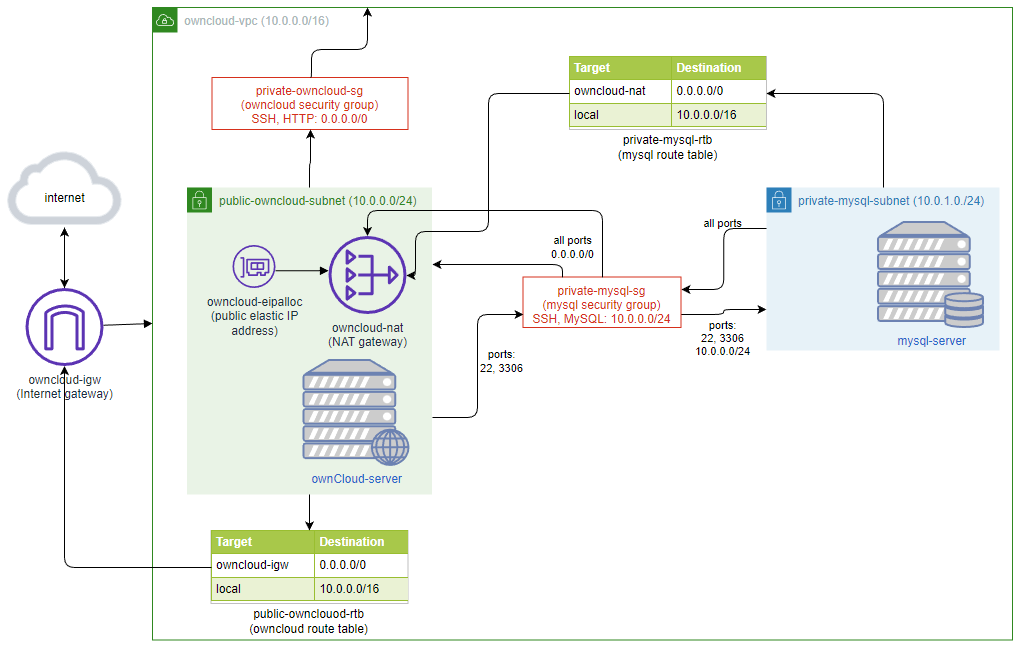
PCPCC | Project

# Creating a file share & sync solution using ownCloud and AWS

Description: To architect a final implementation of an ownCloud application server hosted on AWS for 150 users. This can be hosted in the cloud or on prem.

MySQL Server is hosted in a private subnet that can go out to the internet, but no internet traffic can reach it directly. To access it you must use the web application host setup in the public subnet as a bastion host.

Architecture Diagram



For this implementation the following were used:

MySQL installed with the following:

Installed on Ubuntu 20.04

root user access restricted to local host with password ‘password’

Anonymous user removed

Dropped test database

Created user ‘gluser’ with password ‘password’ with all privileges accessible from anywhere

Created database ‘owncloud’ under ‘gluser’ account

Defined MySQL bind-address to 0.0.0.0 and default\_authentication\_plugin to mysql\_native\_password

NOTES: If this was a production environment more secure passwords would be used, mysql authentication plugin would be configured, gluser access would be limited to the public owncloud subnet and we would use a more restricted bind-address to limit traffic only from the public subnet

ownCloud installed with the following:

Installed on Ubuntu 20.04

php7.4 is installed.

mysql client is install for testing purposes to test connection from public subnet to mysql instances in the private subnet

ownCloud 10.5 for Ubuntu 20.04 is downloaded and installed.

Apache is modified to include `index.php` as a default startup page

Apache is modified to set `/var/www/owncloud` as the document root

Apache is then restarted

### VPC

Name: owncloud-vpc  
Description: This is the VPC that will host our solution in us-east-1 region. The CIDR block is set to 10.0.0.0/16. DNS support and DNS Host names are enabled.

vpc = ec2\_client.create\_vpc(

    CidrBlock         = '10.0.0.0/16',

    TagSpecifications = [{

        'ResourceType': 'vpc',

        'Tags': [

            {'Key': 'Name', 'Value': 'owncloud-pc'}

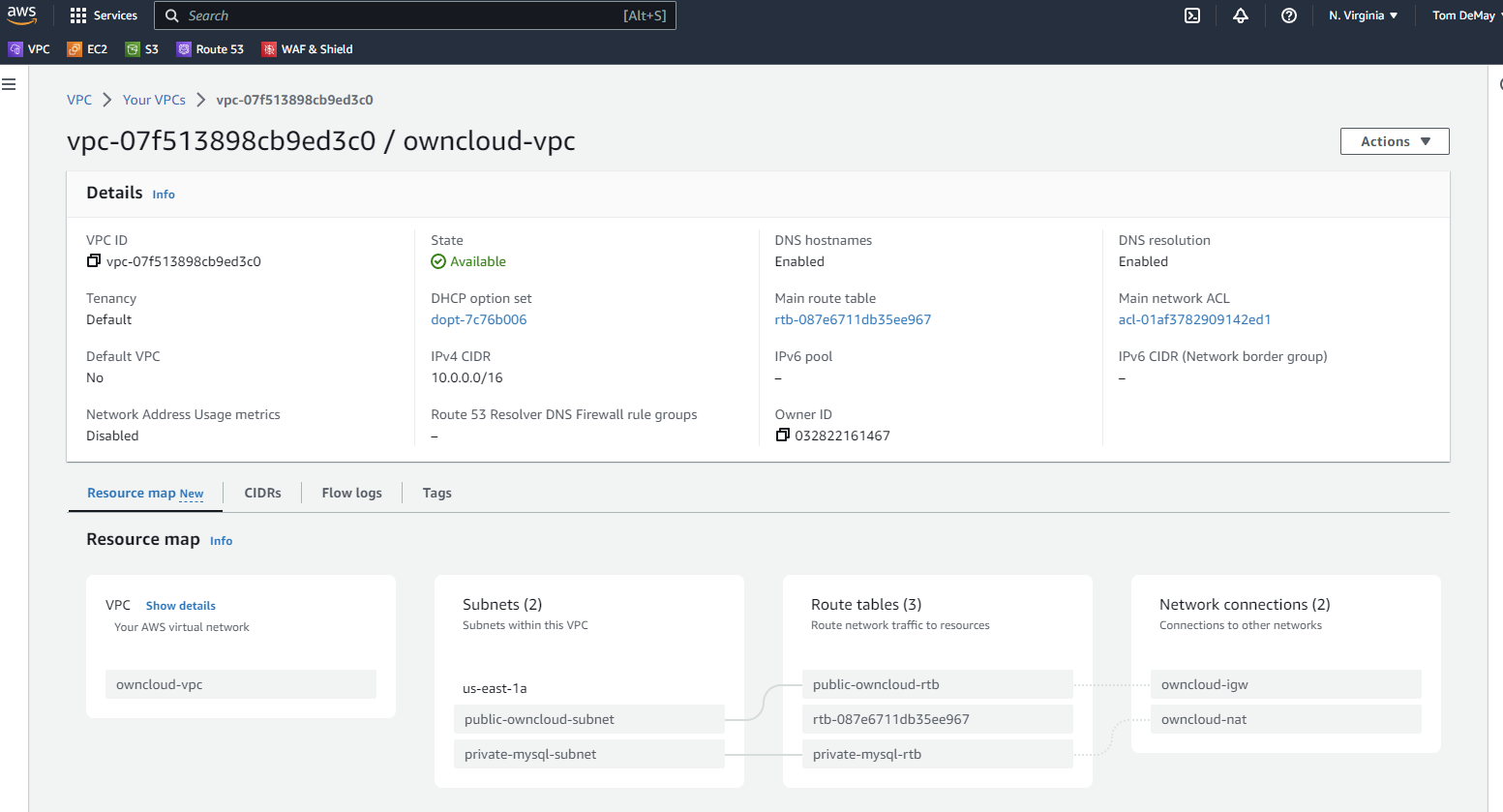
        ]

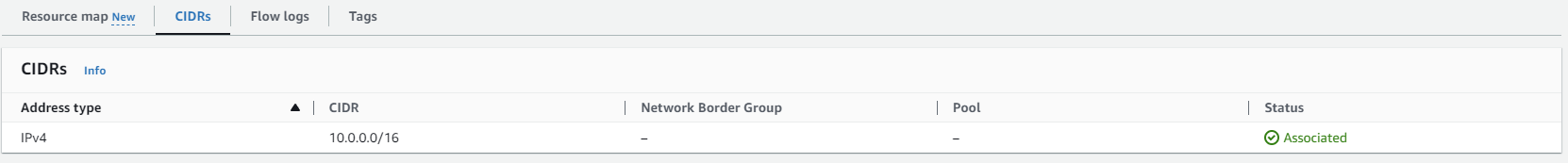
    }]

)

vpc.modify\_attribute(EnableDnsSupport = {'Value': True})

vpc.modify\_attribute(EnableDnsHostnames = {'Value': True})





### Internet Gateway

Name: owncloud-igw  
Description: This is the internet gateway that provides access to and from the cloud to the VPC

igw = create\_internet\_gateway(

    TagSpecifications=[{

        'ResourceType': 'internet-gateway',

        'Tags'        : [

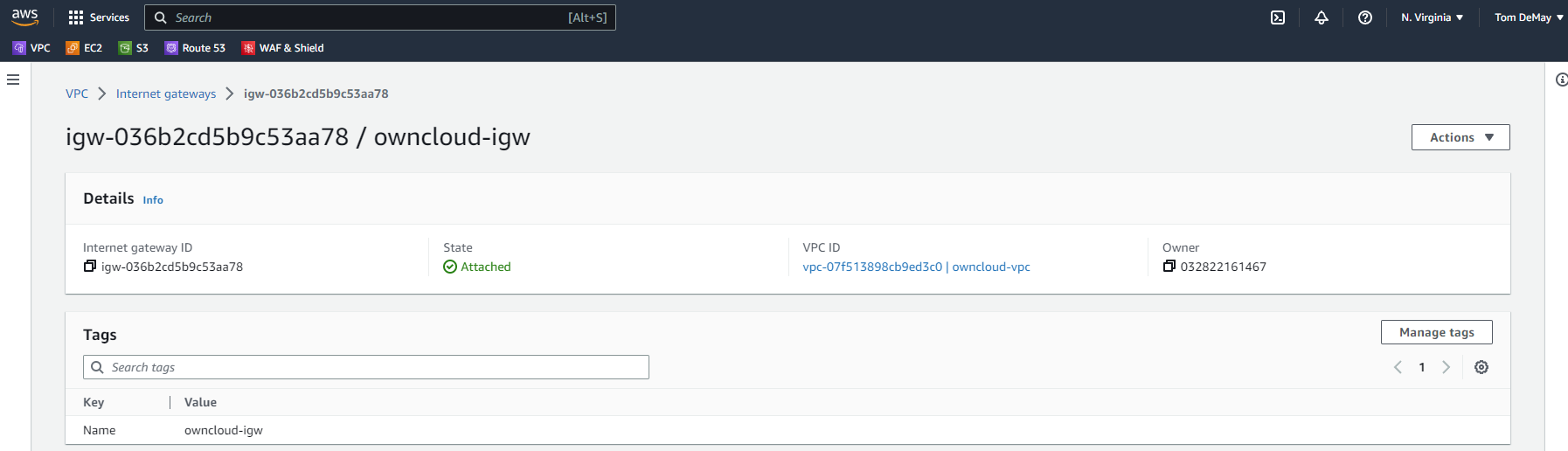
            {'Key': 'Name', 'Value': 'owncloud-igw'}

        ]

    }],

)

igw.attach\_to\_vpc(VpcId=vpc.id)



