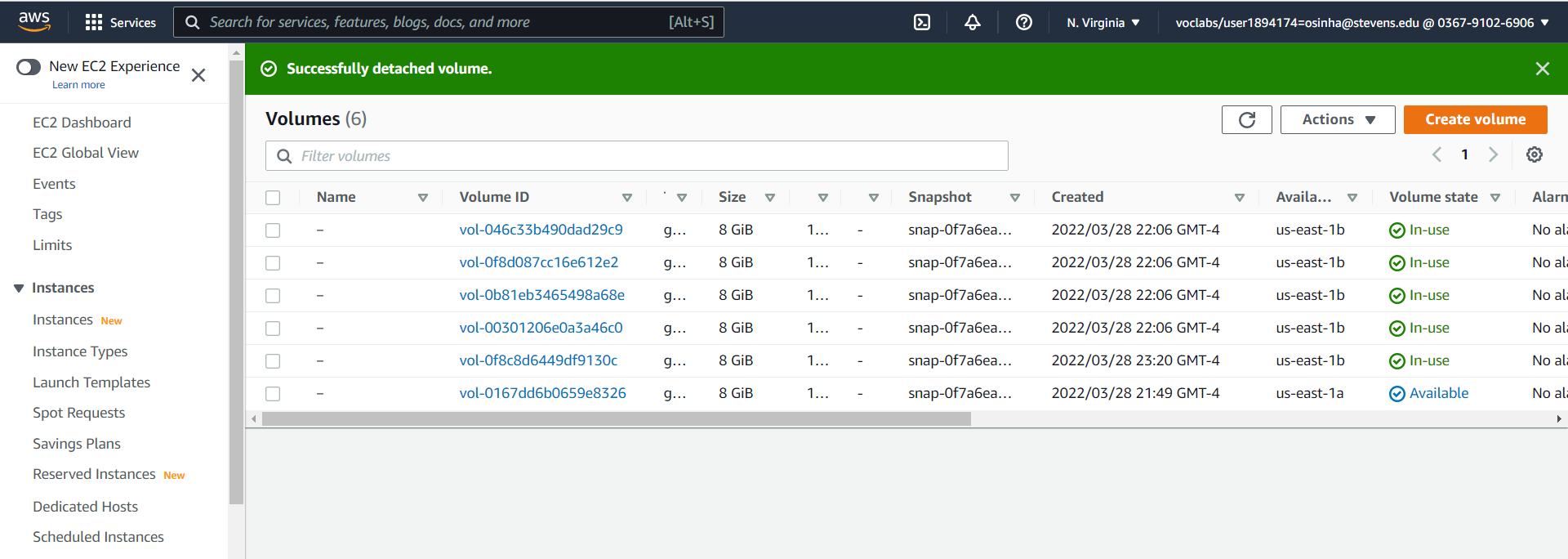
(iv) Detaching the volume manually on console:



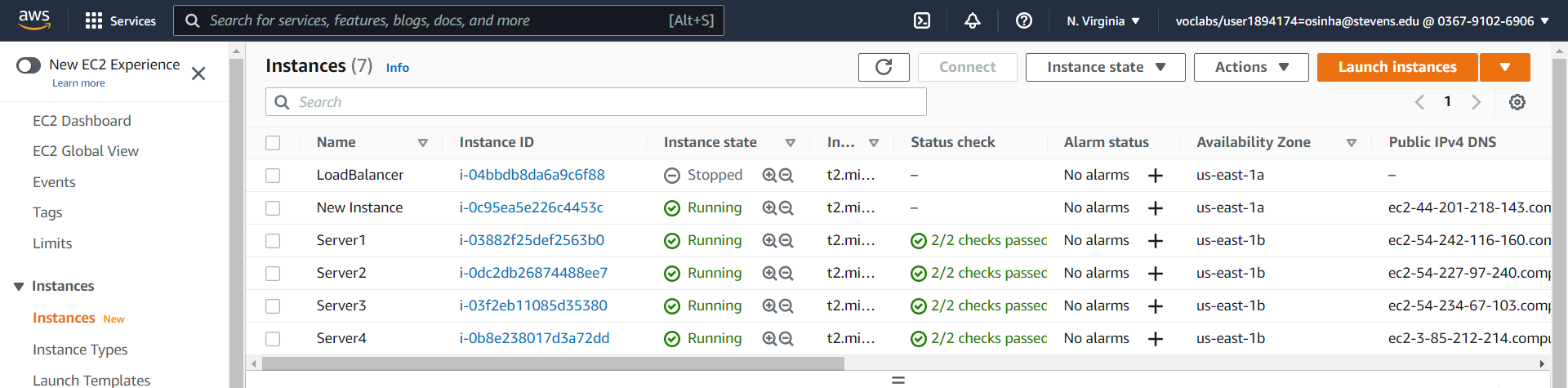
We detach existing data for the new instance:

