GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4150-A Fall 2021
Lab: Photo Gallery Application - SQL and NoSQL Variants

References:

[1] A. Bahga, V. Madisetti, "Cloud Computing Solutions Architect: A Hands-On Approach", ISBN: 978-0996025591

[2] https://aws.amazon.com/documentation/

Due Date:

The lab report will be **due on November 5, 2021 at 11:59 PM.**

In this lab, we will create a Photo Gallery application composed of albums and photos using two variations to store records of photos: SQL and NoSQL. We are going to integrate the same method we used in the previous lab.

In the SQL variant of the application, the records of albums and photos are maintained in a MySQL database instance on Amazon RDS. Whereas in the NoSQL variant of the application, the records of photos are held in a DynamoDB table. This application is implemented in Python and uses the Flask web framework. We are going to deploy the application on an Amazon EC2 instance.

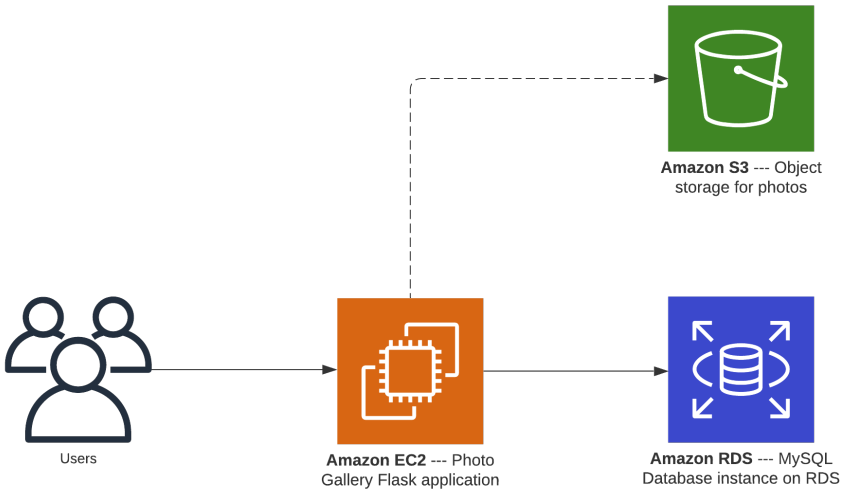


Fig.1 Architecture diagram of the Photo Gallery application - SQL Variant

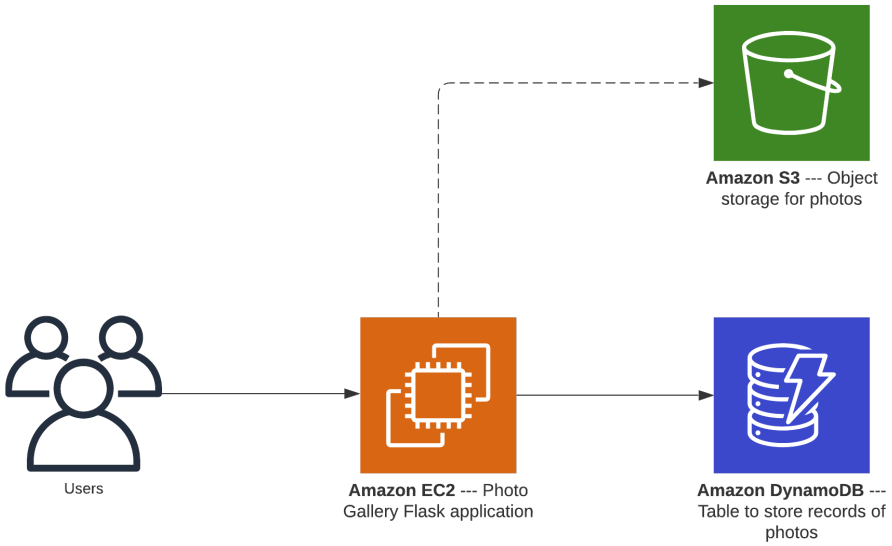
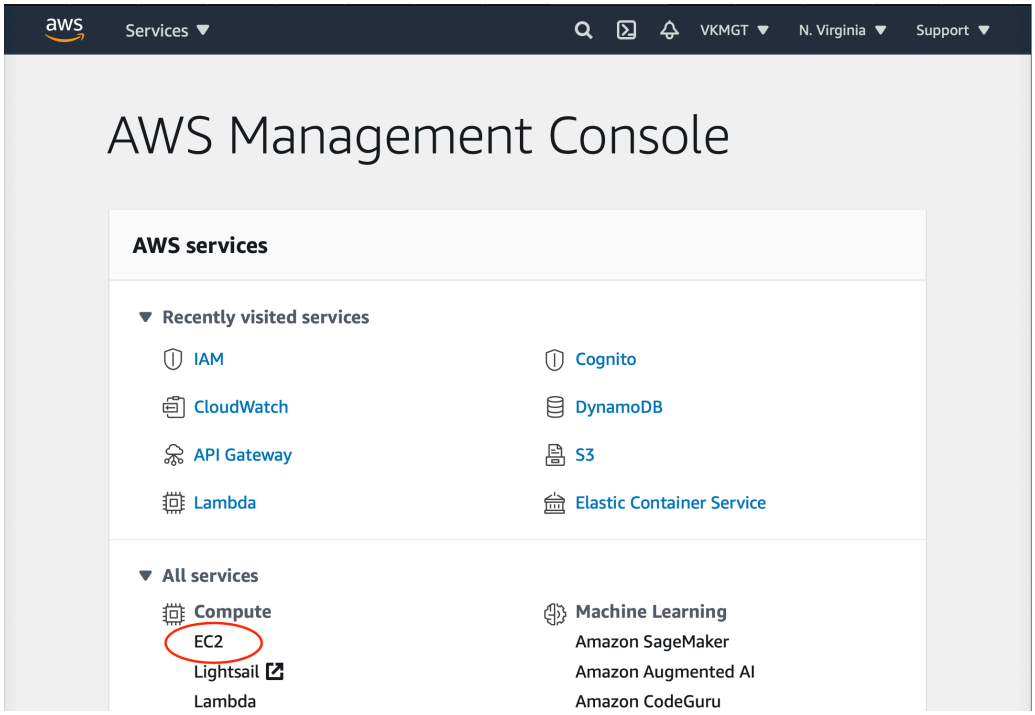


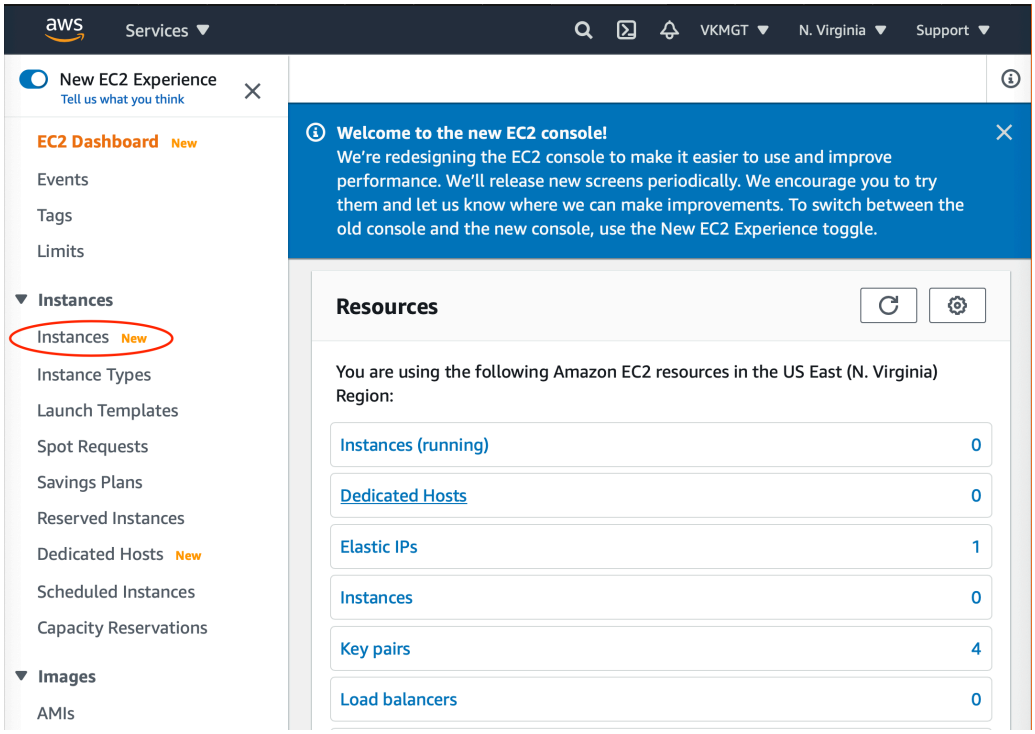
Fig. 2 Architecture diagram of the Photo Gallery application - SQL Variant

1. Create EC2 instance for hosting the static website and resources (10 points)

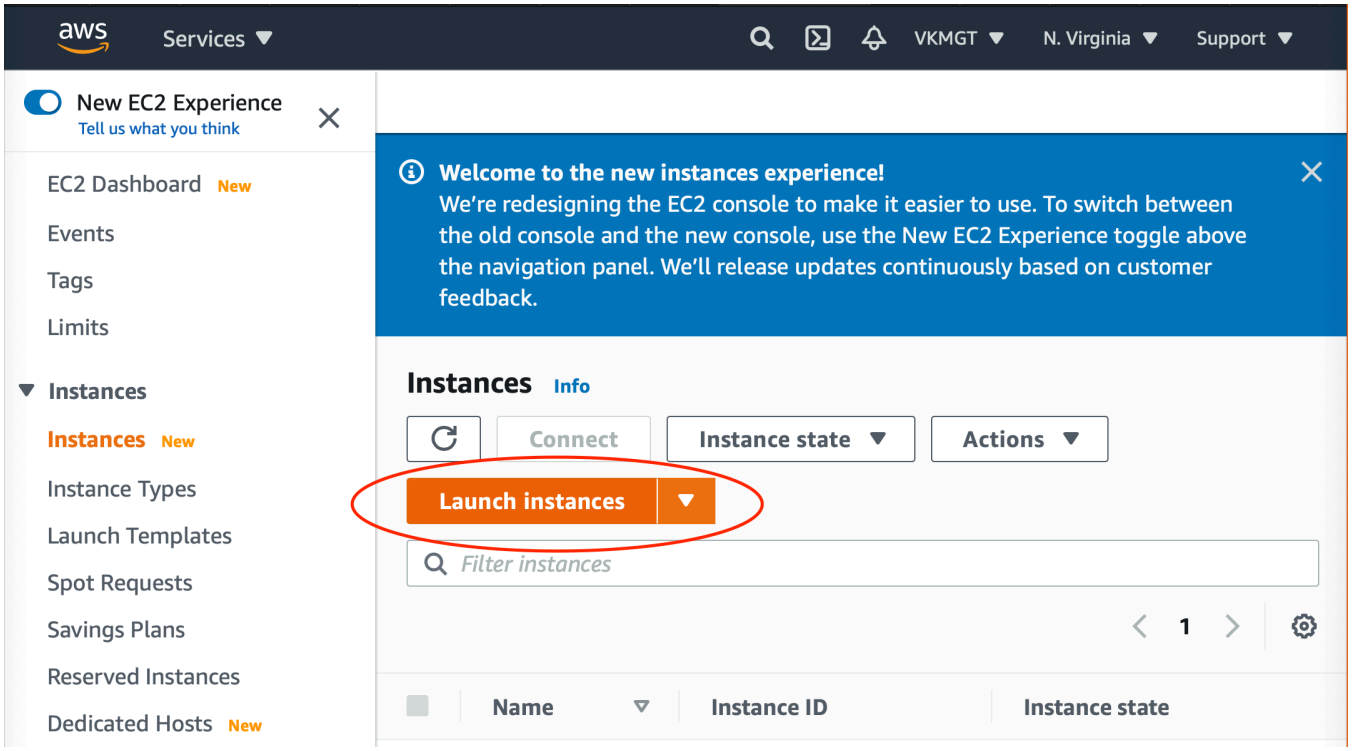
- In the AWS Management Console, under Compute, click **EC2**. You can also use the search input in the navigation bar by typing "**EC2**"



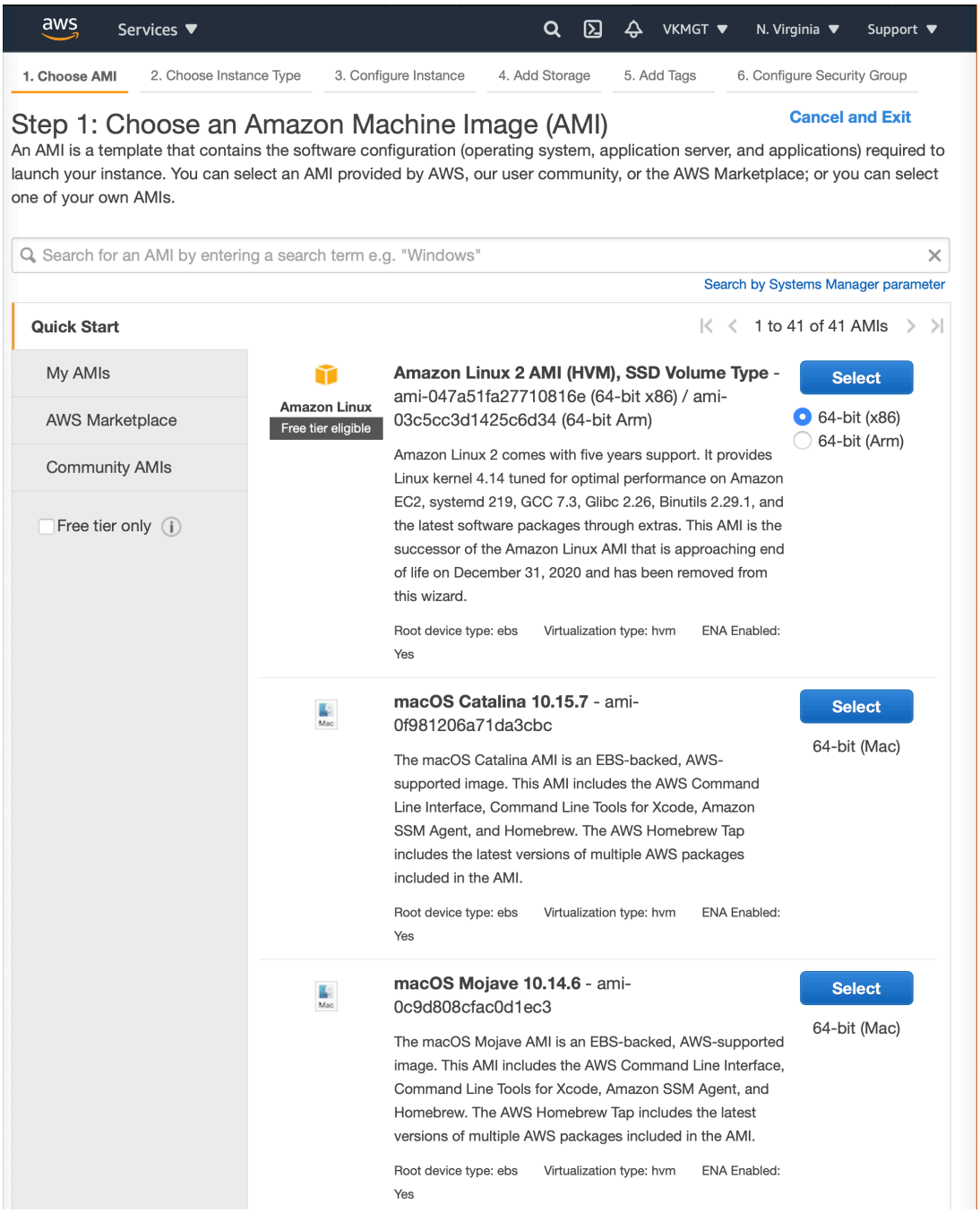
- Once in the EC2 console, click **Instances** under the instances dropdown menu in the navigation bar on the left