### Elastic IP address

Name: owncloud-eipalloc

Description: This elastic IP Address is attached to the NAT gateway (owncloud-nat) on the VPC

eipalloc\_id = ec2\_client.allocate\_address(

    Domain='vpc',

    TagSpecifications=[{

      'ResourceType': 'elastic-ip',

        'Tags'        : [

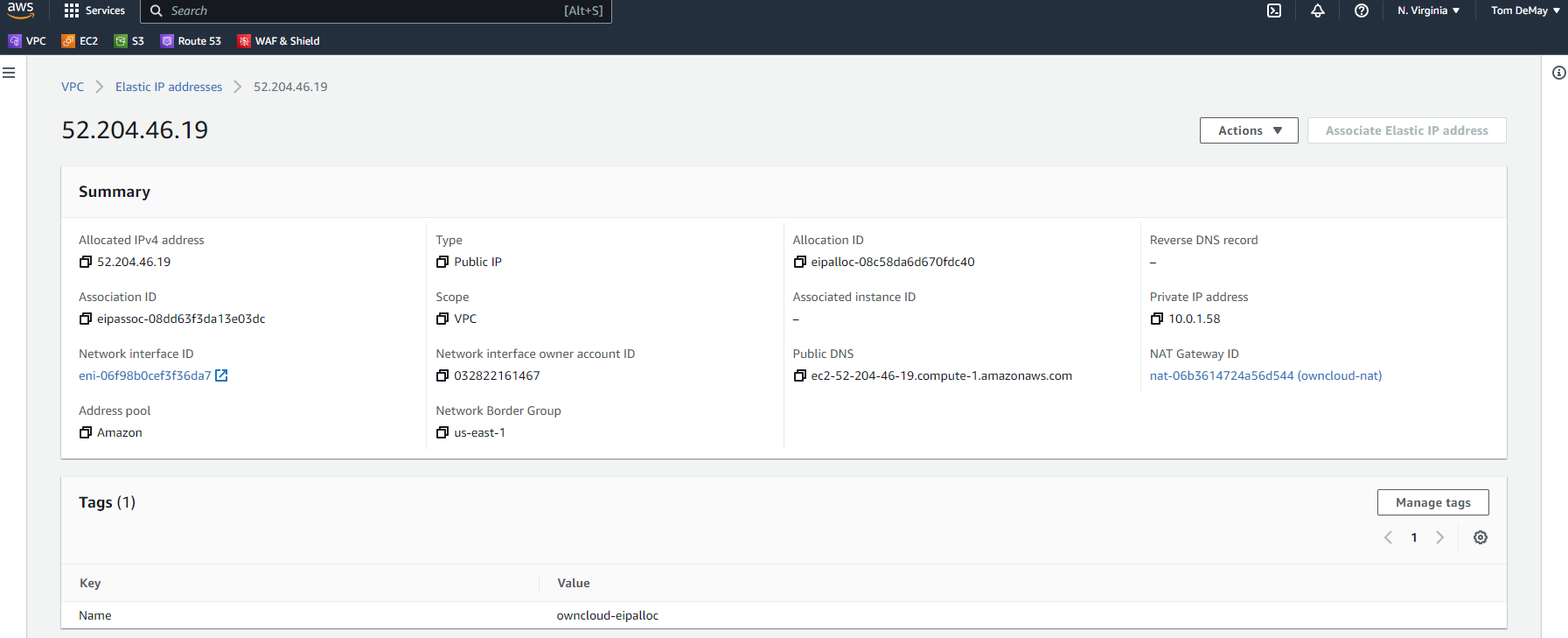
            {'Key': 'Name', 'Value': 'owncloud-nat'}

        ]

    }],

)['AllocationId']

eipalloc = ec2\_resource.VpcAddress(eipalloc\_id)



### Public Subnet

NAME: public-owncloud-subnet  
Description: This is a public subnet to host an EC2 instance with the owncloud application web server.

Route tables attached for 0.0.0.0/0 to the internet gateway. This is assigned CIDR block of 10.0.1.0/24 which allows for 251 IP Addresses

public\_subnet = vpc.create\_subnet(

    TagSpecifications=[{

        'ResourceType': 'subnet',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'public-owncloud-subnet'}

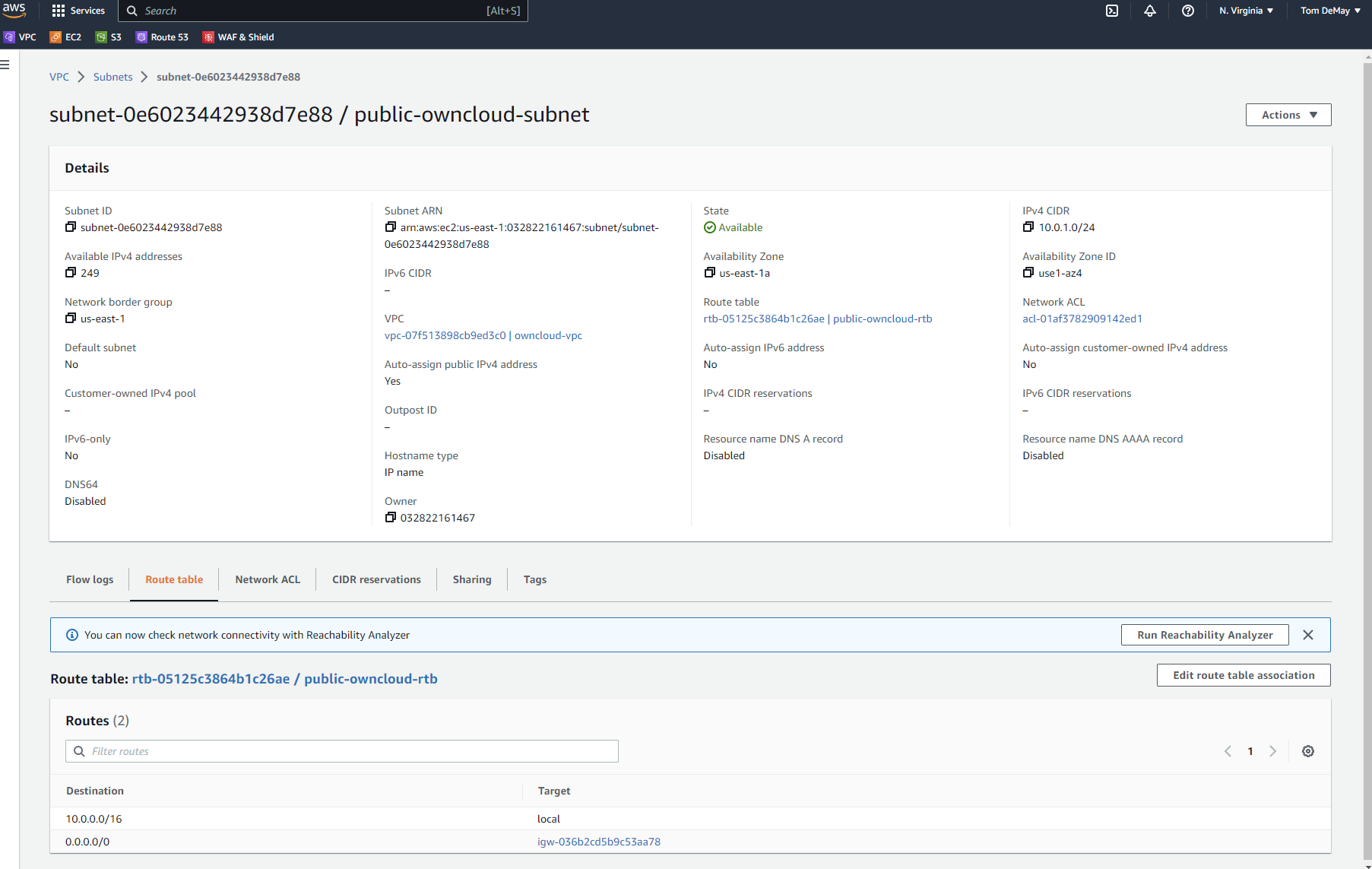
        ]

    }],

    AvailabilityZone='us\_east\_1a',

    CidrBlock='10.0.1.0/24'

)



### Private Subnet

Name: private-mysql-subnet  
Description: This is the private subnet to host an EC2 instance running MySQL. Route tables attached to 0.0.0.0/0 to NAT Gateway. CIDR block is 10.0.2.0/24 which allows for 251 IP Addresses.

private\_subnet = vpc.create\_subnet(

    TagSpecifications=[{

        'ResourceType': 'subnet',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'private-mysql-subnet'}

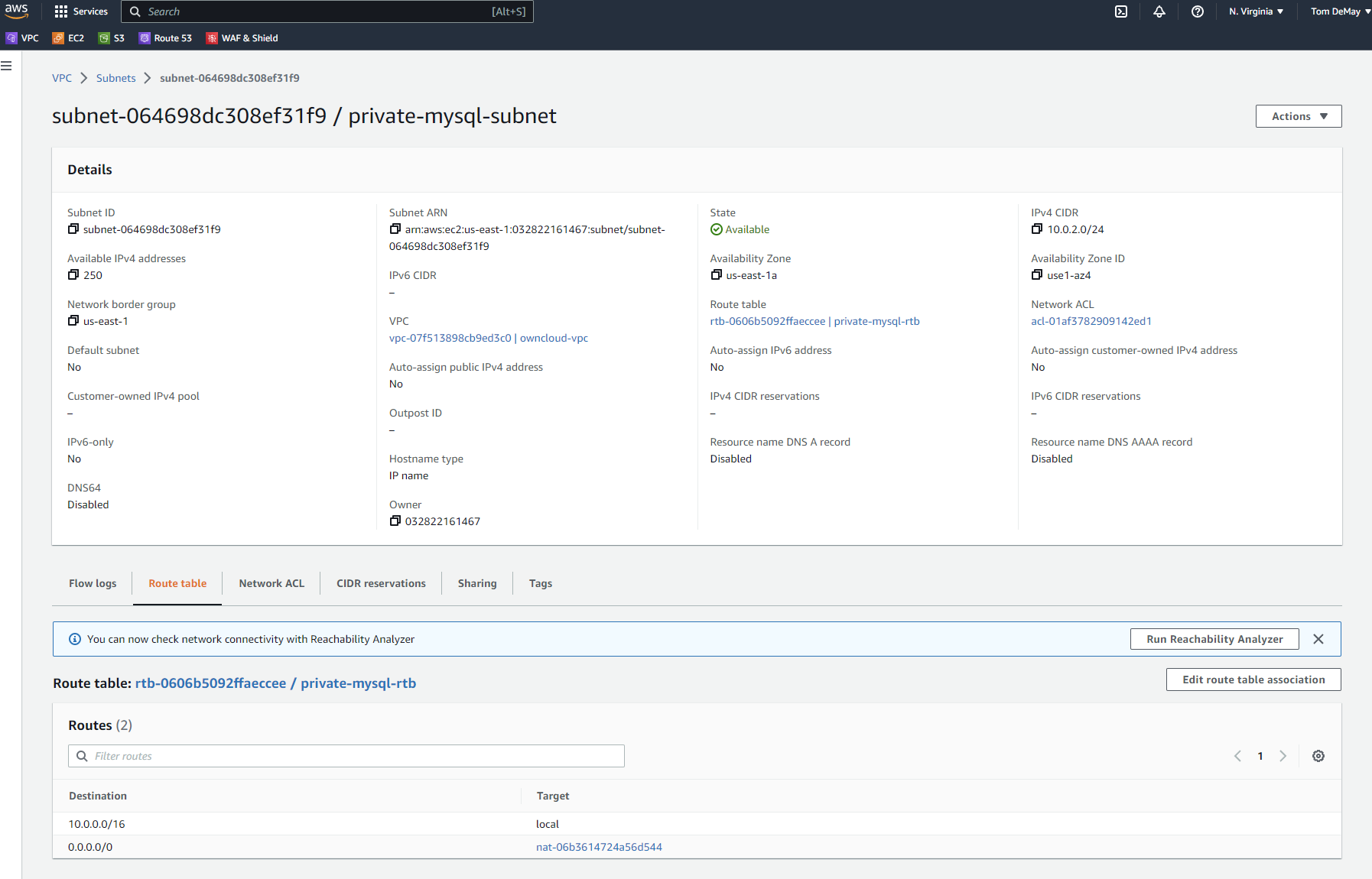
        ]

    }],

    AvailabilityZone='us\_east\_1a',

    CidrBlock='10.0.2.0/24'