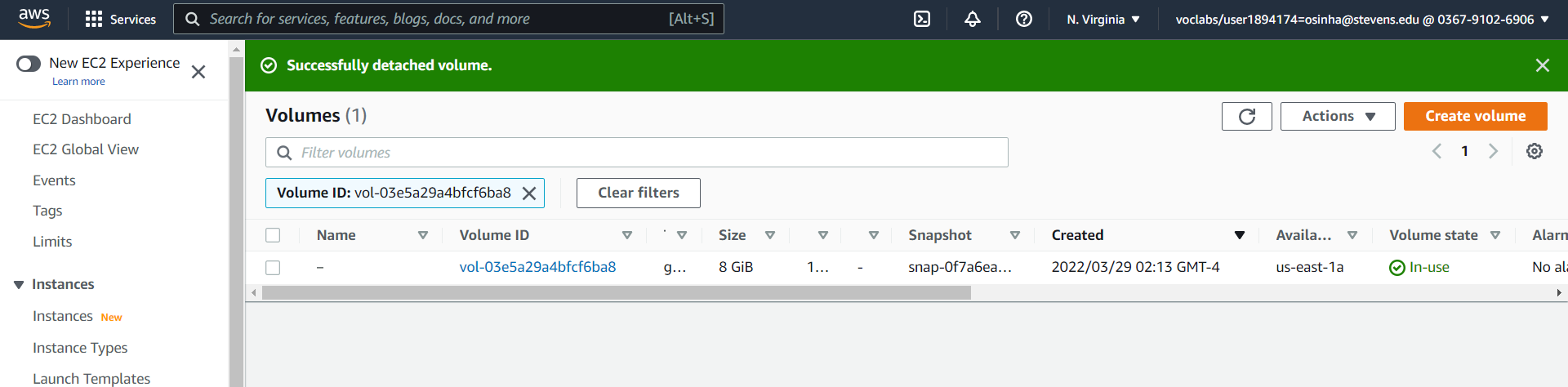


(v) We see the successfully detached *“vol-0167dd6b0659e8326“* volume in list of volumes as “Available”:

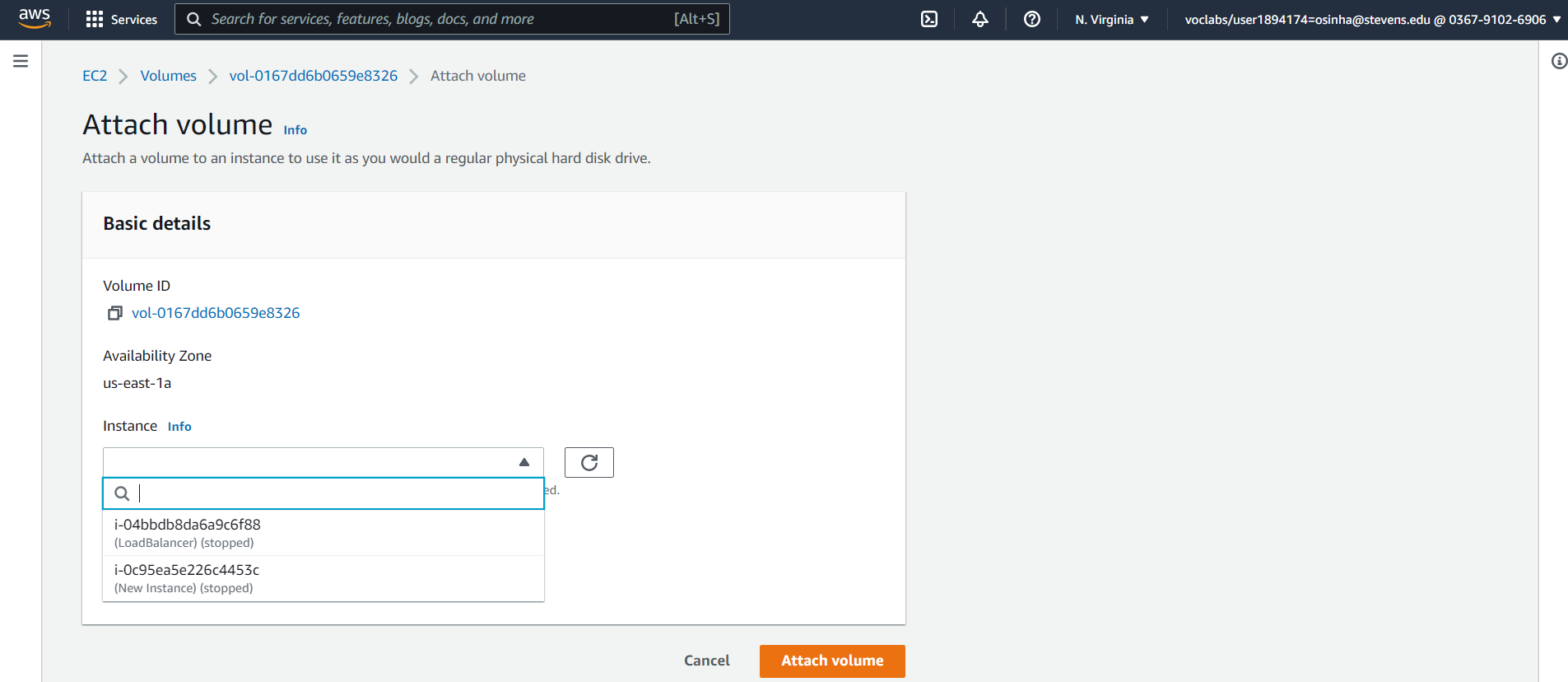


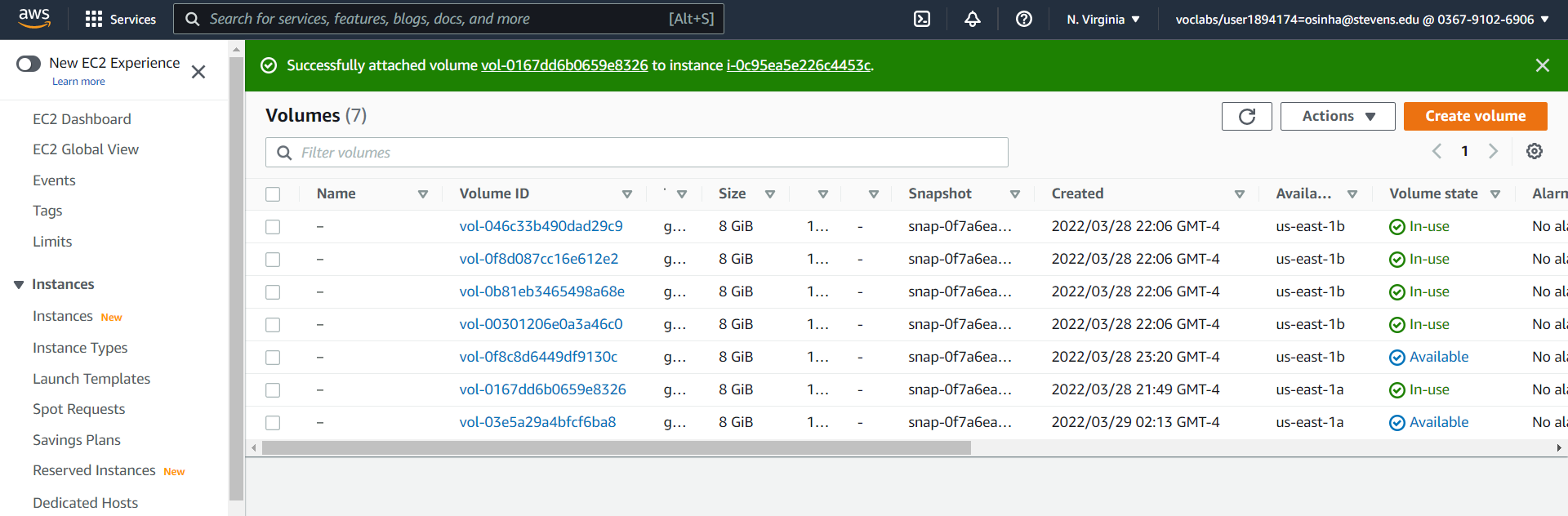
(vi) We create a new instance “New Instance” in the same region. It is stopped in order to detach its existing volume.