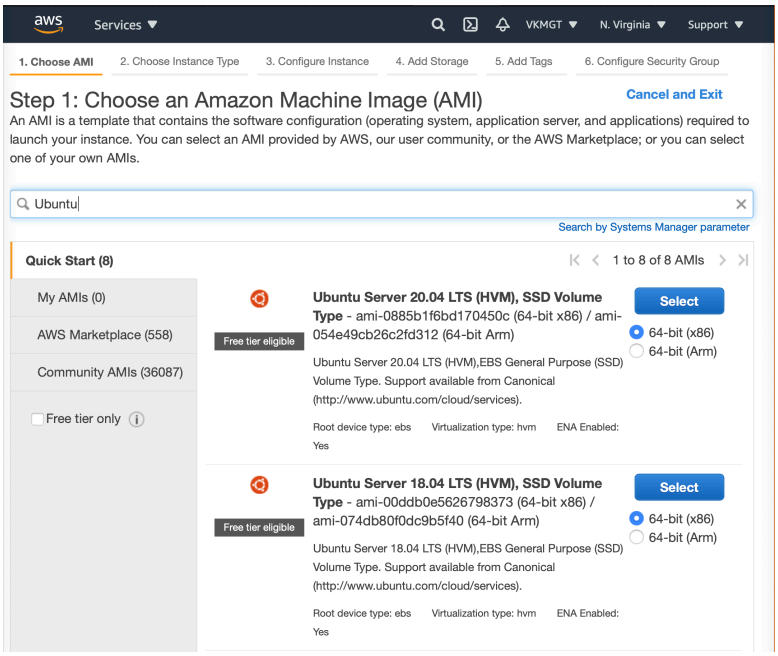
- Lunch an instance by clicking the orange button "**Launch instances**"



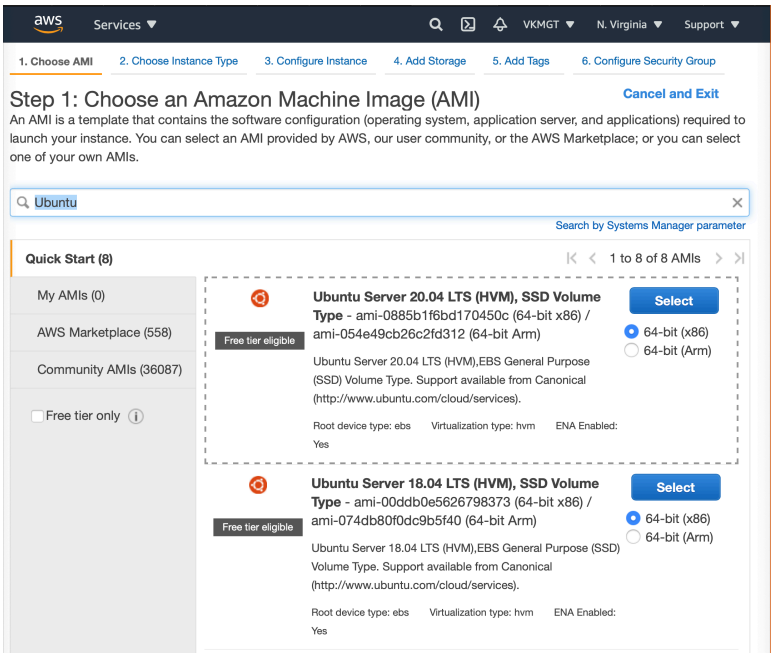
- In the first step to launching an instance, you have the option to choose the **Amazon Machine Image (AMI)** that you would like to use for your server instance. Go through the list to get familiar with the images that AWS offers. For our lab, we are going to use an Ubuntu image to run our flask server



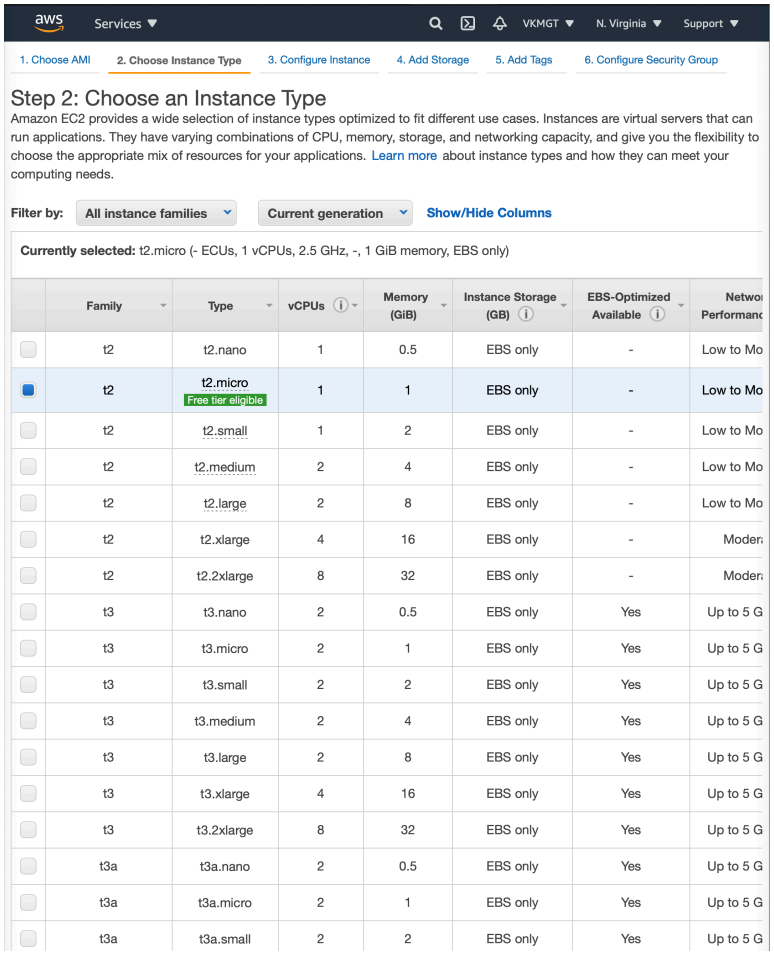
- To find the right Ubuntu image, type "**Ubuntu**" in the search input and find the most recent Ubuntu version. In this given time, the most recent snapshot is **20.04**



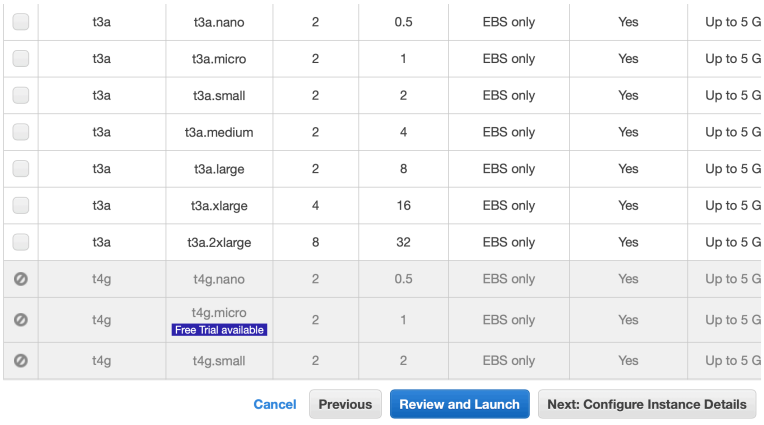
- Select the **Ubuntu Server 20.04 LTS (HVM), SSD Volume Type** image



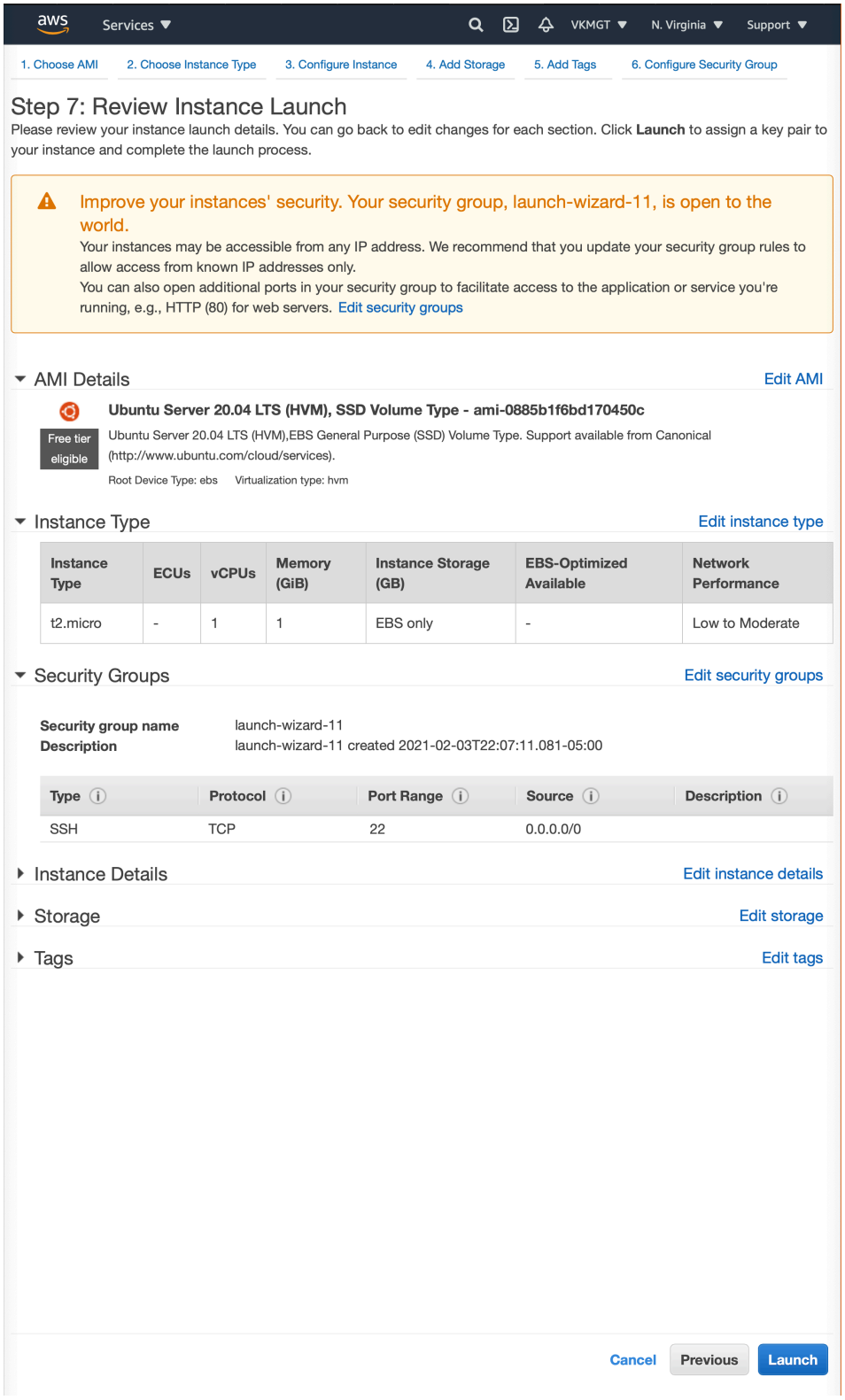
- In the second step, we can select the type of instance that we would like to use. For our lab, we will choose the **t2.micro** since it is the free tier eligible version



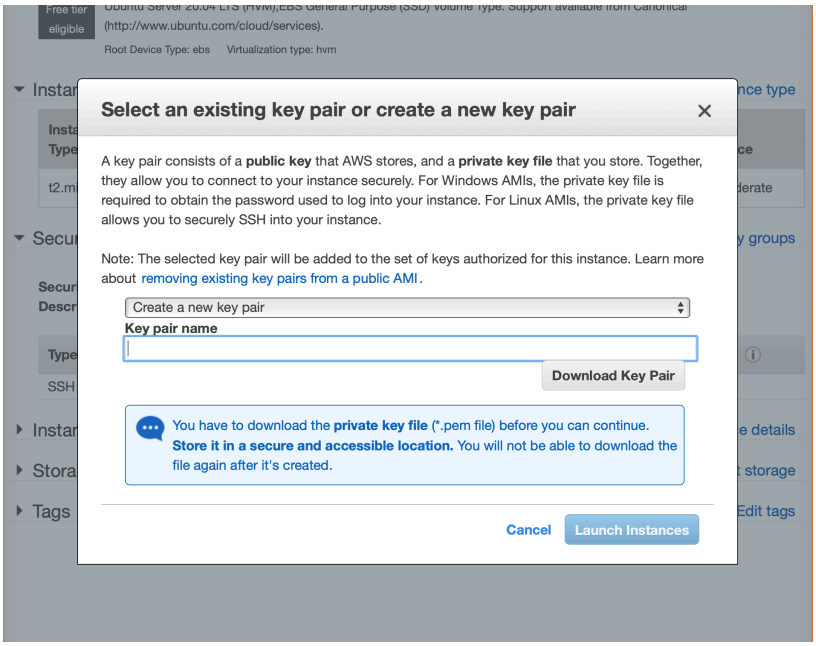
- After selecting the instance type, click "**Review and Launch**"