)



### NAT Gateway

Name: owncloud-nat

Description: This is attached to VPC and public subnet hosting ownCloud web server EC2 instance to allow the private subnet to reach the internet gateway.

nat\_id = Configuration.session.ec2\_client.create\_nat\_gateway(

    AllocationId = eipalloc.id,

    SubnetId = subnet.id,

    TagSpecifications = [{

        'ResourceType': 'natgateway',

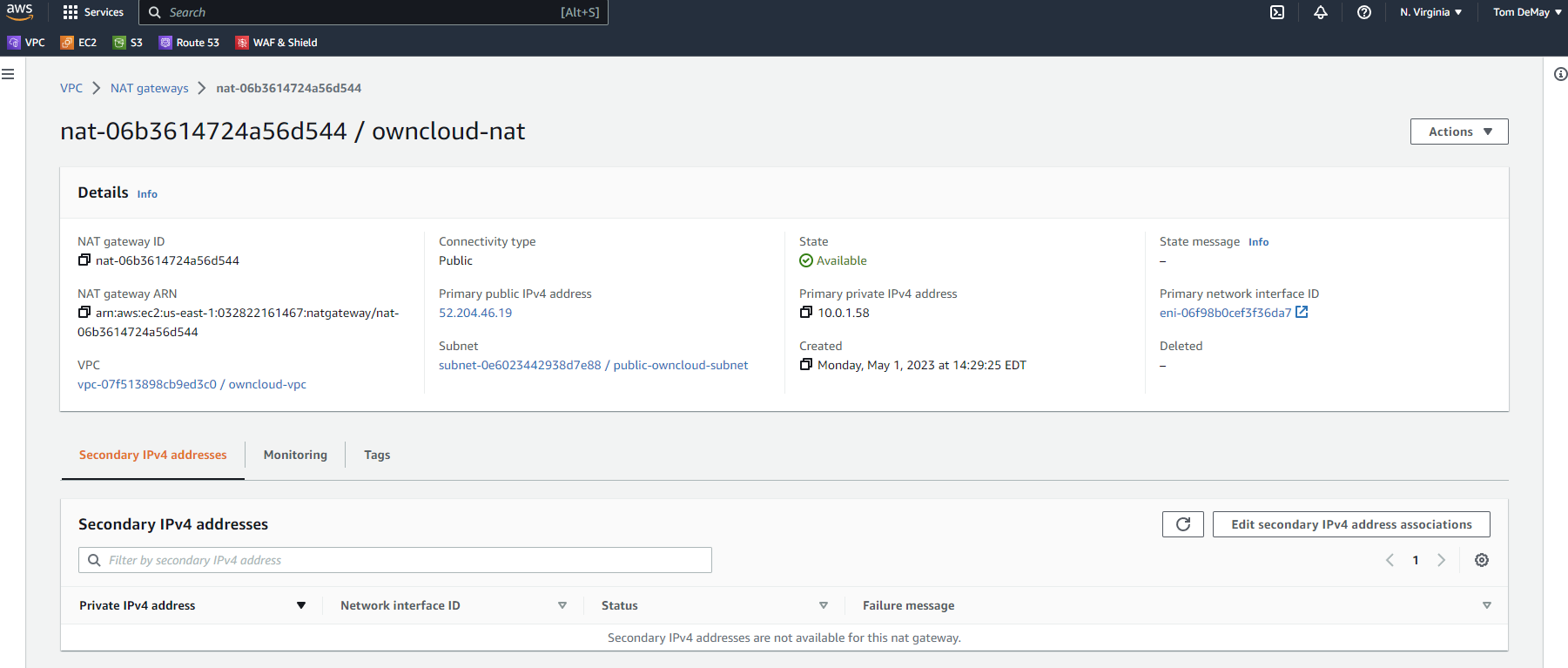
        'Tags': [

            { 'Key': 'Name', 'Value': 'owncloud-nat' }

        ]

    }]

)['NatGateway']['NatGatewayId']



Public Security Group

Name: public-owncloud-sg

Description: This security group permits HTTP, HTTPS and SSH ingress from anywhere and permits all traffic egress to anywhere. SSH can be disabled in production.

public\_sg = ec2\_resource.create\_security\_group(

    Description = 'Opens SSH and HTTP ports from the internet',

    GroupName   = 'public-owncloud-sg',

    VpcId       = vpc.id,

    TagSpecifications =[{

        'ResourceType': 'security-group',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'public-owncloud-sg'}

        ]

    }]

)

public\_sg.authorize\_ingress(

    IpPermissions = [{

        'FromPort': 80,

        'IpProtocol': 'tcp',

        'IpRanges': [{

            'CidrIp': '0.0.0.0/0',

            'Description': 'Opening HTTP from the internet'

        }],

        'ToPort': 80

    }],

    TagSpecifications = [{

        'ResourceType': 'security-group-rule',

        'Tags': [

            {'Key': 'Name', 'Value': 'open-http-public'}

        ]

    }])

public\_sg.authorize\_ingress(

    IpPermissions = [{

        'FromPort': 443,

        'IpProtocol': 'tcp',

        'IpRanges': [{

            'CidrIp': '0.0.0.0/0',

            'Description': 'Opening HTTPS from the internet'

        }],

        'ToPort': 443

    }],

    TagSpecifications = [{

        'ResourceType': 'security-group-rule',

        'Tags': [

            {'Key': 'Name', 'Value': 'open-https-public'}

        ]

    }])

public\_sg.authorize\_ingress(

    IpPermissions = [{

        'FromPort': 22,

        'IpProtocol': 'tcp',

        'IpRanges': [{

            'CidrIp': '0.0.0.0/0',

            'Description': 'Opening SSH from the internet'

        }],

        'ToPort': 22

    }],

    TagSpecifications = [{

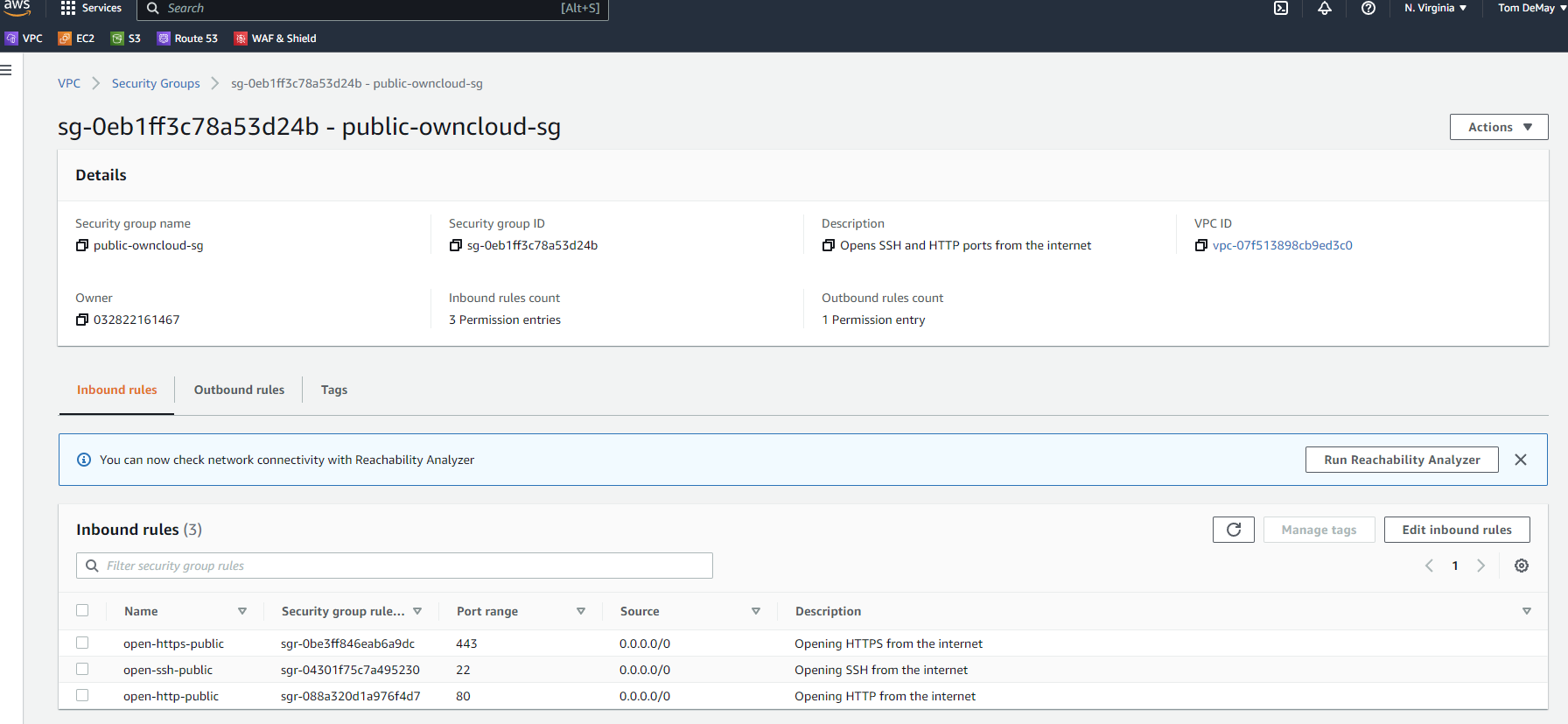
        'ResourceType': 'security-group-rule',

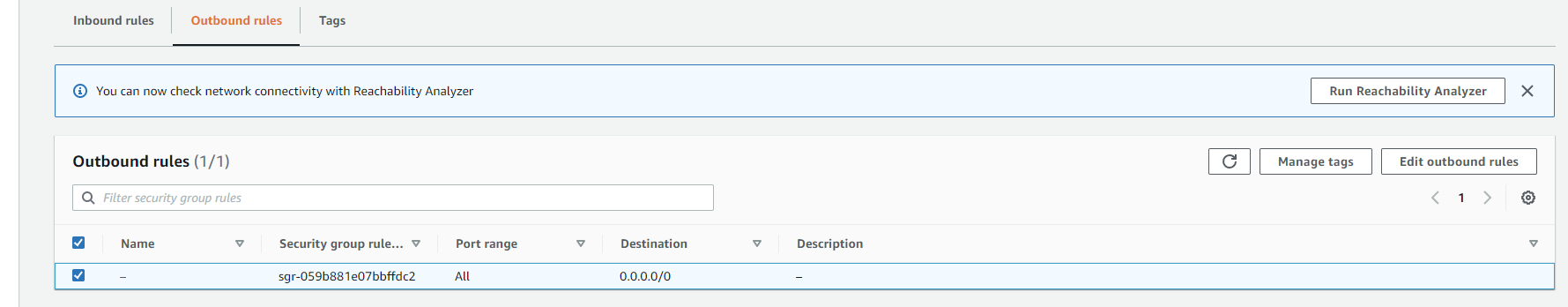
        'Tags': [

            {'Key': 'Name', 'Value': 'open-ssh-public'}

        ]

    }])





### Private Security Group

Name: private-mysql-sg

Description: This is the private security group to limit traffic from public subnet to private subnet ONLY for SSH and MySQL. Once in production the ssh port ingress rule could be removed. In fact, with the bootstrap scripts in place the port isn’t even needed, but may be needed for patches, so I included it. Egress rule permits all traffic to anywhere.

private\_sg = ec2\_resource.create\_security\_group(

    Description = 'Opens SSH and MySQL ports from the public subnet',

    GroupName   = 'private-owncloud-sg',

    VpcId       = vpc.id,

    TagSpecifications =[{

        'ResourceType': 'security-group',

        'Tags'        : [

            {'Key': 'Name', 'Value': private-mysql-sg'}

        ]

    }]

)

private\_sg.authorize\_ingress(

    IpPermissions = [{

        'FromPort': 3306,

        'IpProtocol': 'tcp',

        'IpRanges': [{

            'CidrIp': '10.0.1.0/24',

            'Description': 'Opening MySQL from the public subnet'