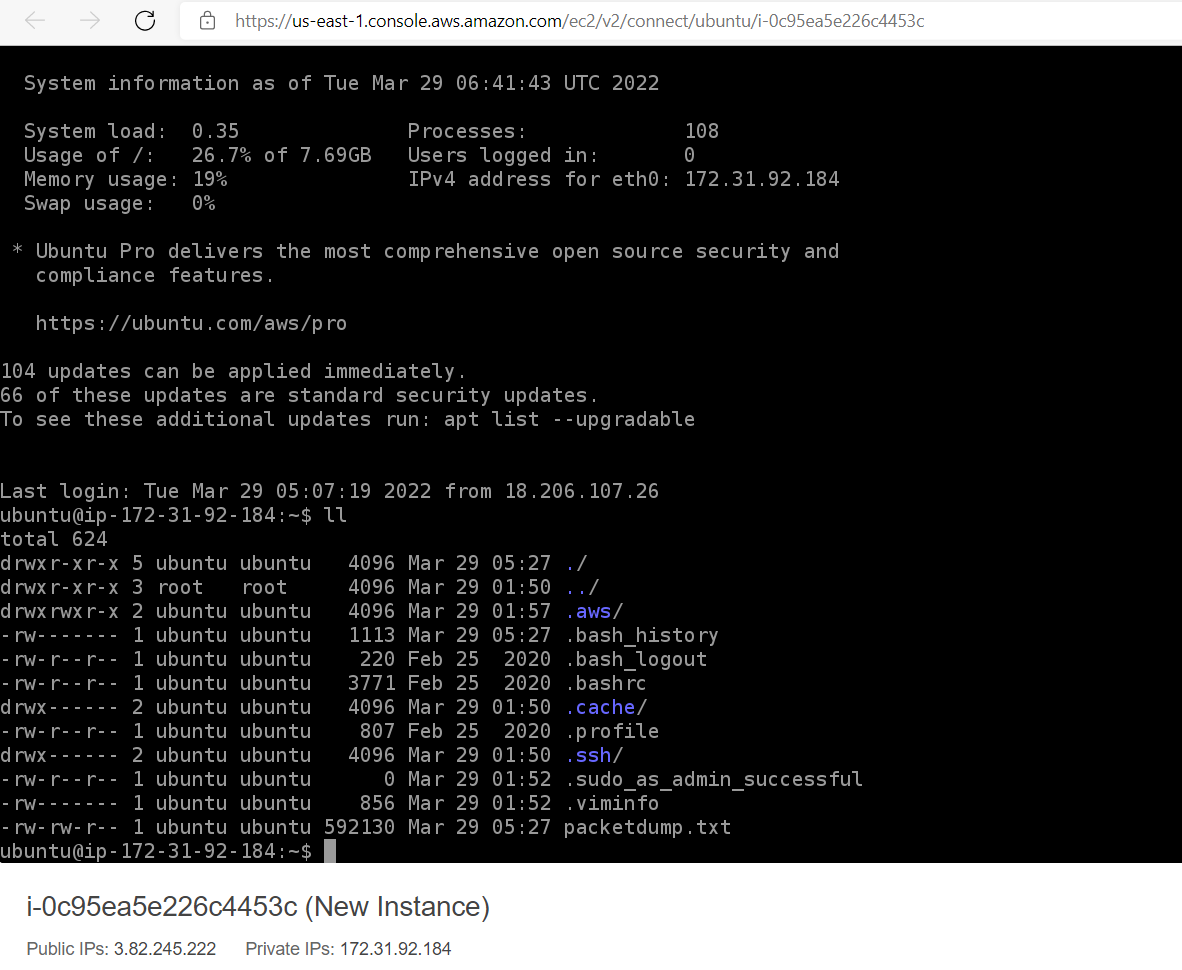


(vii) We now attach the detached volume of the load balancer to this New Instance:





(viii) We verify if the “New Instance” has all files previously held by load balancer. We see in below snap that it has packetdump.txt, .aws (configuration credential directory), etc – all file that were in the load balancer( see step i )



Hence we have successfully take the snapshot of the load balancer AMI, and detach its volume from it. We create a new instance based on this AMI loading it with the load balancers volume. We finally verify that “New Instance” has been successfully transferred all of the load balancer’s files.