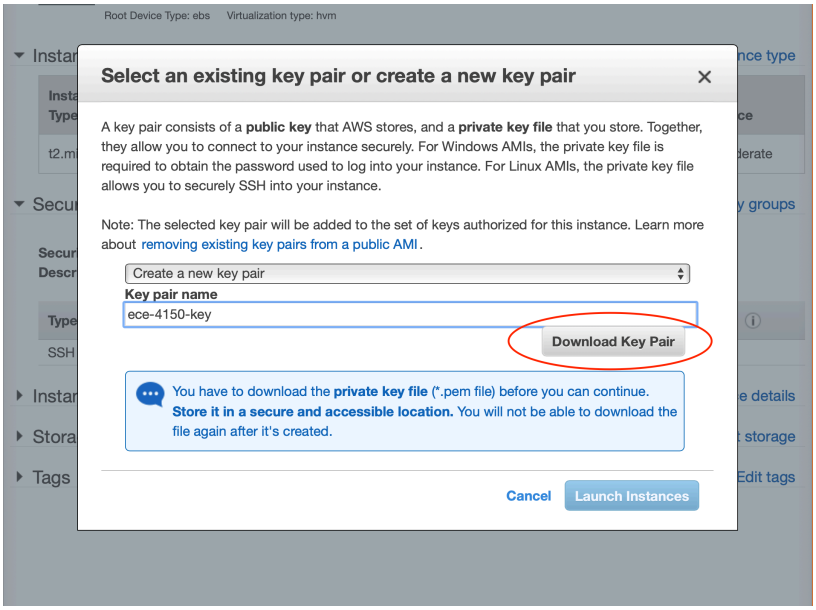


- The last step is to review the configuration of the instance (e.g., AMI, instance type, and other details that we will create by default as part of this lab). Once you review the instance configuration, click "**Launch**"

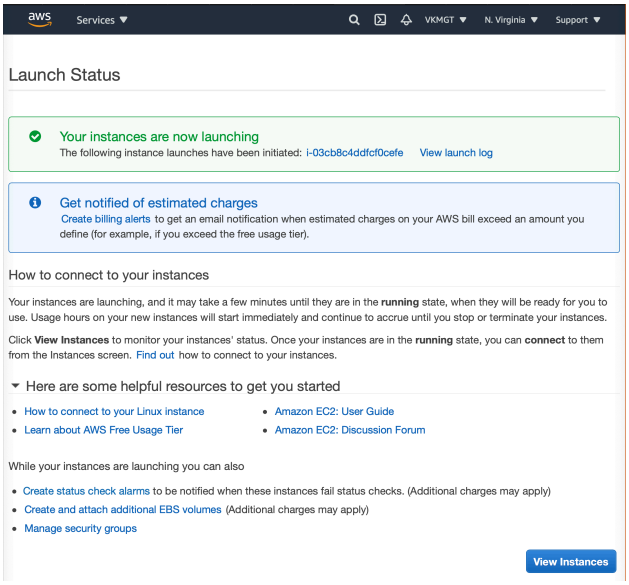


- After you clicked the Launch button, you will have the option to create a new key pair or attach it to an existing key pair. To access an instance, AWS gives you a private key that you can use to access the instance remotely. To access your instance, you will use the SSH protocol to connect via the terminal or PowerShell
- To create a new one, select "**Create a new key pair**" and name the key pair. Once you placed the key's name, download the key by clicking the "**Download Key Pair**" button. After you download the Key Pair, you will be allowed to launch the instance. **Ensure you store that key in a secure place because you won't have access to the instance again if you lose it**

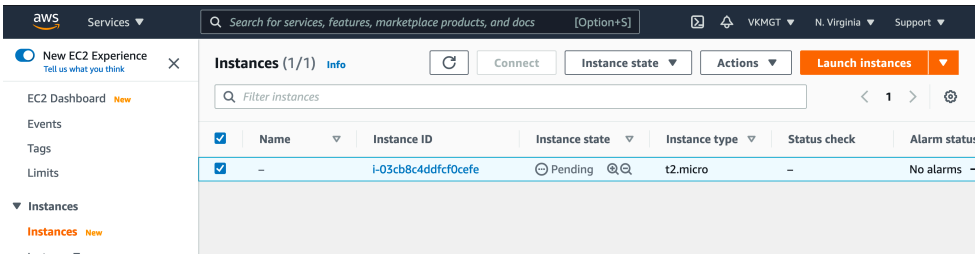




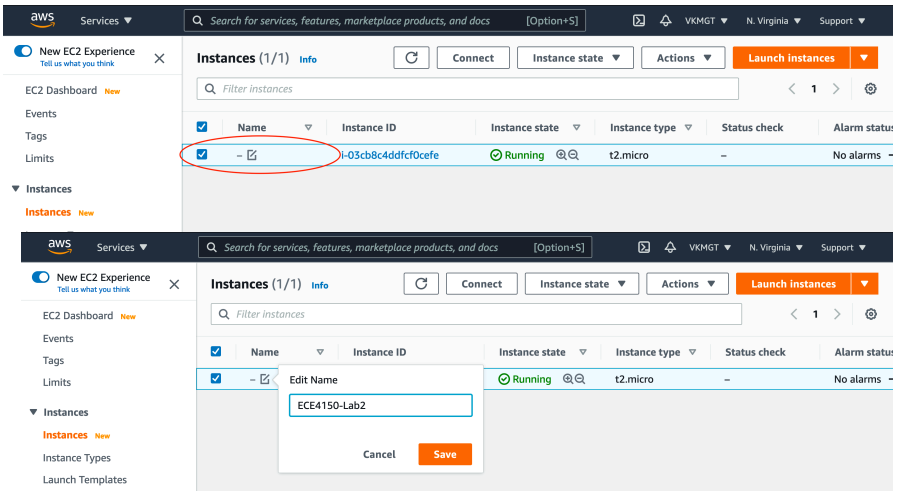
- It might take a few seconds or minutes for the server instance to be available. However, you can view the instance status by clicking the "**View Instances**" button



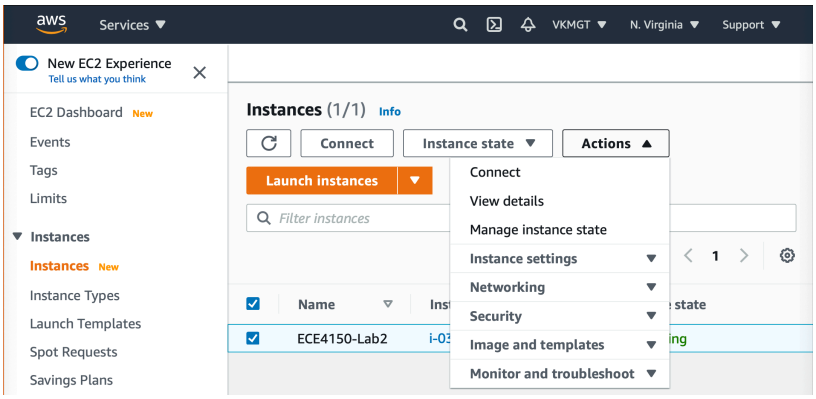
- You should see your instance running, as shown below



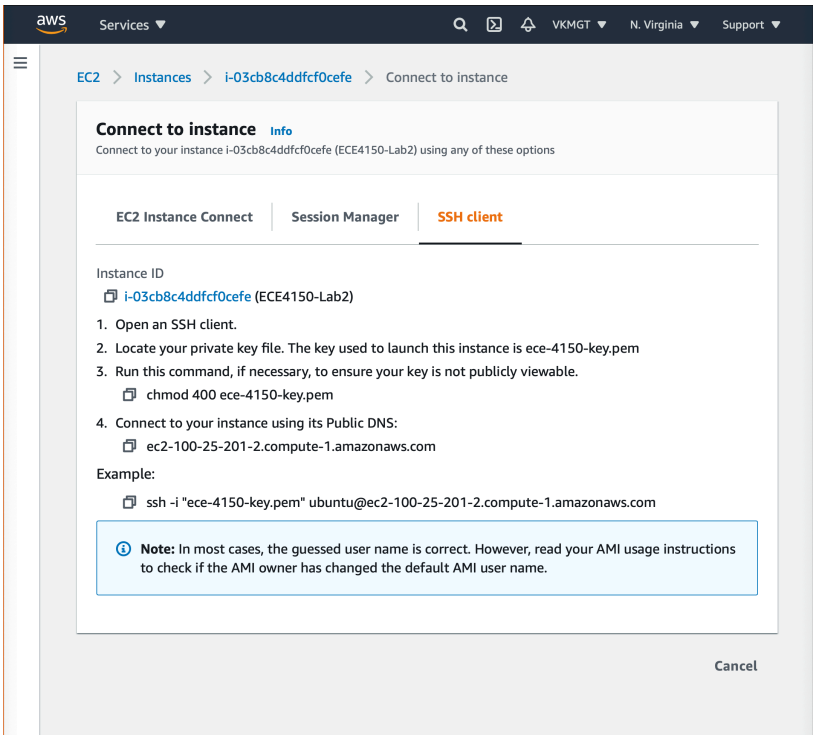
- If you want to change the name of the instance, click in the section below the **Name** column



- To connect to your new server instance, select your instance, click the "**Actions**" dropdown, and click connect

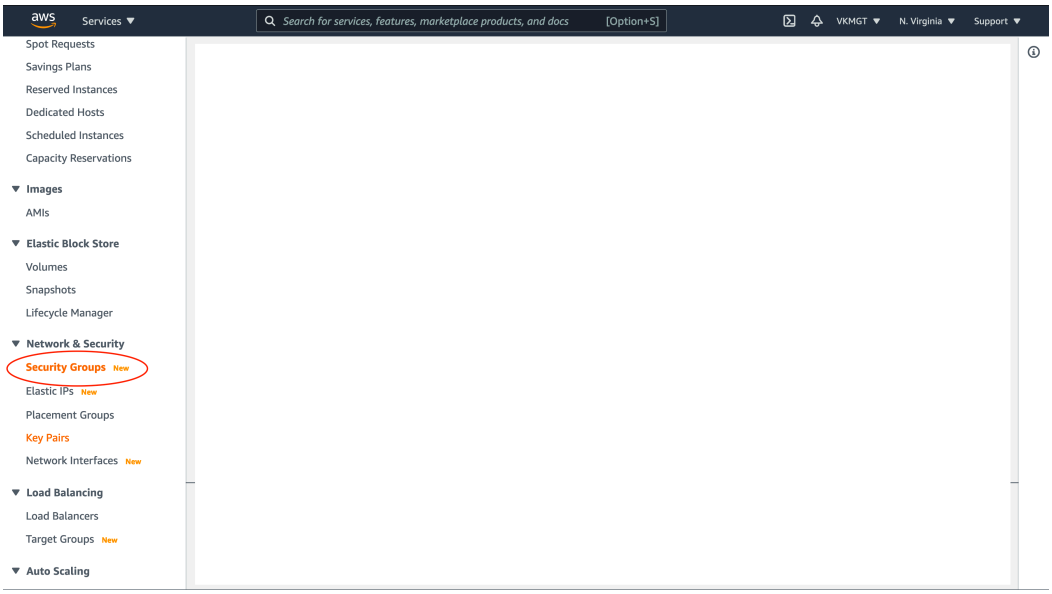


- Under the **SSH client** tab, you will find instructions to understand how to connect to your instance remotely

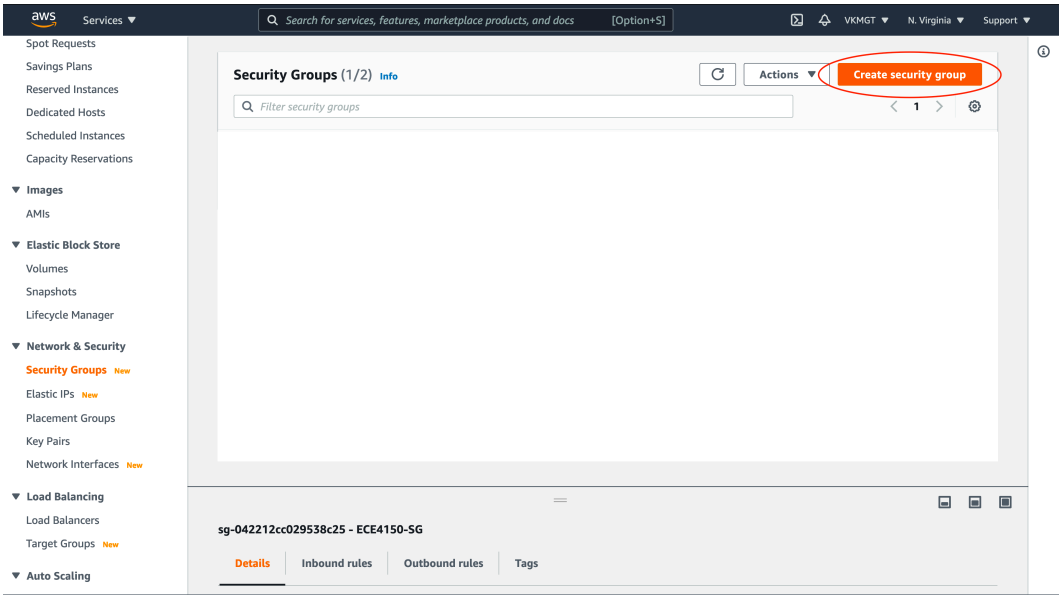


2. **Create a Security Group (SG) to allow access from your EC2 from anywhere or a specific location (4 points)**