        }],

        'ToPort': 3306

    }],

    TagSpecifications = [{

        'ResourceType': 'security-group-rule',

        'Tags': [

            {'Key': 'Name', 'Value': 'open-mysql-private'}

        ]

    }])

private\_sg.authorize\_ingress(

    IpPermissions = [{

        'FromPort': 22,

        'IpProtocol': 'tcp',

        'IpRanges': [{

            'CidrIp': '10.0.1.0/24',

            'Description': 'Opening SSH from the public subnet'

        }],

        'ToPort': 22

    }],

    TagSpecifications = [{

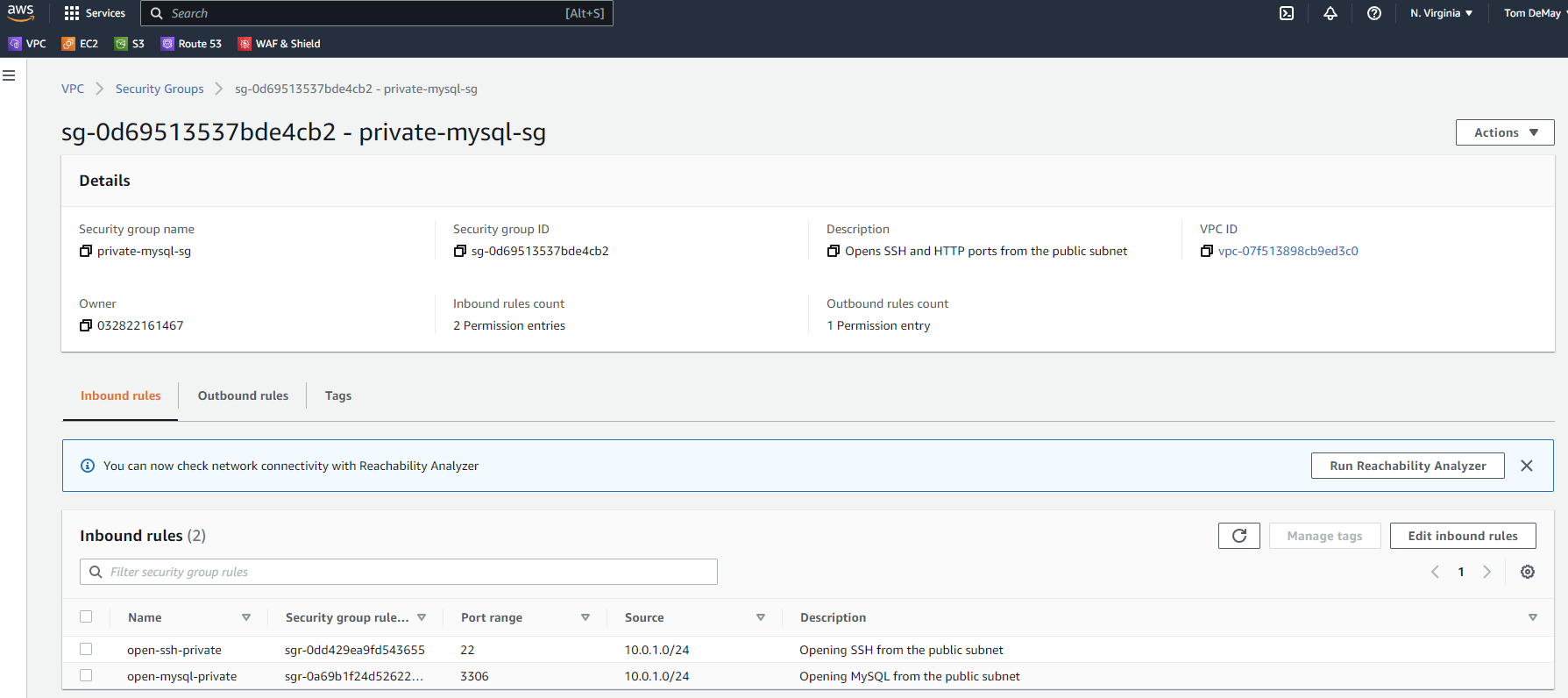
        'ResourceType': 'security-group-rule',

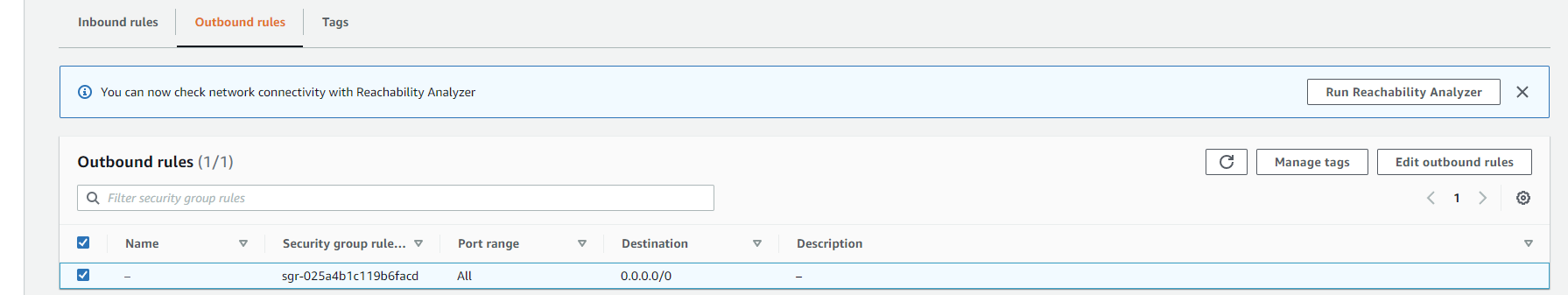
        'Tags': [

            {'Key': 'Name', 'Value': 'open-ssh-private'}

        ]

    }])





### Public Route Table

Name: public-owncloud-rtb

Description: This route table is attached to owncloud VPC and associated with public subnet hosting ownCloud web server EC2 instance. All traffic is routed to internet gateway using CIDR block 0.0.0.0/0 and 10.0.0.0/16 to local

public\_owncloud\_rtb = ec2\_resource.create\_route\_table(

    VpcId=vpc.id,

    TagSpecifications=[{

        'ResourceType': 'route-table',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'public-owncloud-rtb'}

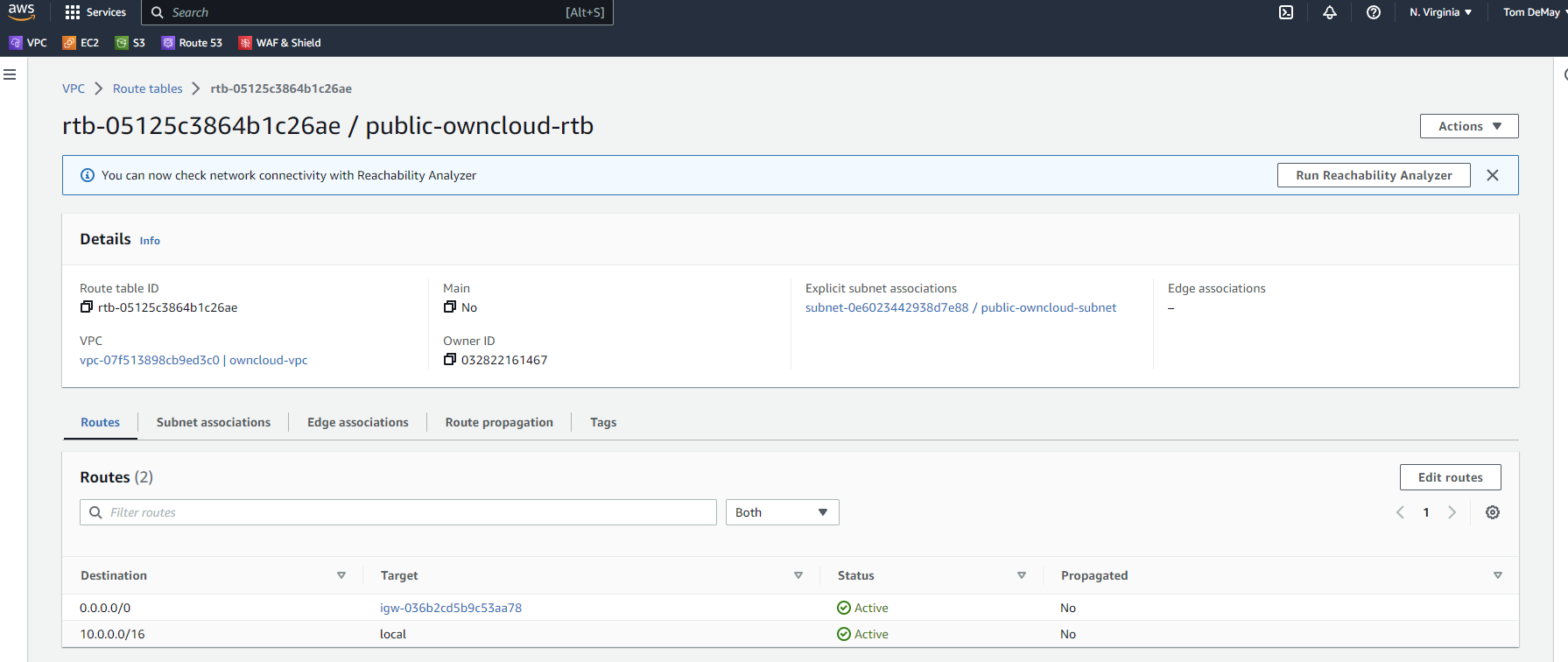
        ]

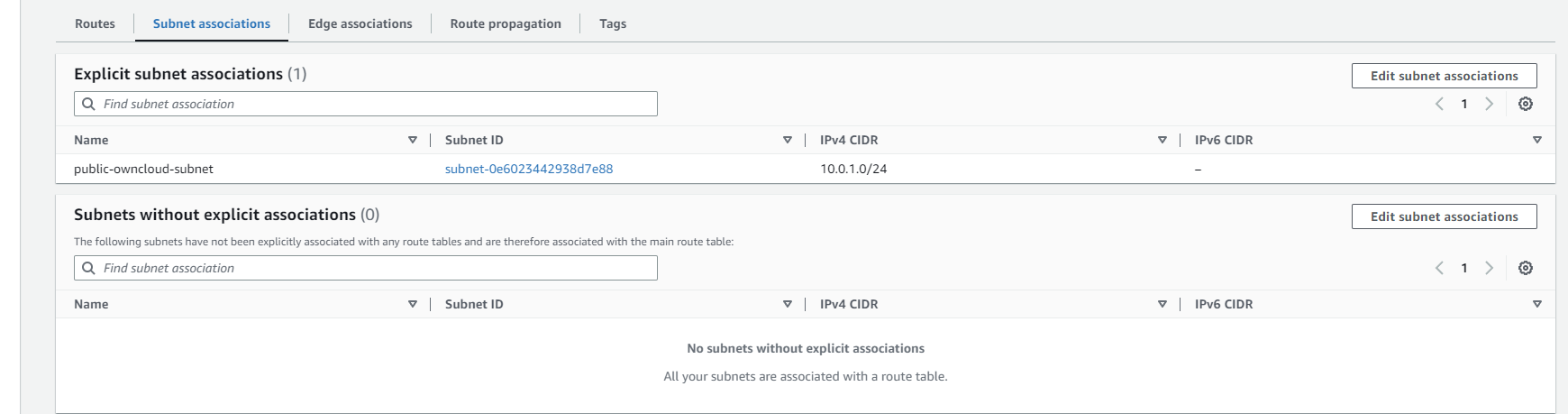
    }]

)

public\_owncloud\_rtb.create\_route(DestinationCidrBlock='0.0.0.0/0', GatewayId=igw.id)

public\_owncloud\_rtb.associate\_with\_subnet(SubnetId=public\_subnet.id)





### Private Route Table

Name: private-mysql-rtb

Description: This route table is attached to owncloud VPC and associated with private subnet hosting MySQL EC2 instance. All traffic is routed to nat gateway using CIDR block 0.0.0.0/0 and 10.0.0.0/16 to local

private\_mysql\_rtb = ec2\_resource.create\_route\_table(

    VpcId=vpc.id,

    TagSpecifications=[{

        'ResourceType': 'route-table',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'private-mysql-rtb'}

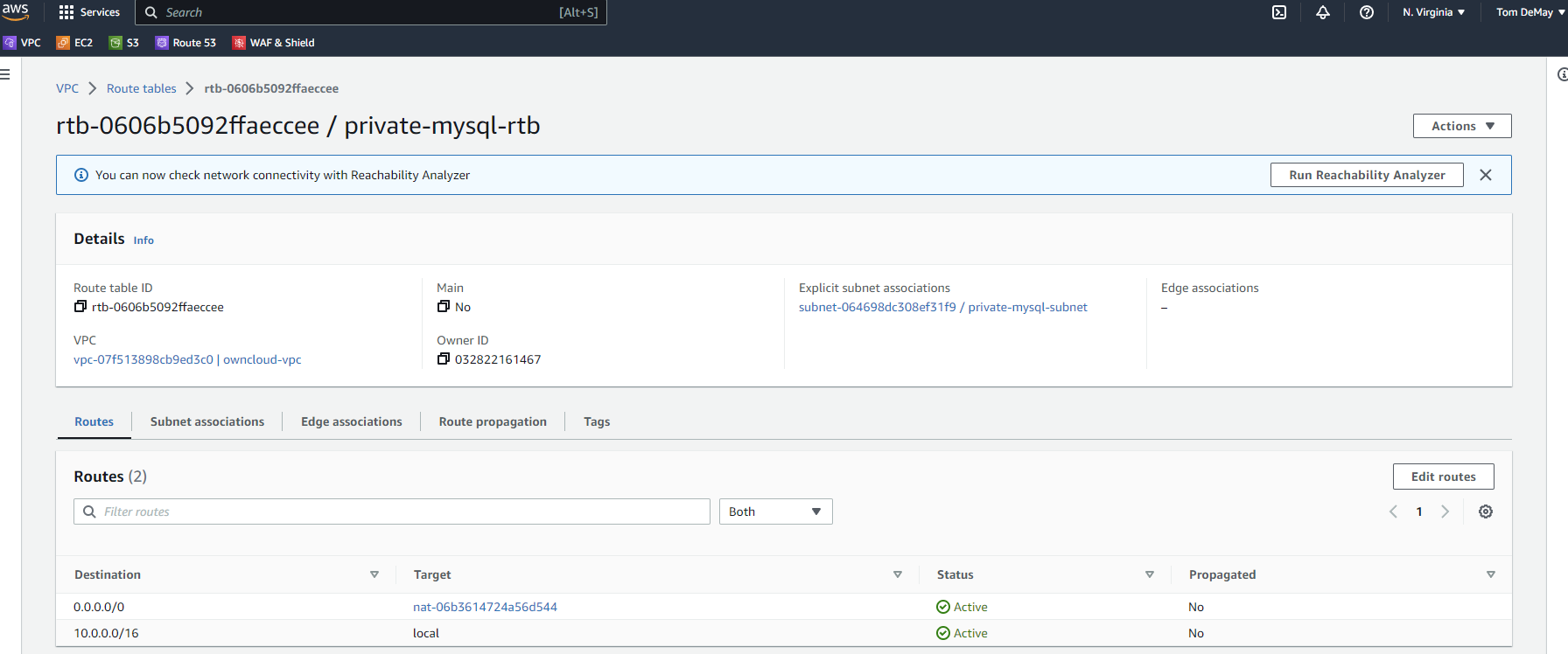
        ]

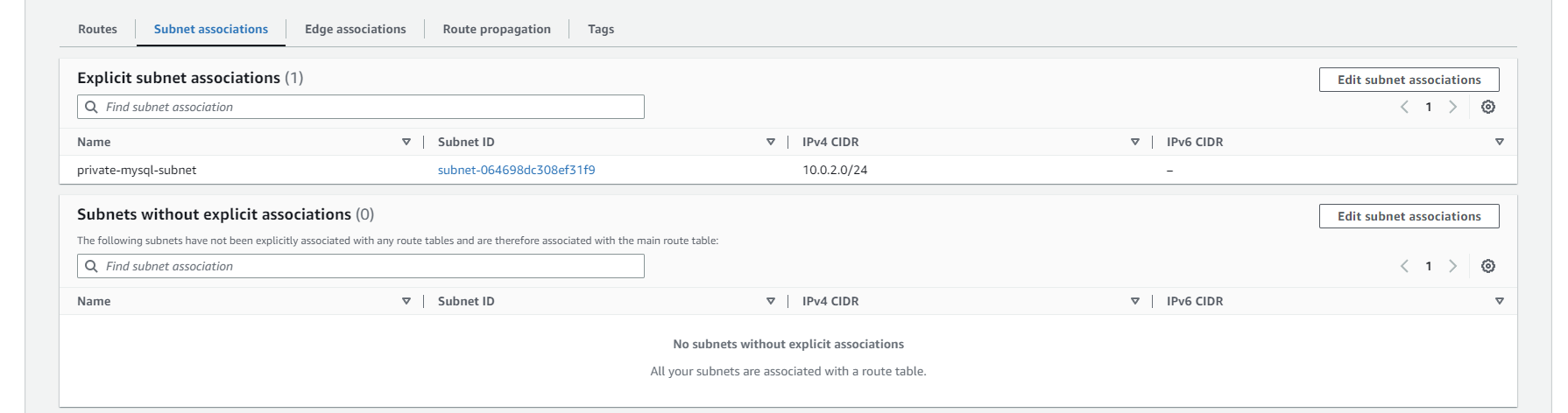
    }]

)

private\_mysql\_rtb.create\_route(DestinationCidrBlock='0.0.0.0/0', NatGatewayId=nat.id)

private\_mysql\_rtb.associate\_with\_subnet(SubnetId=private\_subnet.id)





### EC2 Instance to host web application

Name: owncloud-server

Description: This EC2 instance hosts the ownCloud web application. Currently the files are being stored on the local EBS SSD file system. I chose 30 GiB to start because this was the free tier limit. In production I may have chosen 1 TiB. For only 150 users I felt this was a good way to start. Amazon EBS volume is replicated within its Availability Zone, offers high availability and 99.999% durability. Usage could be scaled up within minutes if needed. EBS supports up to 64TiB if needed. If the ownCloud web application instance is scaled horizontally, some other AWS Managed Storage service like EFS or S3 must be considered to share the files with all ownCloud web application instances.

Ubuntu Server 20.04 LTS (HVM), SSD Volume AMI was used for this instance to be compatible with ownCloud 10.5.

HTTP, HTTPS and SSH are permitted access from anywhere.

Per owncloud system requirements for small workgroups for up to 150 users (<https://doc.owncloud.com/server/next/admin_manual/installation/deployment_recommendations.html#recommended-system-requirements>) it is recommended to have storage size from 100 GB to 10 TB, but they included the database on the same host. We are putting the database on another host on a private subnet for security so we don’t need the full amount of storage space on this host that is recommended. Per their system recommendations, only one machine is required to handle the load so no auto scaling group or load balancer would be needed at this time.

with open('./bootstrap/owncloud-server.sh') as f:

instance\_id = ec2\_client.run\_instances(

    ImageId             = 'ami-0aa2b7722dc1b5612',

    InstanceType        = 't2.micro',

    KeyName             = 'glkey',

    MaxCount            = 1,

    MinCount            = 1,

    UserData            = f.read(),

    SecurityGroupIds    = [ public\_sg.id ],

    SubnetId            = public\_subnet.id,

    BlockDeviceMappings = [{

        'DeviceName':       '/dev/sda1',

        'Ebs': {

            'VolumeSize':   30

        }

    }],

    TagSpecifications   = [{

        'ResourceType': 'instance',

        'Tags'        : [

            {'Key': 'Name', 'Value': owncloud-server'}

        ]

    }]

)['Instances'] [0]['InstanceId']

owncloud\_server = ec2\_resource.Instance(instance\_id)

bootstrap script: ./bootstrap/owncloud-server.sh

#!/bin/bash

apt-get update

apt-get upgrade -y

apt install mysql-client-core-8.0 -y

apt-get install apache2 -y

apt-get install php7.4 php7.4-cli libapache2-mod-php php7.4-mysql -y

curl https://attic.owncloud.org/download/repositories/10.5/Ubuntu\_20.04/Release.key | apt-key add -

echo 'deb http://attic.owncloud.org/download/repositories/10.5/Ubuntu\_20.04/ /' | tee /etc/apt/sources.list.d/owncloud.list