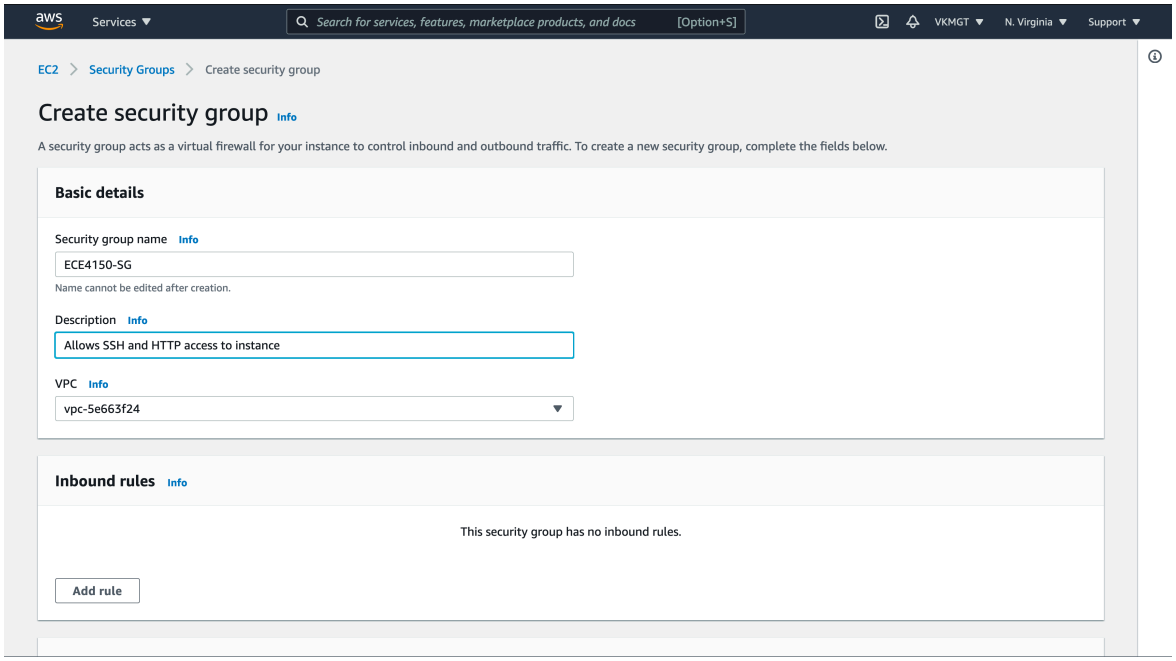
- In your EC2 Console, under **Network & Security**, click **Security Groups**. You can also use the search input in the navigation bar by typing "**Security Groups**" Under "Features"



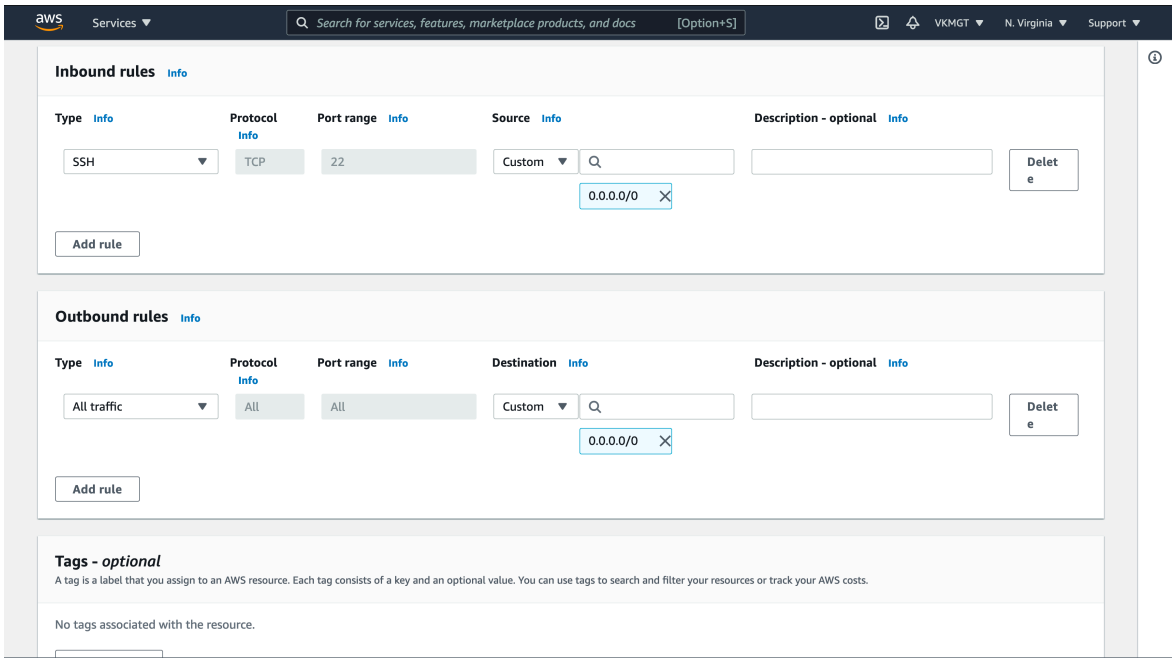
- Create a new Security Group by clicking the orange button called "**Create security group**" in the top right corner



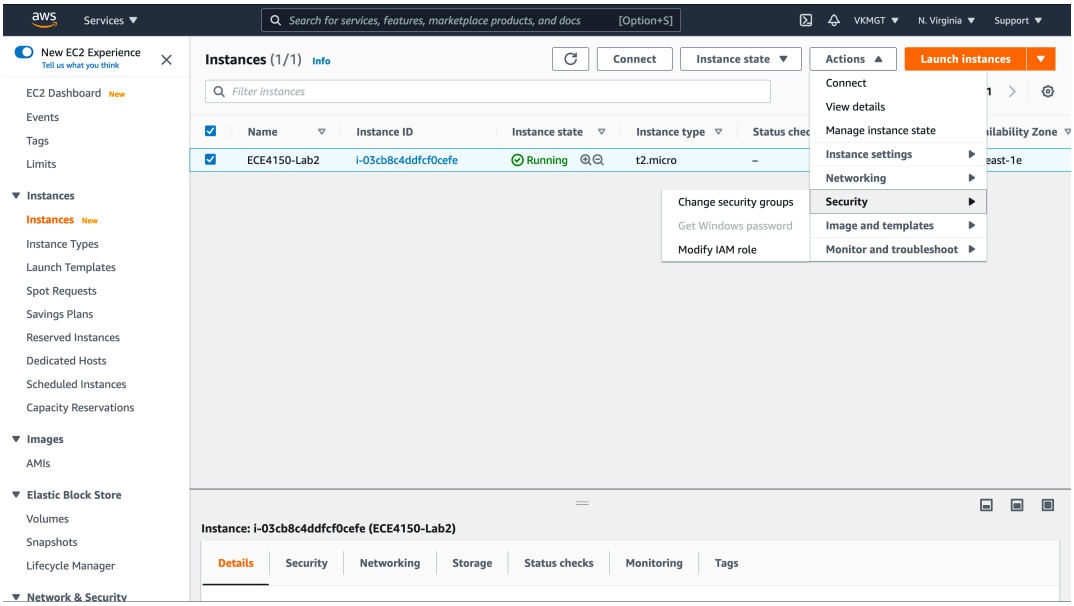
- Add a name and description to the security group as shown below



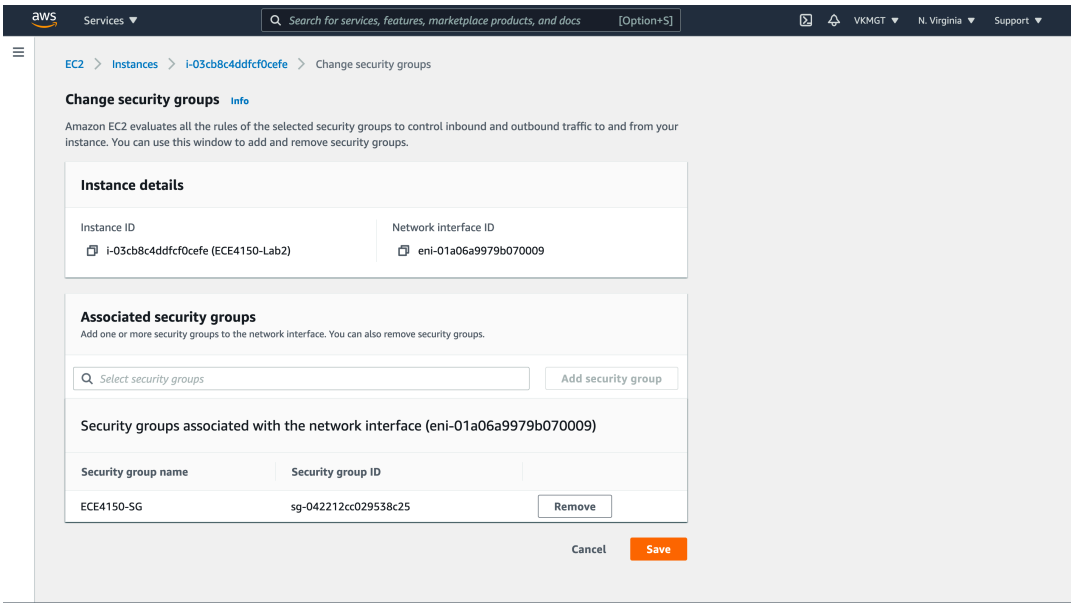
- At his time, we are going to add one single inbound rule. We will allow access through SSH from anywhere, as shown below. Do not change the configuration for the outbound rule



- Attached the new security group to the EC2 instance. Click "**instances**" in the left navigation menu. Select your server instance, and under **Actions**, click **Security** and **Change security groups**

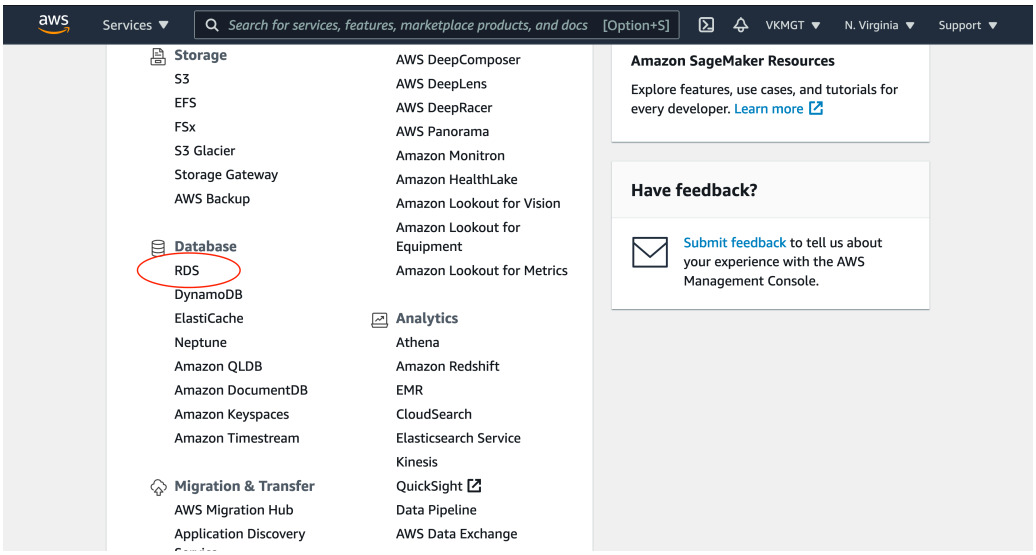


- Remove the attached security group and add your new security group, then click "**Save**"

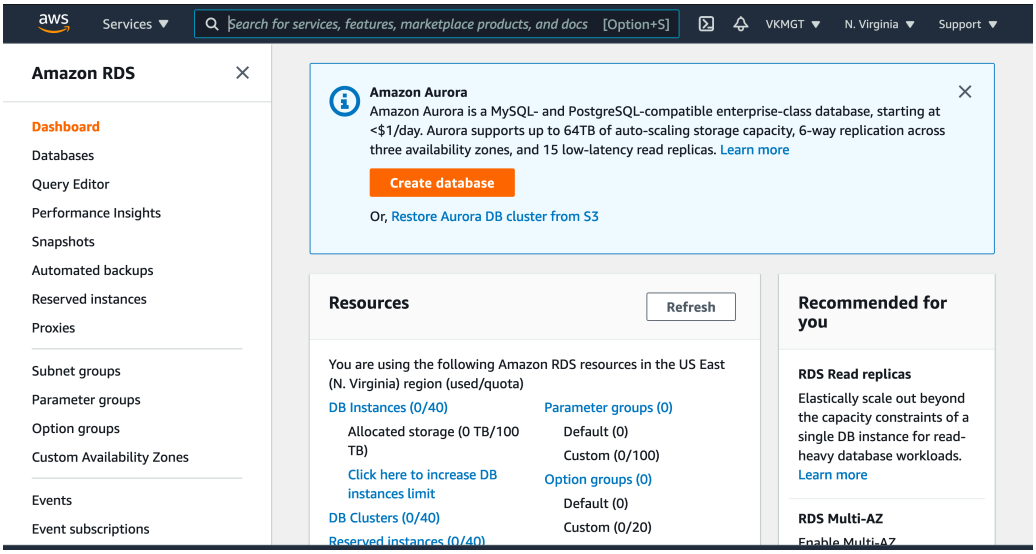


3. Create an RDS database instance for storing records of photos (10 points)

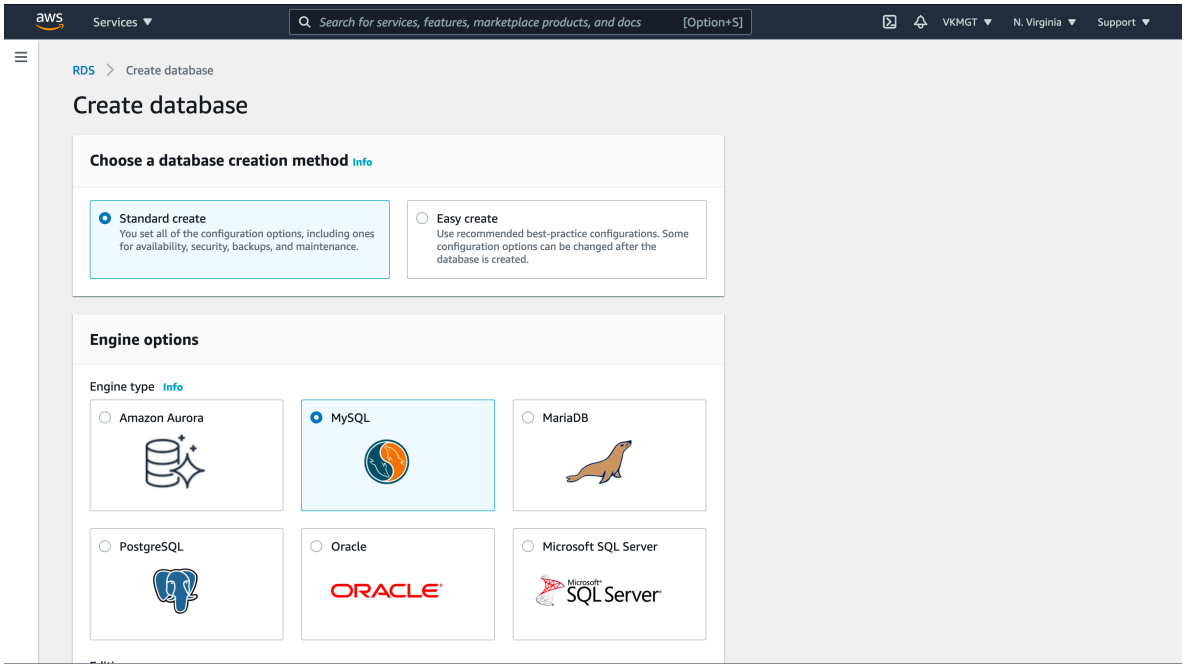
- In the AWS Management Console, under Database, click **RDS**. You can also use the search input in the navigation bar by typing "**RDS**"