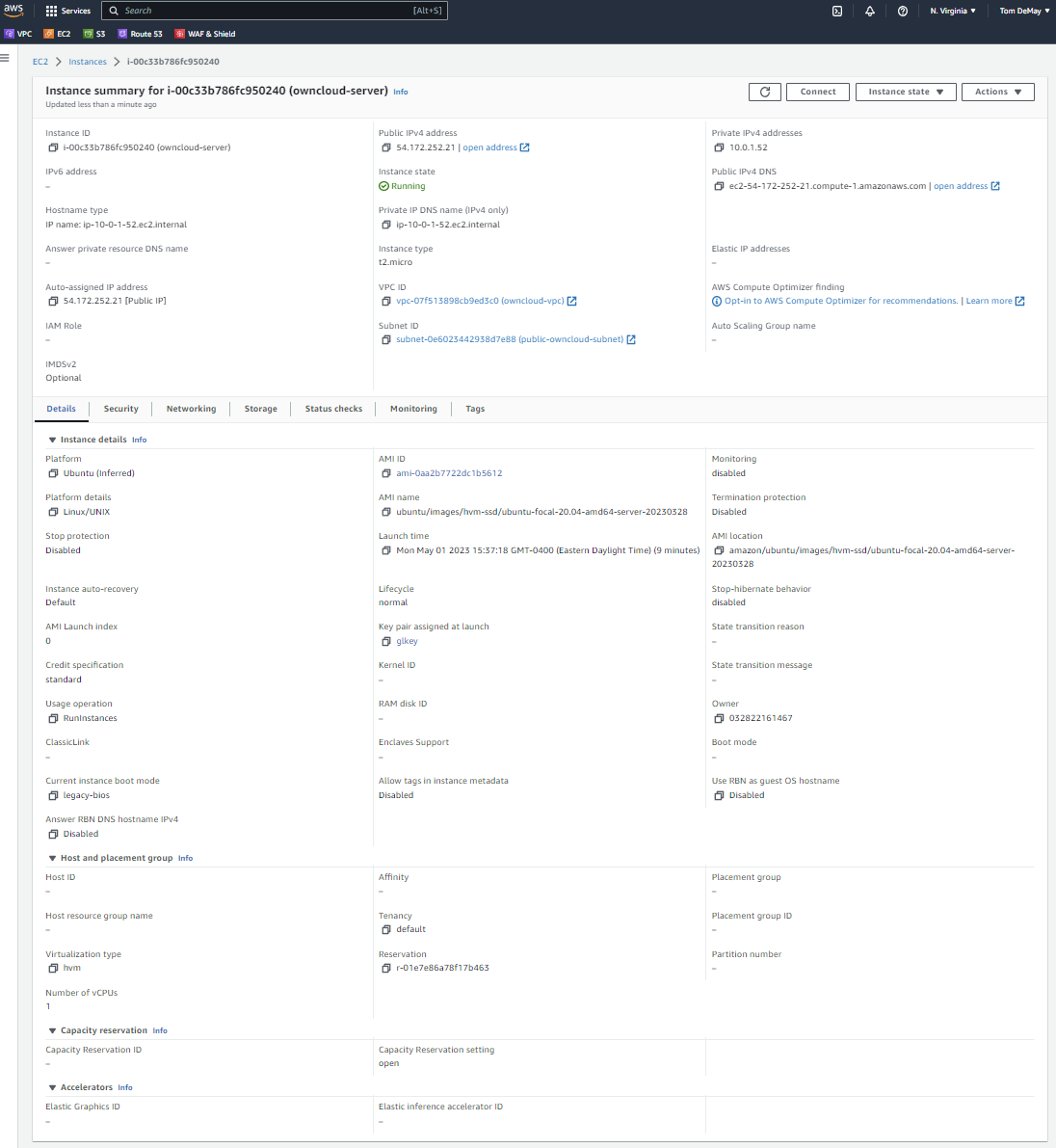
apt-get update

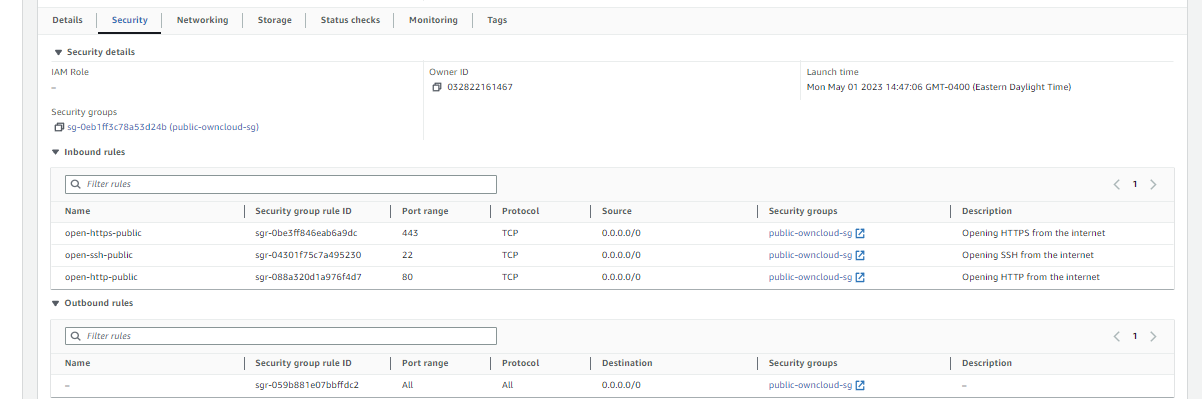
apt-get install php7.4-bz2 php7.4-curl php7.4-gd php7.4-imagick php7.4-intl php7.4-mbstring php7.4-xml php7.4-zip owncloud-files -y

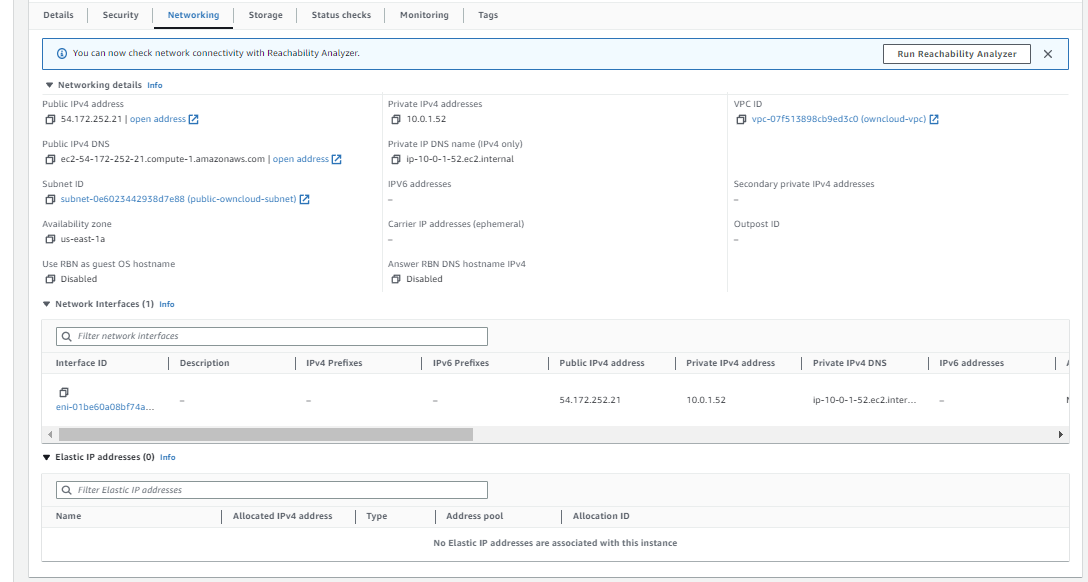
sed -i "s/DirectoryIndex /DirectoryIndex index.php /" /etc/apache2/mods-enabled/dir.conf

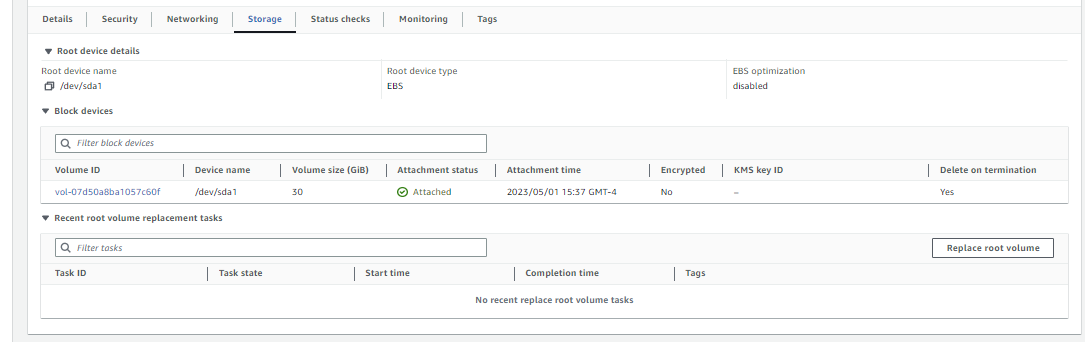
sed -i "s|DocumentRoot /var/www/html|DocumentRoot /var/www/owncloud|" /etc/apache2/sites-enabled/000-default.conf

systemctl restart apache2









### EC2 Instance to host MySQL

Name: mysql-server  
Description: This EC2 instance hosts the MySQL server instance. It is currently accepting SSH and MySQL traffic from the public subnet only to better secure the database. Once in production SSH can be dropped to further secure this. In fact since everything is being configured by a bootstrap scripts SSH is not needed at all.

I chose 30 GiB SSD EBS file system, because 30 GiB is the maximum for free tier limit, but if this was production I would go with something much higher, perhaps 1 TiB. EBS can be scaled up in minutes if needed. EBS supports up to 64 TiB if needed. Owncloud recommends 100 GB to 10 TB for total storage, including the web application and database. But does not specify any recommendations if using a separate EC2 Instance for the database.

with open('./bootstrap/mysql-server.sh') as f:

instance\_id = ec2\_client.run\_instances(

    ImageId             = 'ami-0aa2b7722dc1b5612',

    InstanceType        = 't2.micro',

    KeyName             = 'glkey',

    MaxCount            = 1,

    MinCount            = 1,

    UserData            = f.read(),

    SecurityGroupIds    = [ private\_sg.id ],

    SubnetId            = private\_subnet.id,

    BlockDeviceMappings = [{

        'DeviceName':       '/dev/sda1',

        'Ebs': {

            'VolumeSize':   30

        }

    }],

    TagSpecifications   =[{

        'ResourceType': 'instance',

        'Tags'        : [

            {'Key': 'Name', 'Value': 'mysql-server'}

        ]

    }]

)['Instances'] [0]['InstanceId']

mysql\_server = ec2\_resource.Instance(instance\_id)

bootstrap script: ./bootstrap/mysql-server.sh

#!/bin/bash

apt-get update

apt-get upgrade -y

apt-get install mysql-server -y

systemctl start mysql.service

mysql --user=root <<\_EOF\_

  ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'password';

  DELETE FROM mysql.user WHERE User='';

  DELETE FROM mysql.user WHERE User='root' AND Host NOT IN ('localhost', '127.0.0.1', '::1');

  DROP DATABASE IF EXISTS test;

  DELETE FROM mysql.db WHERE Db='test' OR Db='test\\\_%';

  CREATE USER 'gluser'@'%' IDENTIFIED WITH mysql\_native\_password BY 'password';

  GRANT ALL PRIVILEGES on \*.\* TO 'gluser'@'%' WITH GRANT OPTION;

  FLUSH PRIVILEGES;

\_EOF\_

mysql --user=gluser -ppassword <<\_\_EOF\_\_

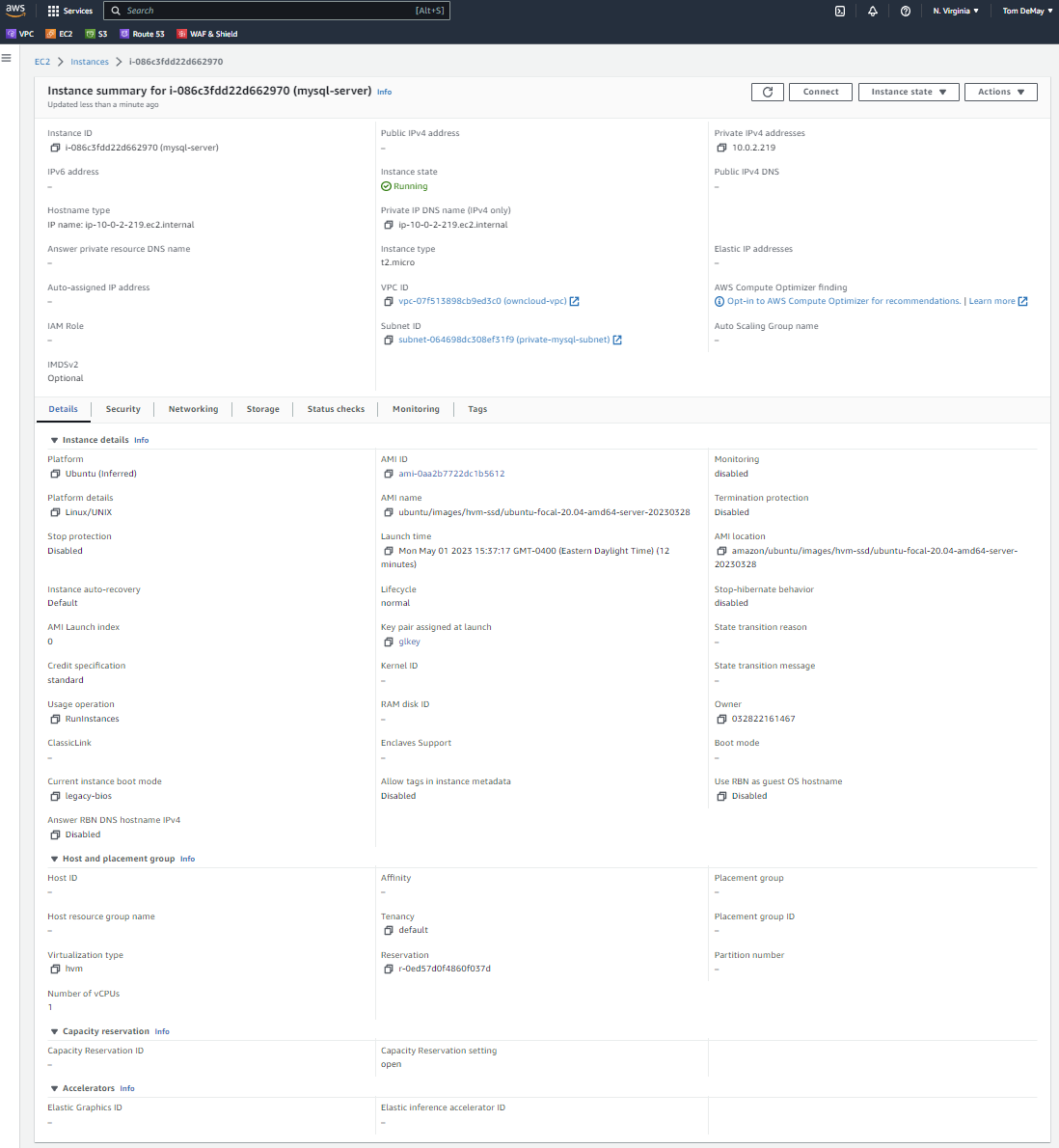
    CREATE DATABASE owncloud\_db;