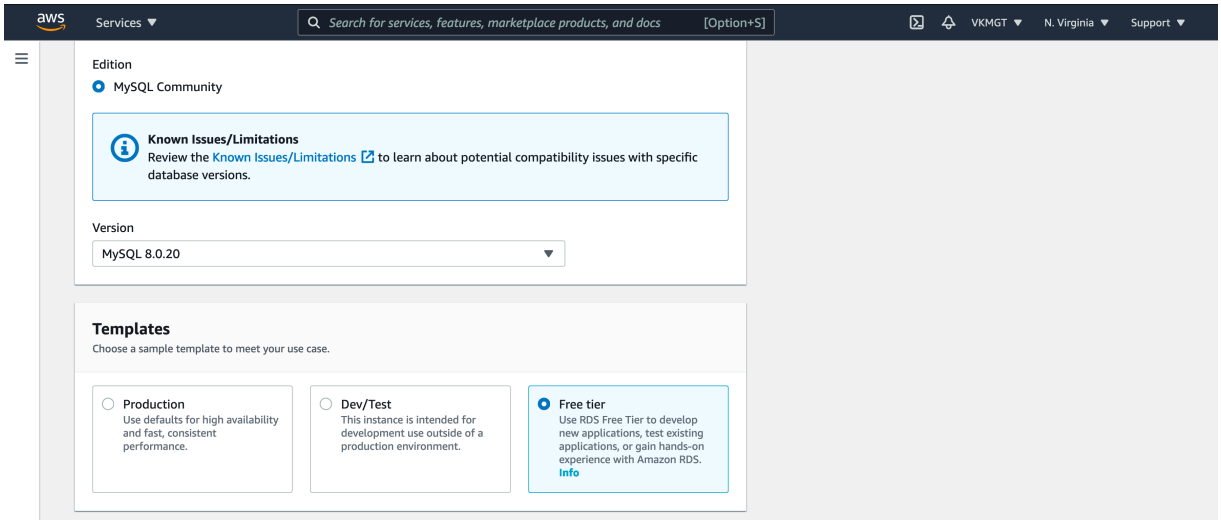


- Lunch a database instance by clicking the orange button "**Create database**"

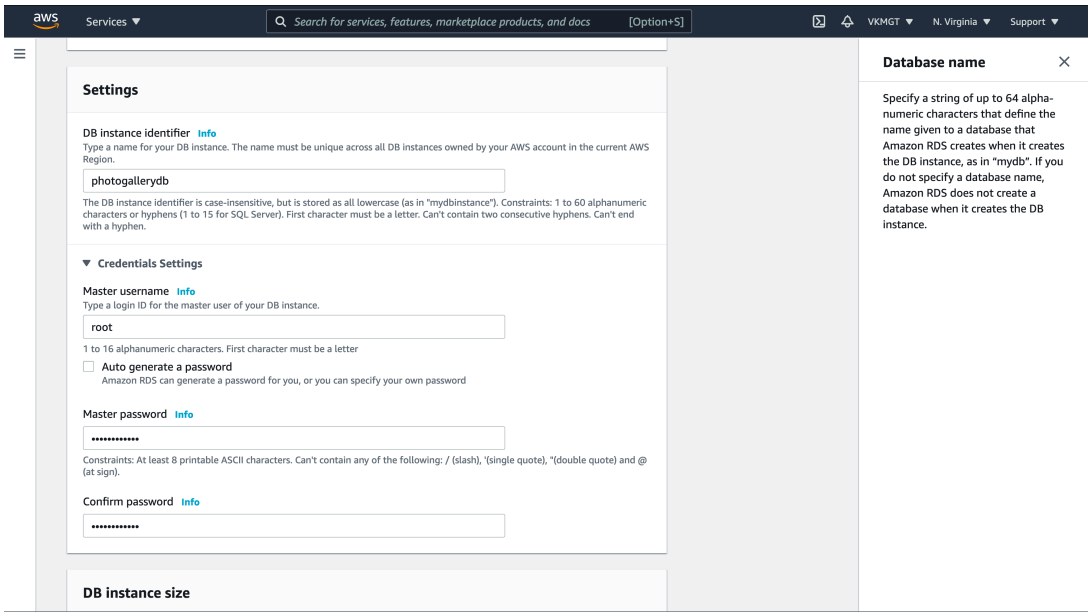


- Follow the configurations as shown below

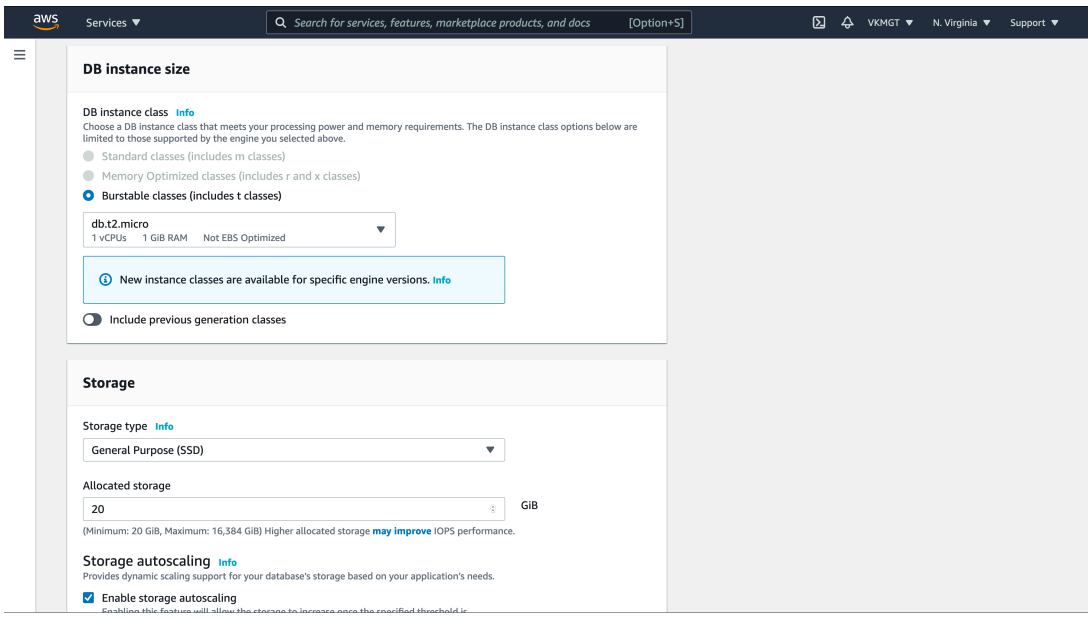


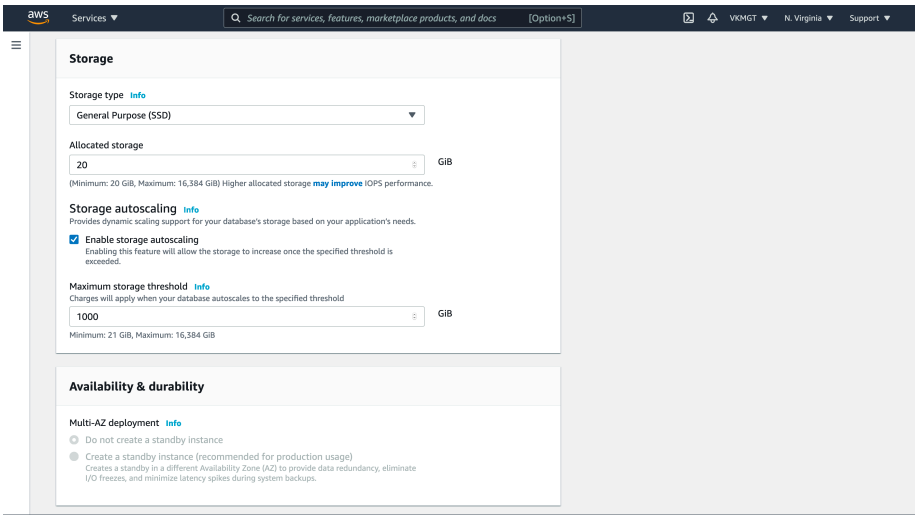


- Under settings, for the “DB instance identifier”, type the name of your DB instance. For this lab, it is going to be **photogallerydb**. For the Master username, type the the login ID use to connect to the database instance; we are going to use **root** for the login ID. Create a password for authentication.

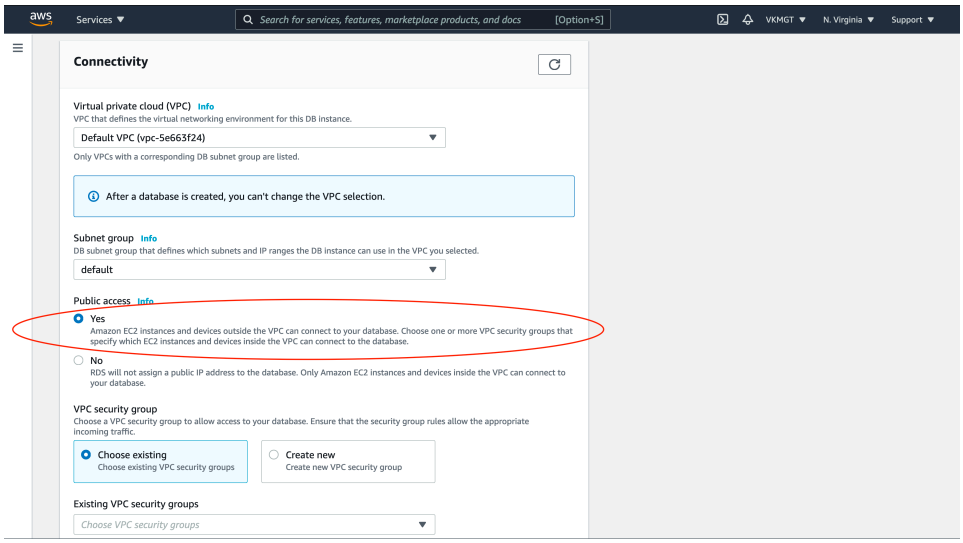


- Follow the default configurations as shown below

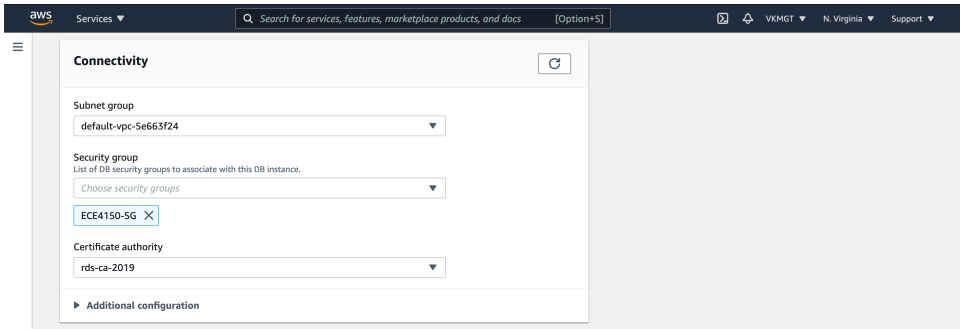




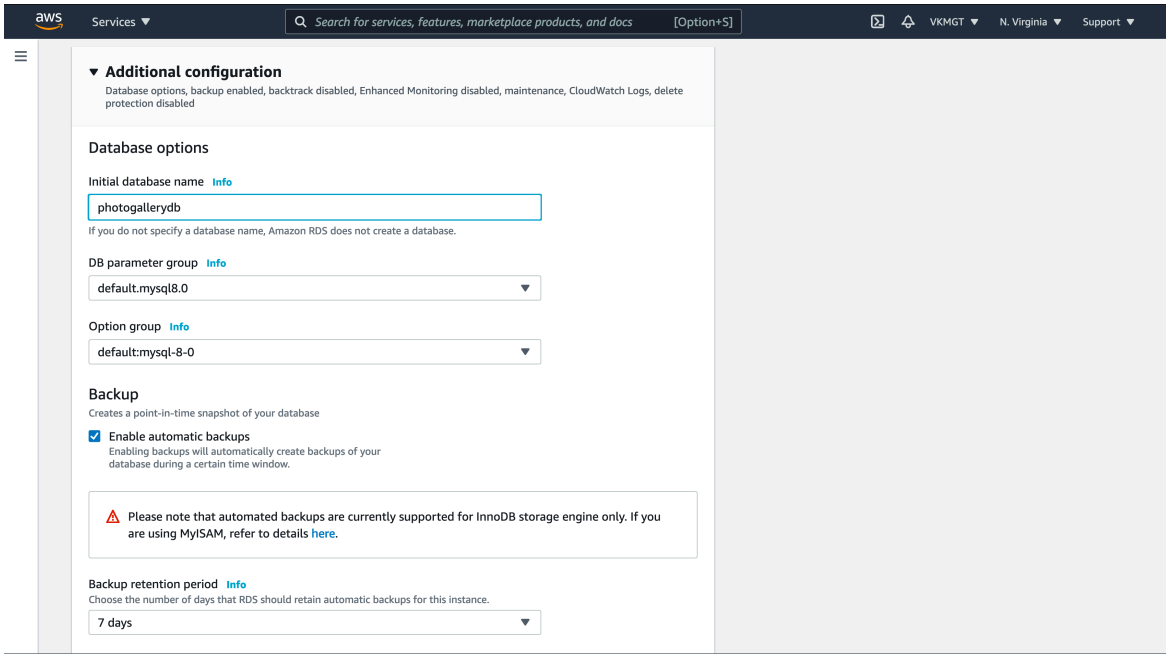
- Make sure you select the make your database publicly available over the internet, as shown below