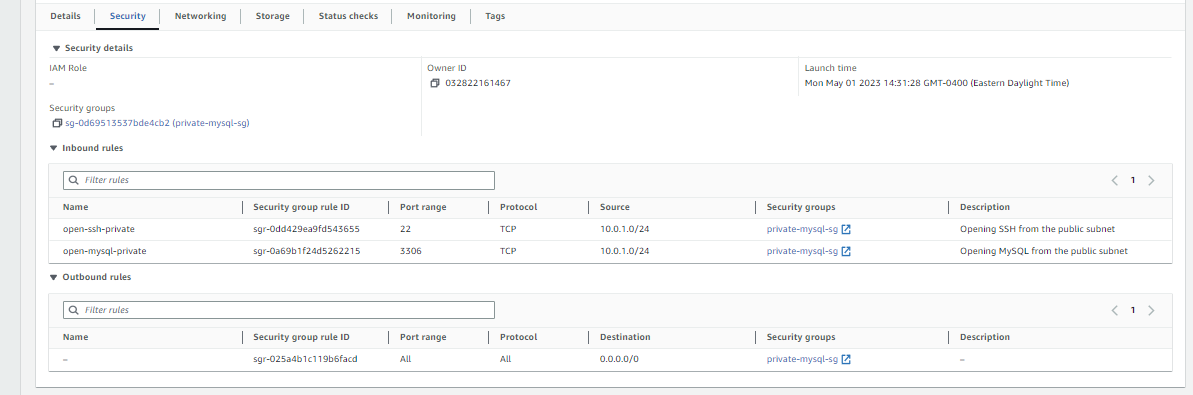
\_\_EOF\_\_

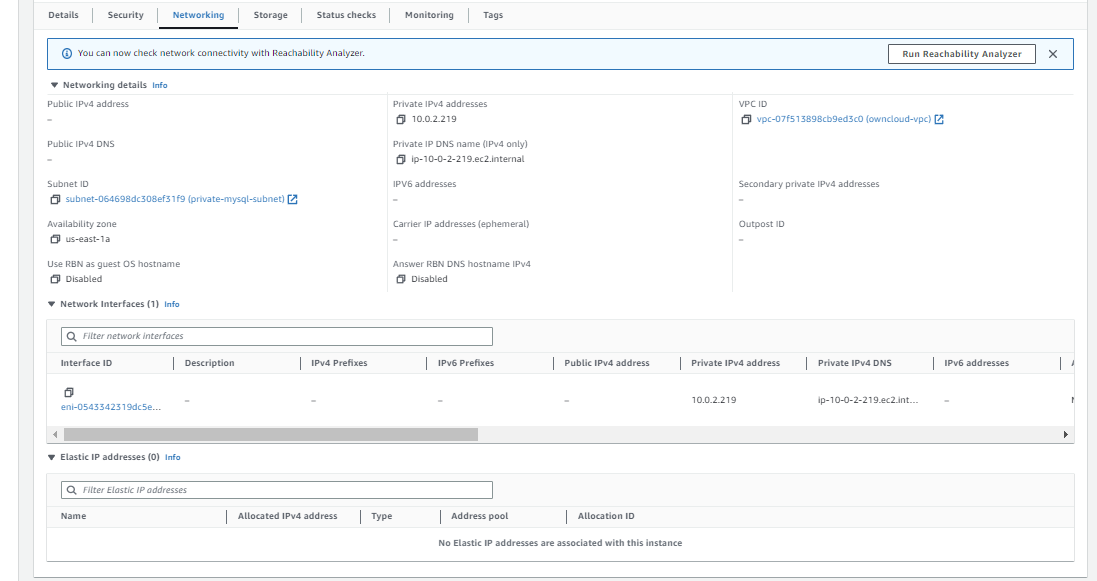
sed -i "s|^bind-address.\*= 127.0.0.1|bind-address = 0.0.0.0|" /etc/mysql/mysql.conf.d/mysqld.cnf

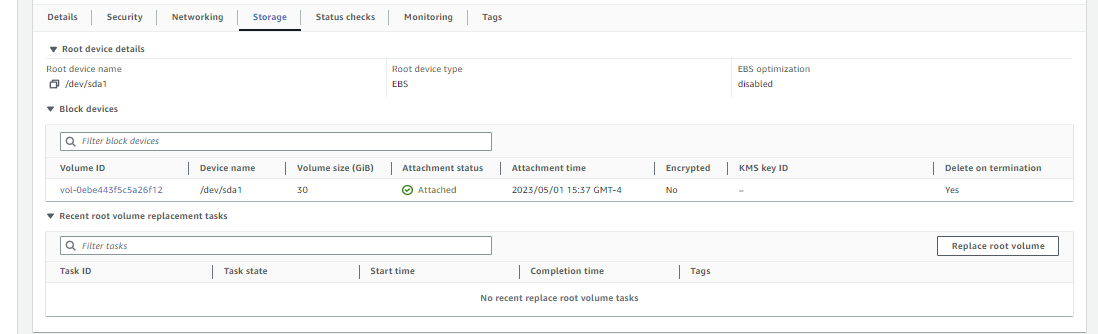
sed -i '/^\[mysqld\]/s/$/\ndefault\_authentication\_plugin= mysql\_native\_password/' /etc/mysql/mysql.conf.d/mysqld.cnf

/etc/init.d/mysql restart



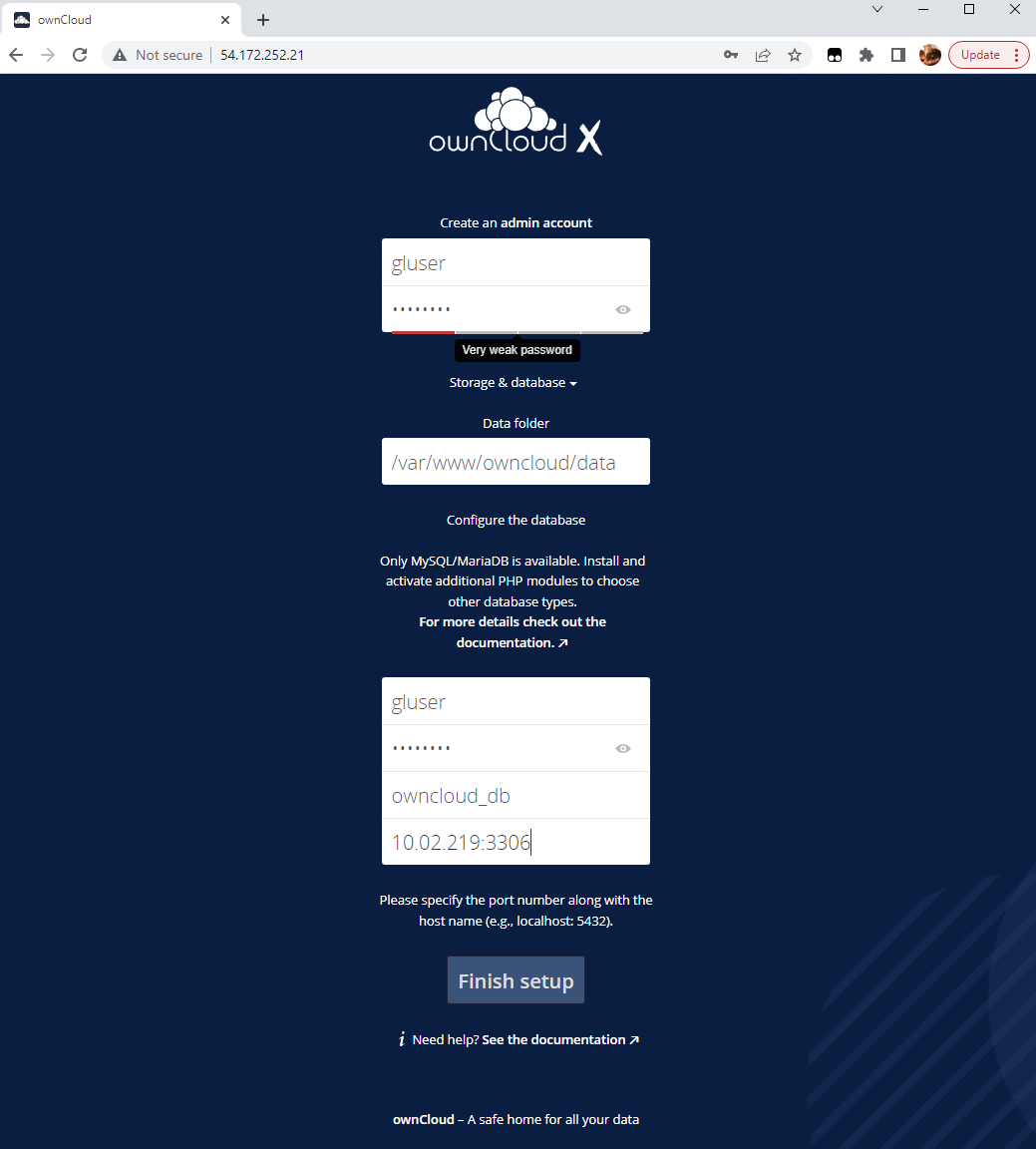




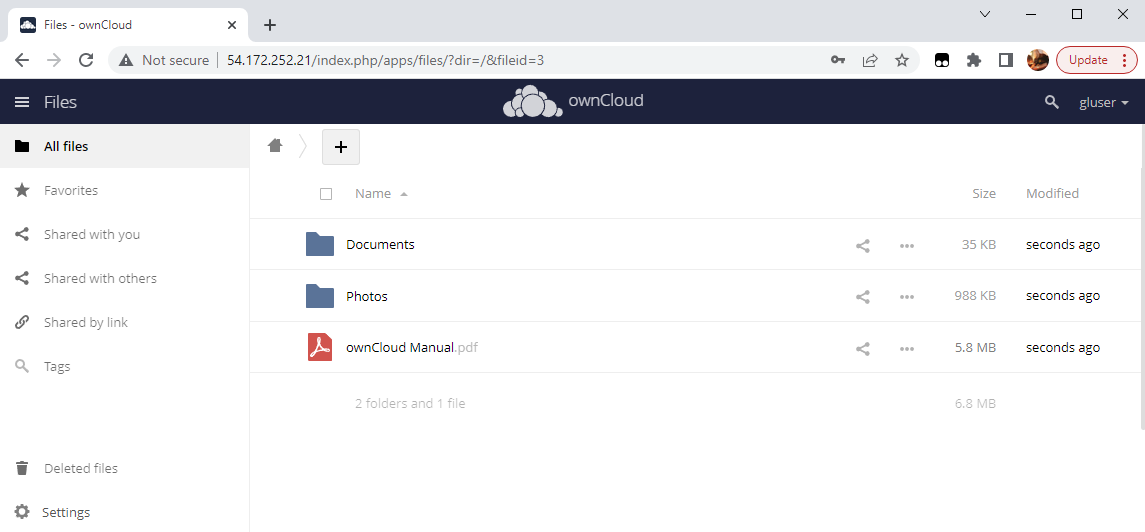


## ownCloud web application

Once the scripts are complete you can navigate to the public IP address of the web application server to access ownCloud web application. The username and password provided to the database are gluser/password. Obviously in production we would want to use a more secure password. We may also want to script setting up the database and admin account for ownCloud to bypass this initial setup screen.