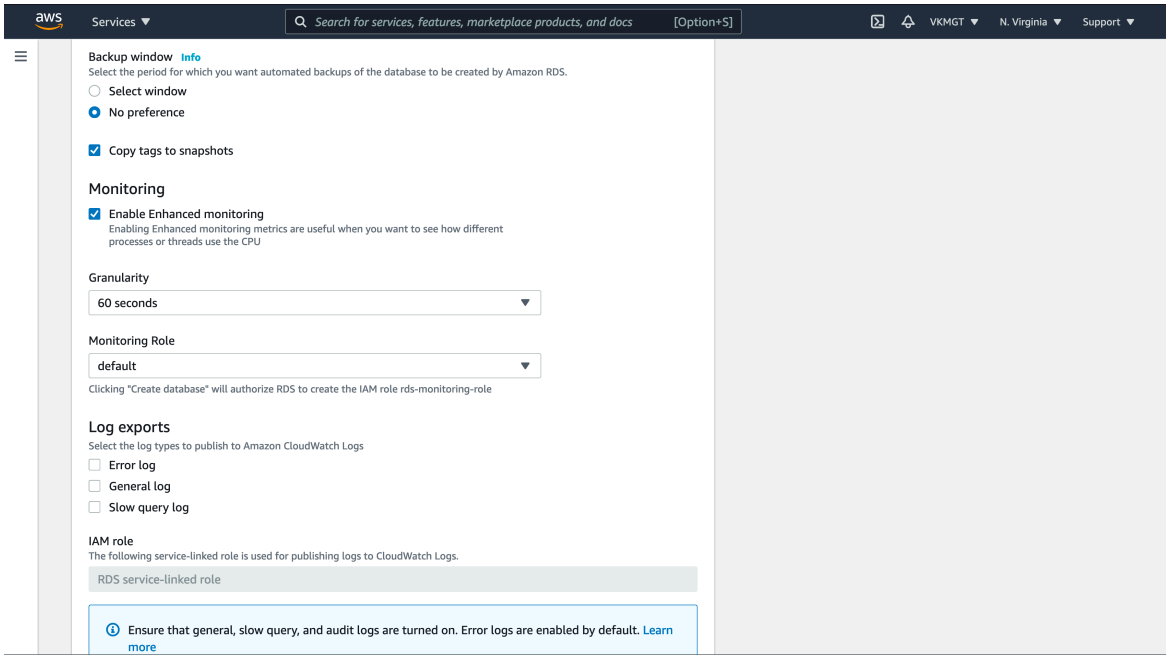


- Attach the security group that you created previously

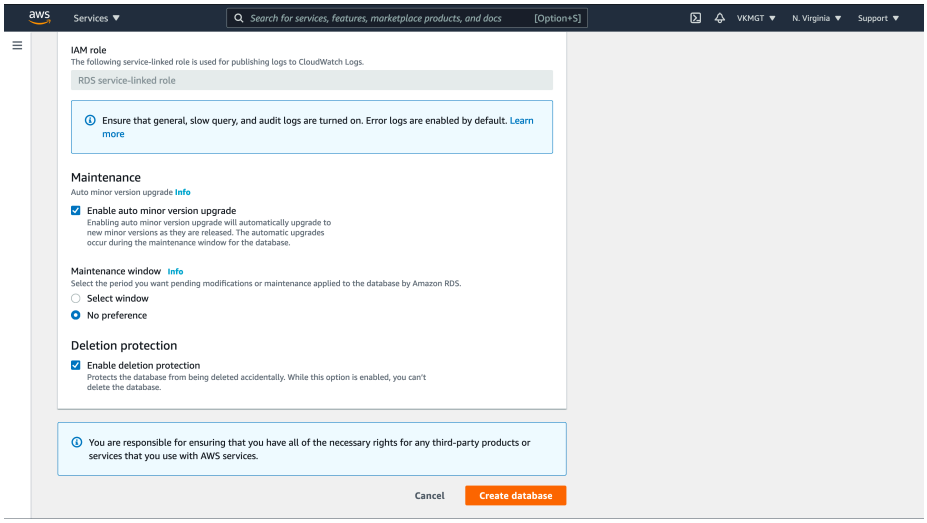


- On **Additional configuration**, under **Initial database name**, type the same name you use for the **DB instance identifier**, "**photogallerydb"**, and under Monitoring, "**Enable Enhanced Monitoring**" as shown below

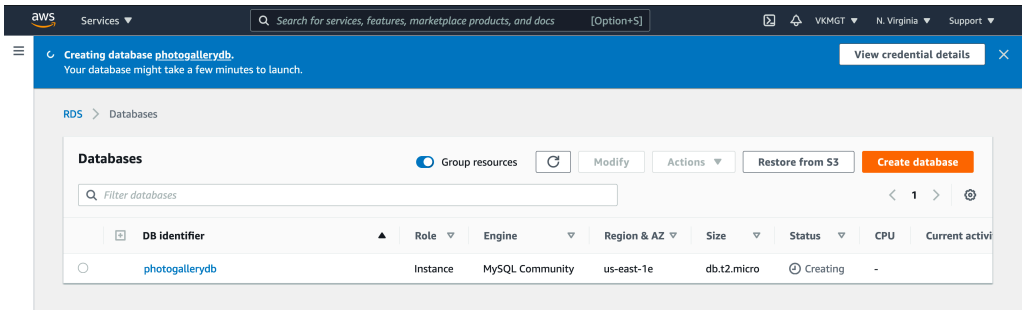


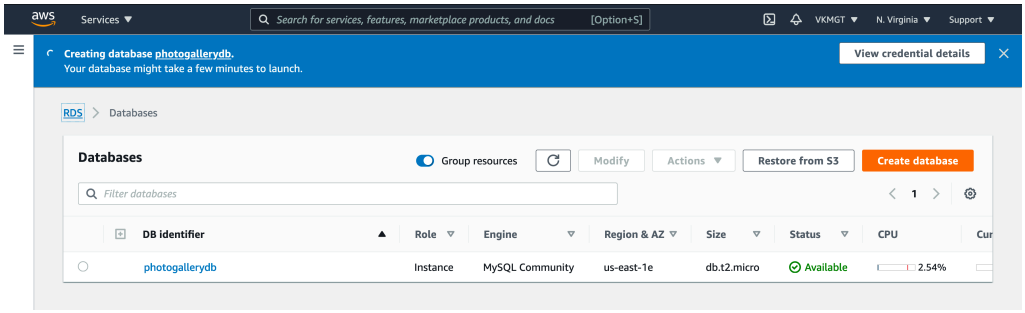


- Once you completed the configuration, create the database by clicking **Create database**

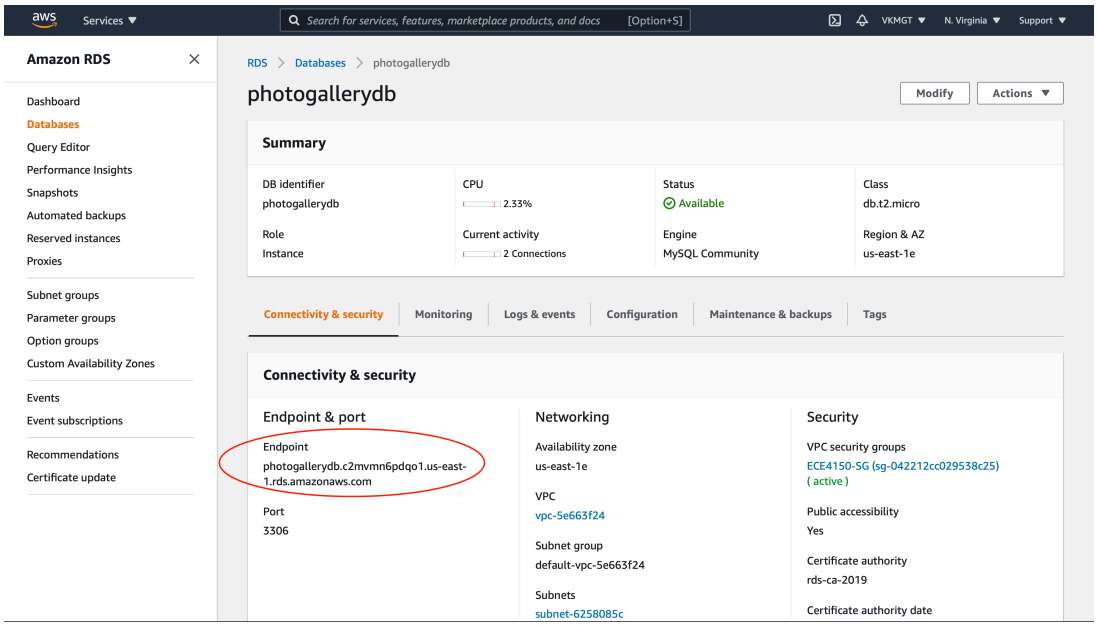


- It might take around 10-20 minutes to create, depending on the allocation. It will change from Creating to Available under Status





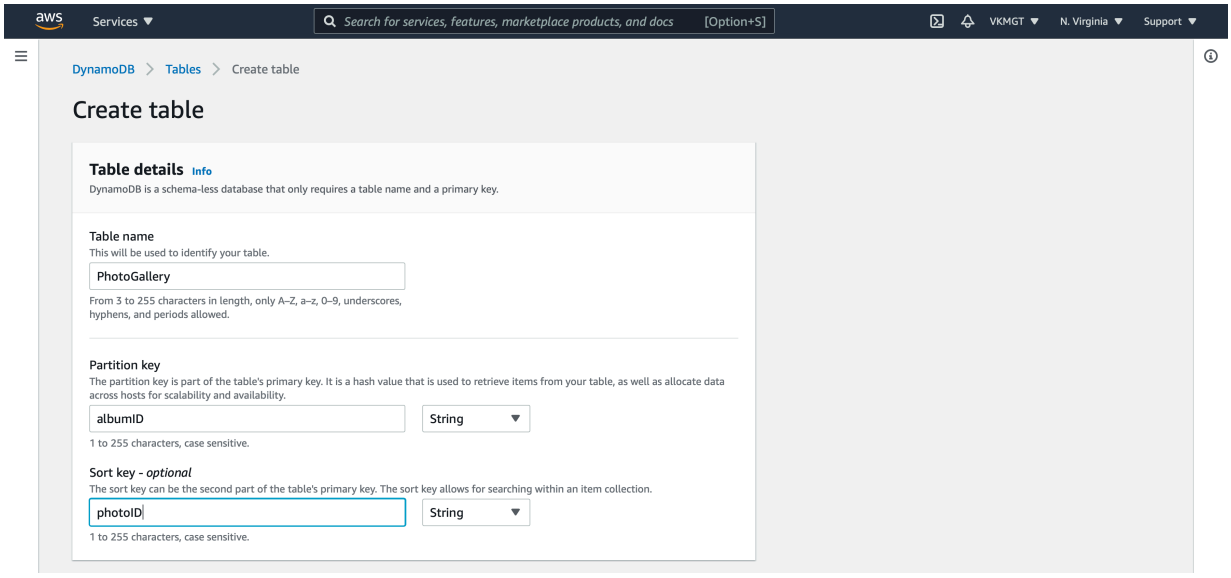
- Click on the newly created database, copy and save the endpoint to the database instance located under **Endpoint & port**, as shown below



- Go back to the Security Group section under the EC2 Console and add another inbound rule for a **Custom TCP** type from port **3306** (You also can add a **MySQL/Aurora** rule type) and configure the source coming from anywhere (**0.0.0.0/0**)

4. Create a DynamoDB table for storing records of photos (2 points)

- Create a DynamoDB table named **PhotoGallery** with a partition key and a sort key named **albumID** and **photoID**, respectively, as shown below



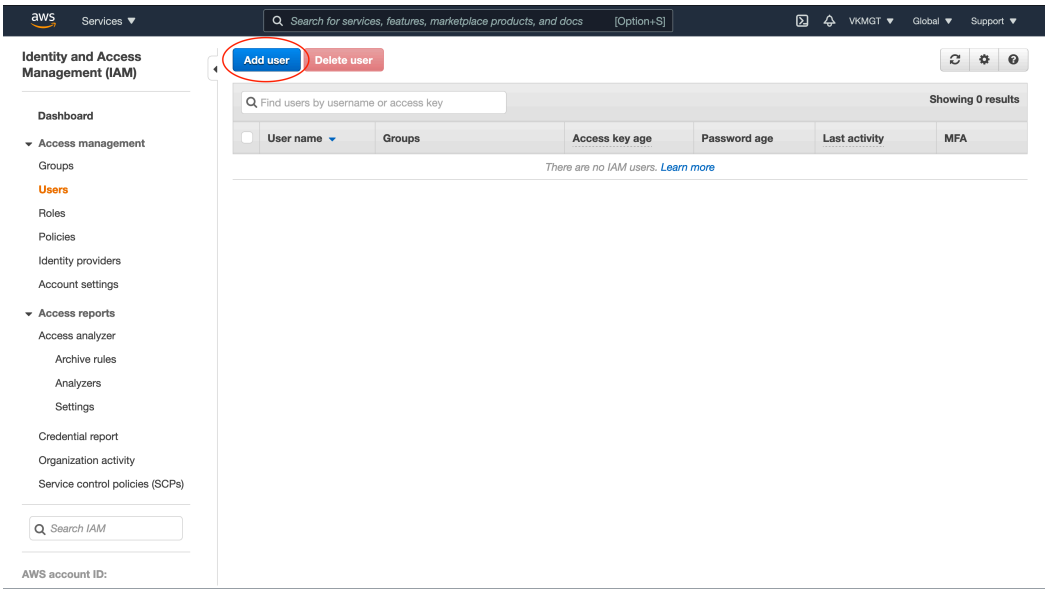
5. Create an S3 Bucket for storing photos (2 points)