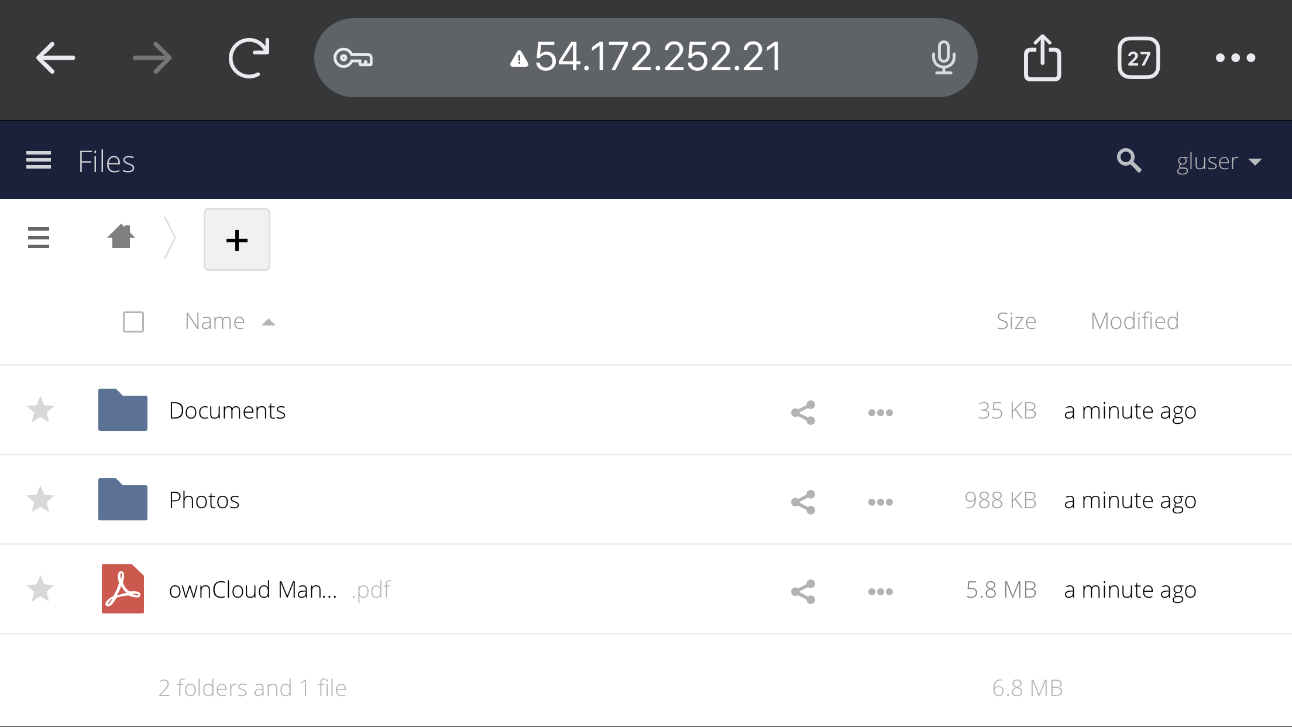


Once the admin account and database are configured, the user is asked to log in again and then taken to the application. This is what is accessible through a windows desktop.



## ownCloud web application view on mobile phone

Description: The web application is available to anyone on the internet at the public IP address or DNS host name assigned to the EC2 instance hosting the ownCloud web application. This is a screen shot of the application on a mobile phone.



## Lessons and Observations:

Our requirements were to support up to 150 users. Actual storage requirements were not available.

Per owncloud deployment recommendations (<https://doc.owncloud.com/server/next/admin_manual/installation/deployment_recommendations.html#recommended-system-requirements>) ownCloud recommends the following for a small workgroup of up to 150 users:

* Ubuntu 20.04
* PHP 7.4
* One machine running the web application, database server as well as local storage.
* Authentication should be done via an existing LDAP or Active Directory server. (not included in this implementation)
* The server should have at least 2 CPU cores and 16GB RAM
* Storage size should be 100 GB to 10TB depending on requirements
* No Load balancer is needed

We went with 2 servers to better protect the database from outside attacks. The MySQL Server is running on one EC2 instance within a private subnet that only has access to it from the public subnet. The web application and local storage is provided on another EC2 instance in the public subnet, accessible from the internet. MySQL server can send any traffic to the internet as needed through a NAT gateway attached to the public subnet.

We went with a ‘t2.micro’ instance for the both EC2 instances for demonstration purposes because it’s provided in the free tier. But t2.large or t2.xlarge should be used as t2.micro, with 1 CPU and 1 GiB of memory does not satisfy the ownCloud deployment recommendations of 2 CPUs and 16GB of memory. t2.large provides 2 CPUs with 8 GiB of memory and t2.xlarge provides 4 CPUs with 16 GiB of memory (1 GiB = 1.074 GB). Since MySQL is not running on the same host as the ownCloud web application as recommended, t2.large might be enough for both EC2 instances to save costs. Either EC2 instance can be upgraded quickly if need be by detaching the EBS volume from the existing EC2 instance and reattaching it to a new larger EC2 instance. However, this would require a few minutes of downtime to bring up the new host, reconfigure and verify everything is in working order before making it available.

A 30 GiB SSD Amazon EBS was used for demonstration purposes because it’s the maximum limit provided in the free tier. If this was an actual deployment, 1 TiB might be enough to start off with for the web hosts and probably 1 TiB for the database server. EBS includes replication within the Availability Zone and 99.999% durability and can be scaled quickly by taking a snapshot of the existing volume and detaching it and attaching a larger volume to the EC2 Instance using the snapshot.

If our requirements grow and we decide to upgrade our architecture to support more users, ownCloud deployment recommends the following for 150 to 5,000 users for a mid-sized enterprise deployment:

* RedHat Enterprise Linux and SUSE Linux Enterprise Server 12+
* PHP 7.4
* 2 to 5 application servers
* A cluster of two database servers
* Storage size up to 200TB on a separate NFS Server
* Authentication via an existing LDP or Active Directory server
* A Redis server for file locking

To accomplish this we can start with a target group of 2 hosts with auto scaling up to 5 with a load balancer for the web application EC2 Instance. We can use two EC2 Instances to host a MySQL/MariaDB Galera cluster. Amazon RDS may be a better option as it is a fully managed service which includes the following:

* Security and compliance
* Performance and scalability
* Automated patching and upgrades
* Data durability and redundancy
* Monitoring and Alerting
* Backup and Recovery

AWS Database Migration Service can be used to migrate the databases with minimal downtime.

Storage would have to be moved from an EBS Volume to a separate NFS Server that all application servers can access to share files. Amazon Elastic File System would be a good choice as it is an AWS managed server that offers the following:

* Easy to create and configure. No provisioning, deploying, patching or maintenance required
* It is fully elastic so no capacity planning is required
* You pay for only the storage you use.
* You can reduce costs by up to 92 percent by automatically moving infrequently access files using Amazon EFS Intelligent-Tiering
* Fully managed file system designed for 99.999999999% durability and 99.99% of availability
* A redis server will be required to handle file locking.

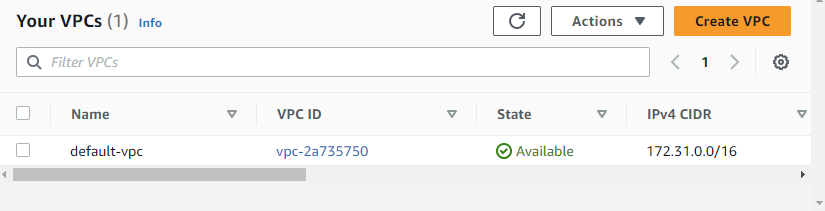
An AWS Network Firewall can be used to further secure the application from various attacks, such as denial of service attacks, etc. Depending on government regulations various firewall policies may be mandatory, such as financial institutions. AWS Network Firewall automatically scales and supports thousands of custom rules.

AWS Direct connect could be used to keep all the data in-house in the enterprises own data center. Using AWS Outposts you can host an appliance in your enterprise’s data center that will provide some AWS managed services on premises.

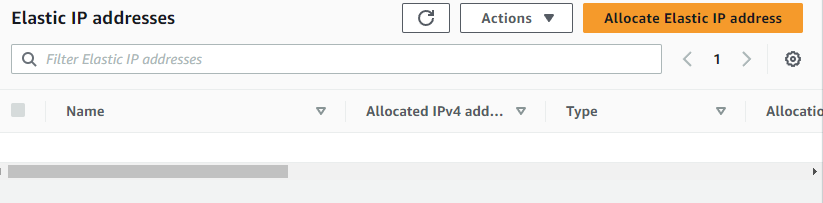
## Cleanup

The following screen shots demonstrate the cleanup scripts have successfully cleaned up the resource.

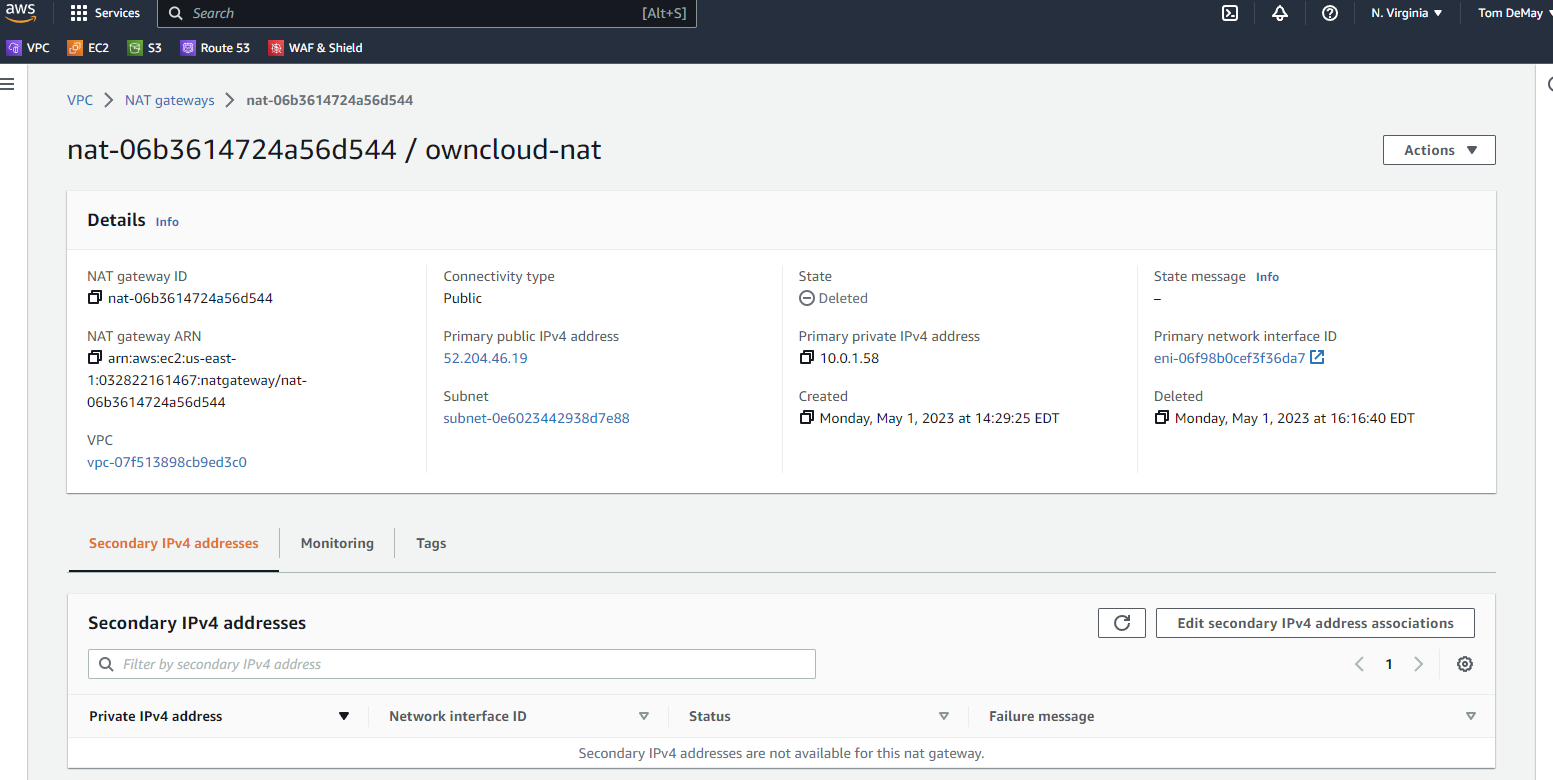
owncloud-vpc has been deleted. Deleting the VPC also requires dependent objects to be deleted too. This screen shot confirms the VPC, EC2 instances, subnets and security groups have been deleted. The default VPC is the only VPC left.



Elastic IP address has been released



NAT gateway has been deleted



Internet gateway has been deleted. Only one remaining is the internet gateway for the default VPC