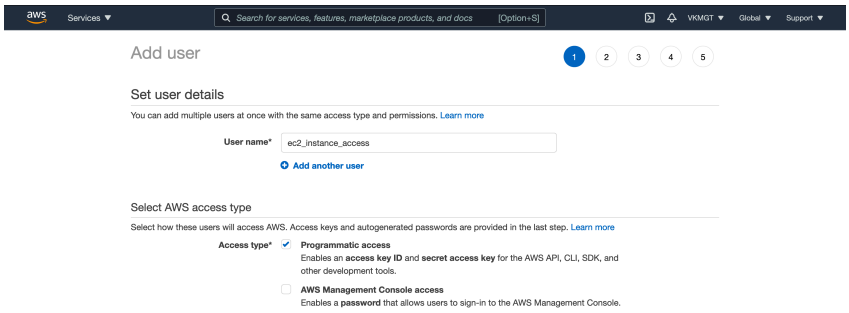
- Create a new S3 bucket for storing photos. **Uncheck Block all public access under Bucket setting for Block Public Access section**
- For the name of the bucket, use **photobucket-lastname-year-courseNumber** (e.g., **photobucket-corporan-2021-4150**)
- Create two folders in this bucket. One called '**photos**' and another one called '**thumbnails**'
- Add a bucket policy below to enable public access to the photos uploaded. Replace '**mybucketname**' with the name of the S3 bucket created

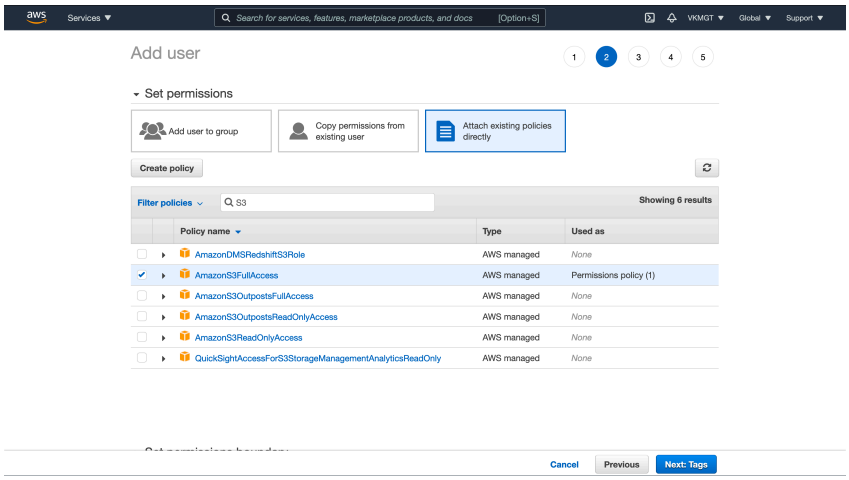
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::mybucketname/photos/*"
        }
    ]
}
```

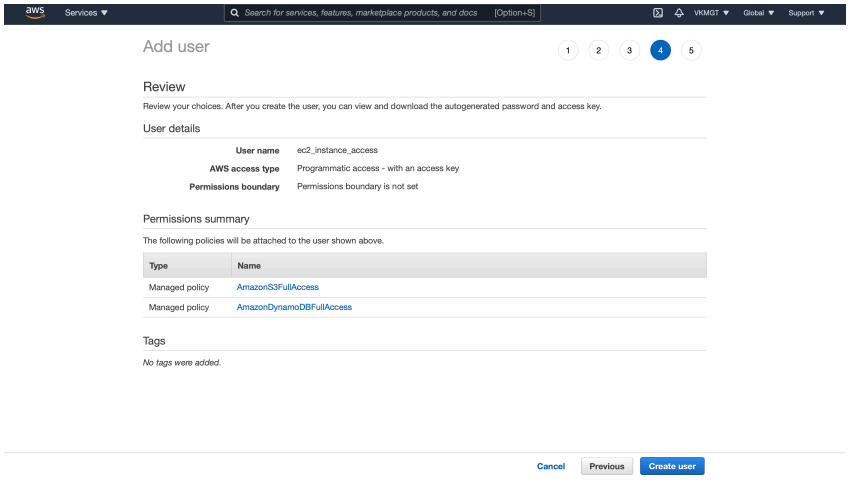
6. Create IAM User (2 points)

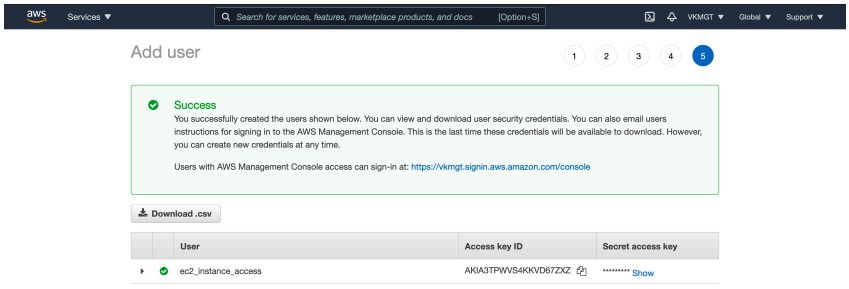
- In the AWS IAM console, create a new IAM user called **ec2_instance_access** for the EC2 instance to S3 service, as shown below. Make sure that the Access Type is only **programmatic**. Attach policy called **AmazonS3FullAccess** and **AmazonDynamoDBFullAccess** to this user. Save the **Access key ID** and **Secret access key**




Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Add user
1 2 3 4 5
Set user details
You can add multiple users at once with the same access type and permissions. Learn more
User name*
ec2_instance_access
Add another user
Select AWS access type
Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more
Access type*
Programmatic access
Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.
AWS Management Console access
Enables a password that allows users to sign-in to the AWS Management Console.

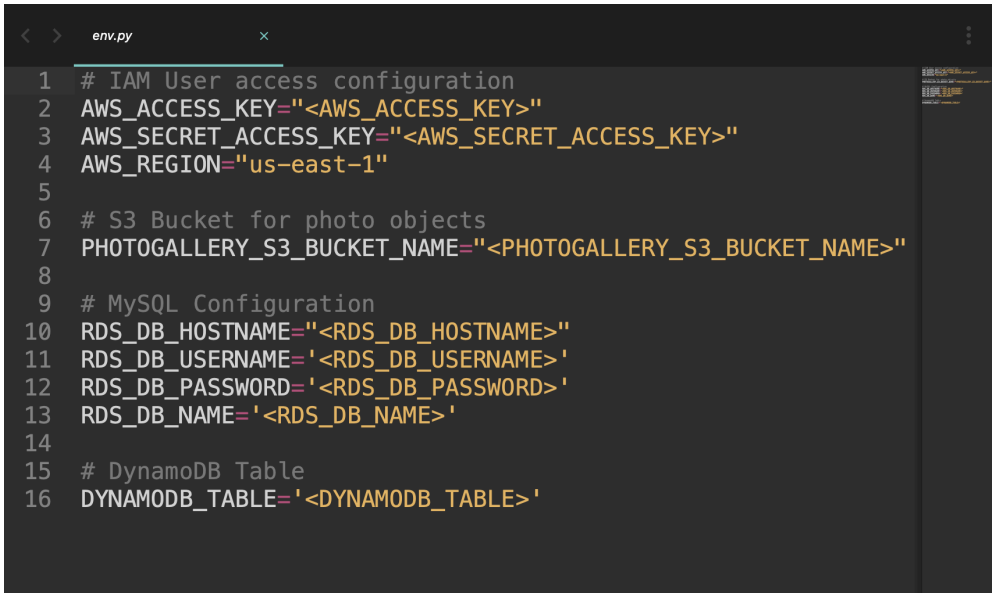

Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Add user
1 2 3 4 5
Set permissions
Add user to group
Copy permissions from existing user
Attach existing policies directly
Create policy
Filter policies
S3
Showing 6 results
Policy name
Type
Used as
AmazonDMSRedshiftS3Role
AWS managed
None
AmazonS3FullAccess
AWS managed
Permissions policy (1)
AmazonS3OutpostsFullAccess
AWS managed
None
AmazonS3OutpostsReadOnlyAccess
AWS managed
None
AmazonS3ReadOnlyAccess
AWS managed
None
QuickSightAccessForS3StorageManagementAnalyticsReadOnly
AWS managed
None
Cancel
Previous
Next: Tags


Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Add user
1 2 3 4 5
Review
Review your choices. After you create the user, you can view and download the autogenerated password and access key.
User details
User name
ec2_instance_access
AWS access type
Programmatic access - with an access key
Permissions boundary
Permissions boundary is not set
Permissions summary
The following policies will be attached to the user shown above.
Type
Name
Managed policy
AmazonS3FullAccess
Managed policy
AmazonDynamoDBFullAccess
Tags
No tags were added.
Cancel
Previous
Create user


Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Add user
1 2 3 4 5
Success
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.
Users with AWS Management Console access can sign-in at: https://vkmgt.signin.aws.amazon.com/console
Download .csv
User
Access key ID
Secret access key
ec2_instance_access
AKIA3TPWVS4KKVD67ZXZ
********** Show

7. Update env.py script and move the entire code directory to the EC2 instance (2 points)

- In the files provided, locate the **env.py** script under the **utils** directory and update all the variables required to run the Photo Gallery Application using the two variants.
- DO NOT PLACE THE NAME INSIDE THE ANGLE BRACKETS. For example, if the name of the S3 bucket to store photos is **photobucket-corporan-2021-4150** replace **<PHOTOGALLERY_S3_BUCKET_NAME>** for the name of the bucket.



8. Transferring Files between your computer to and Amazon EC2 instance (2 points)

- For linux/Unix/Mac system, we can use a command-line tool "scp" (secure copy) to transfer files between your laptop and Amazon instance. With scp you can copy files between computers on a network.
  - To **upload** a file from your computer to the EC2 instance:

```
$scp –i amazonkey.pem lab2/code.py ubuntu@ec2–host.amazonaws.com:~/data/
```

  - To **upload** a directory from your computer to the EC2 instance:

```
$scp –i amazonkey.pem –r lab2 ubuntu@ec2–host.amazonaws.com:~/data/
```

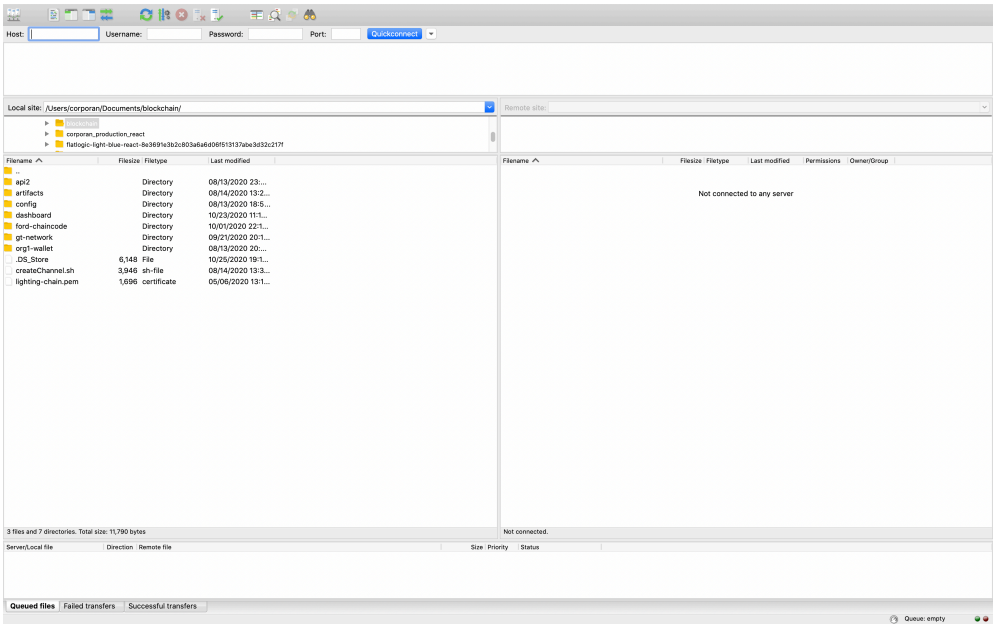
  - To **download** a file from the EC2 instance to your computer to:

```
$scp –i amazonkey.pem ubuntu@ec2–host.amazonaws.com:/data/code.py ~/Download
```

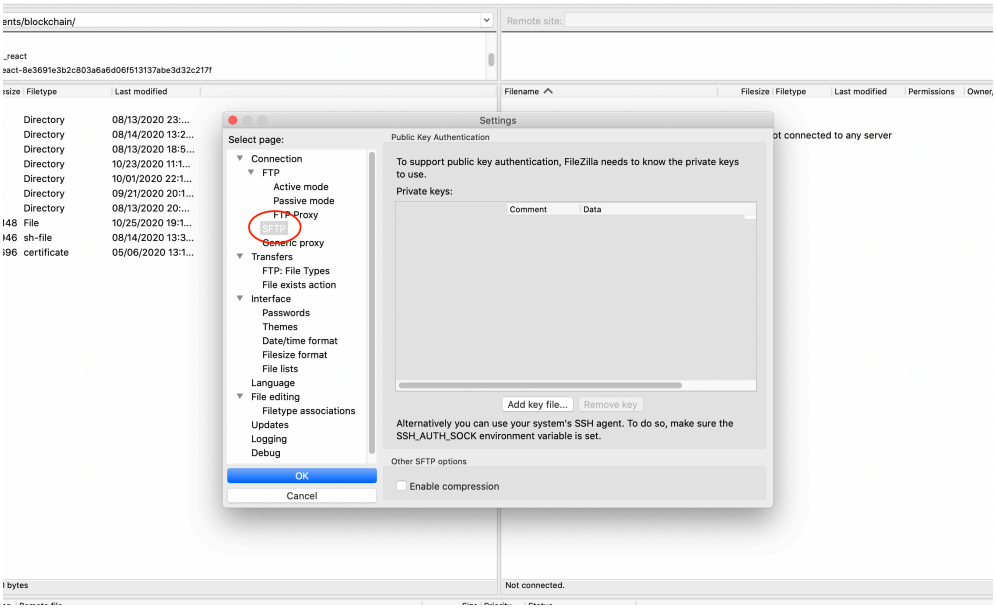
  - To **download** a directory from your computer to the EC2 instance:

```
$scp –i amazonkey.pem –r ubuntu@ec2–host.amazonaws.com:/data/lab2 ~/Download
```

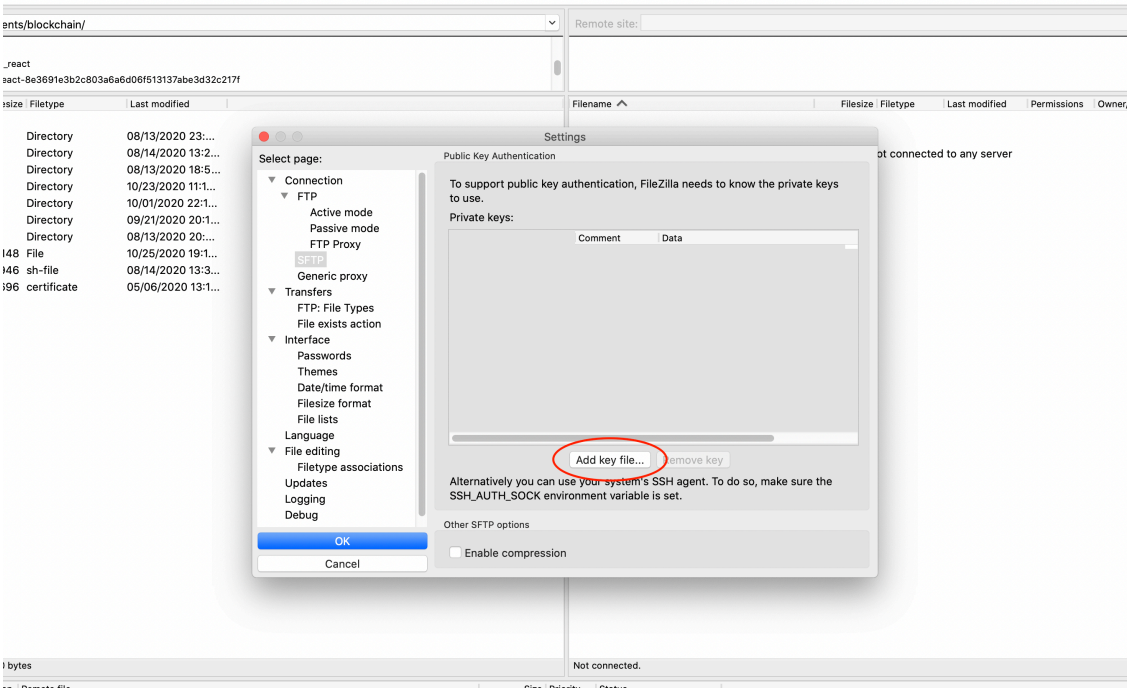
- If you want a more user-friendly tool to transfer data, FileZilla is the right choice. It is free, supports Windows/Linux/Mac systems, and has a friendly user interface. It supports FTP, SFTP, and other file transfer protocols. Go to https://filezilla-project.org/ to download it.



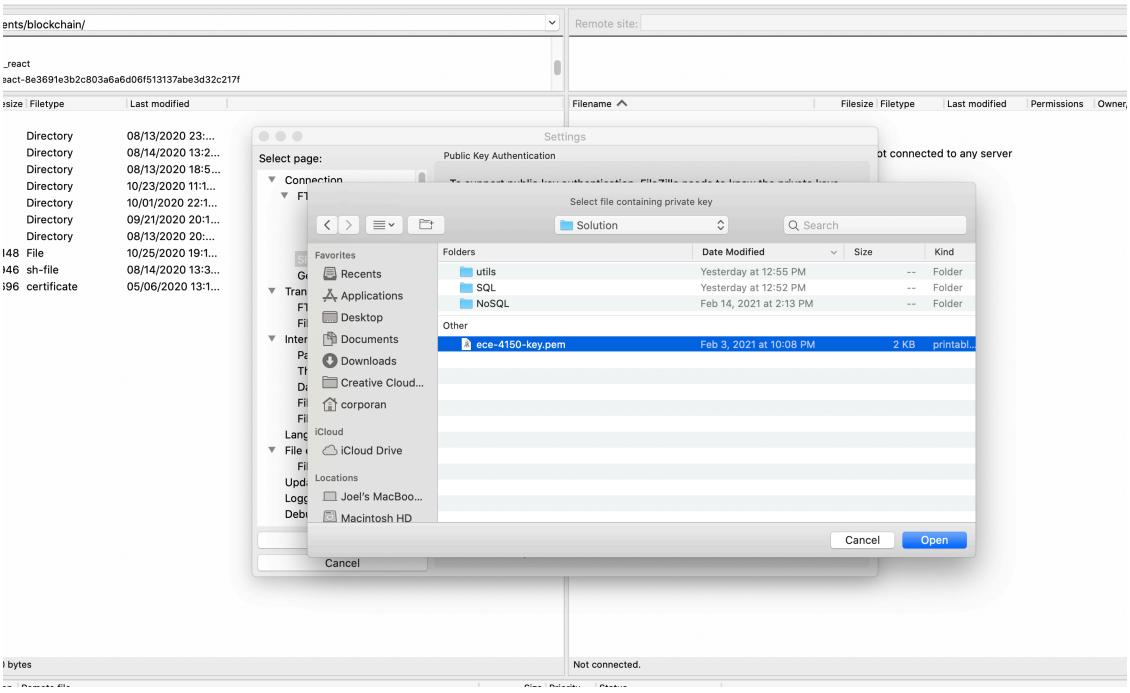
- If you want to use FileZila to upload to or download data from a normal FTP server if you have the user and password, just put the information in the “Host”, “Username”, “Password” box and connect. However for Amazon instance, we use key-pair to log in instead of password for better safety. So it is a little bit more complicated to configure.
- Under “Settings” and click “SFTP”:



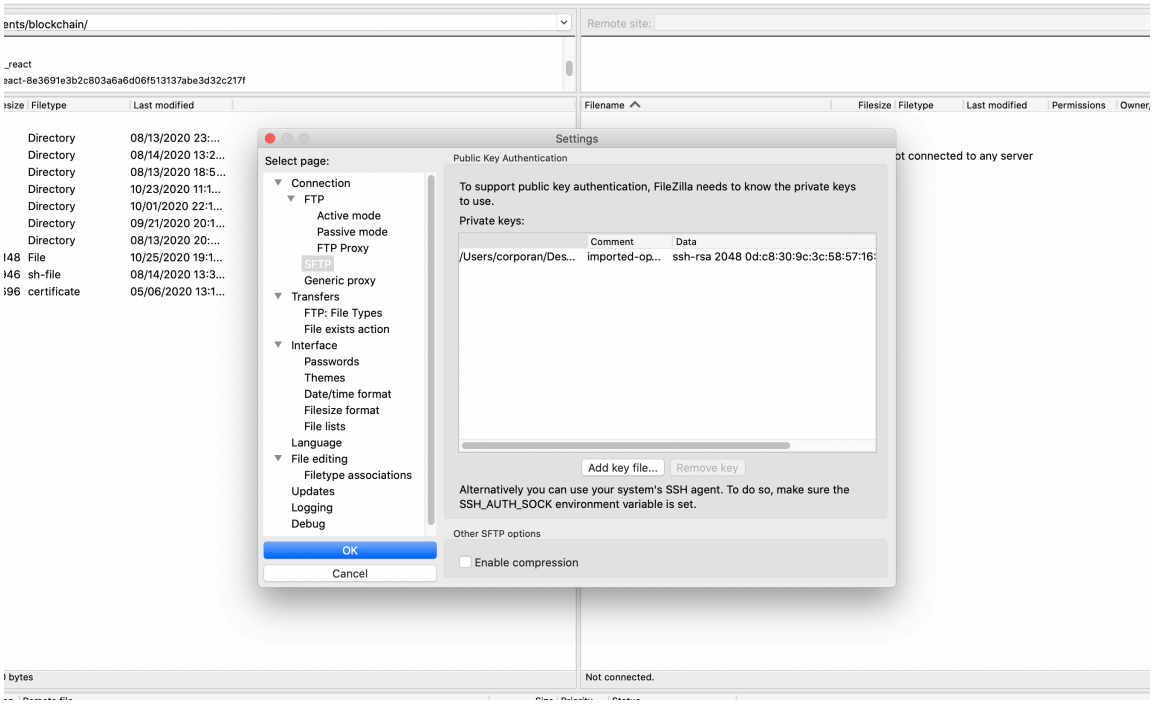
- Click “Add key file…”:



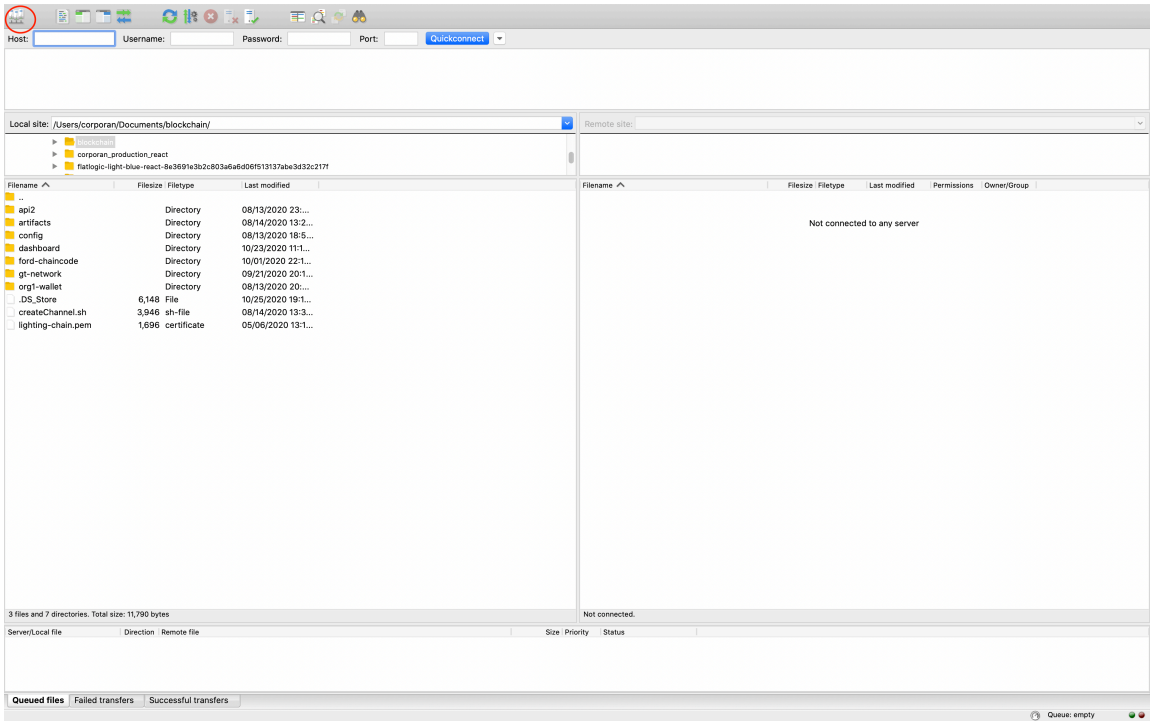
- Then select the ”.pem” file you used to connect to Amazon instance using SSH.



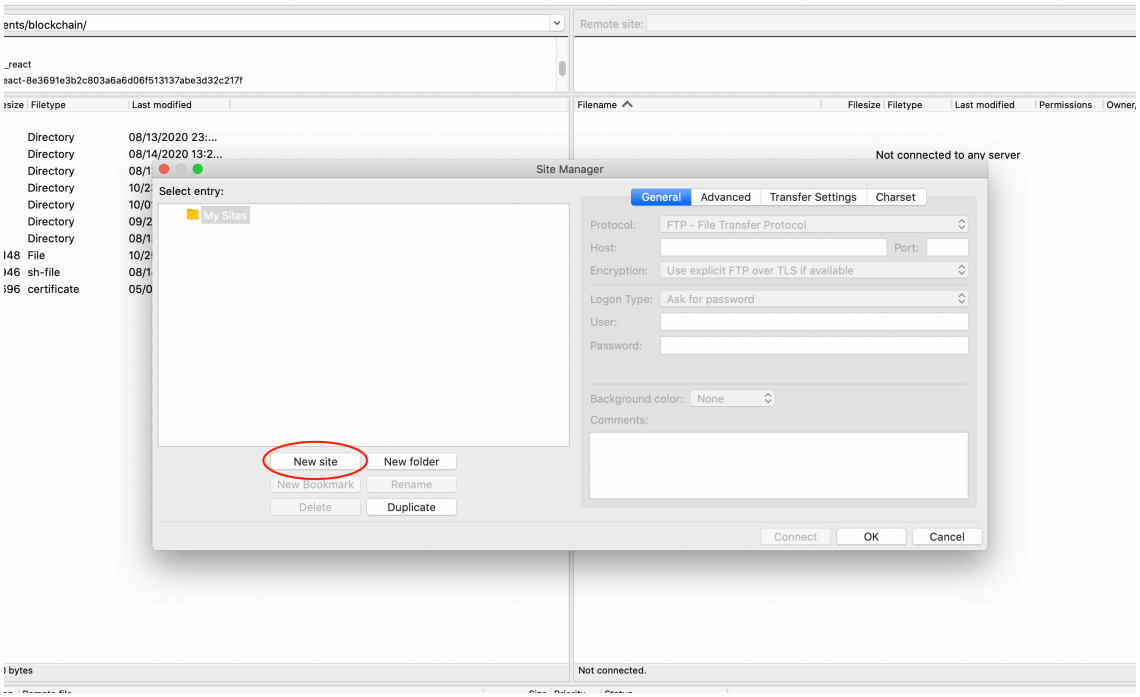
- You will see the a private key has been added.



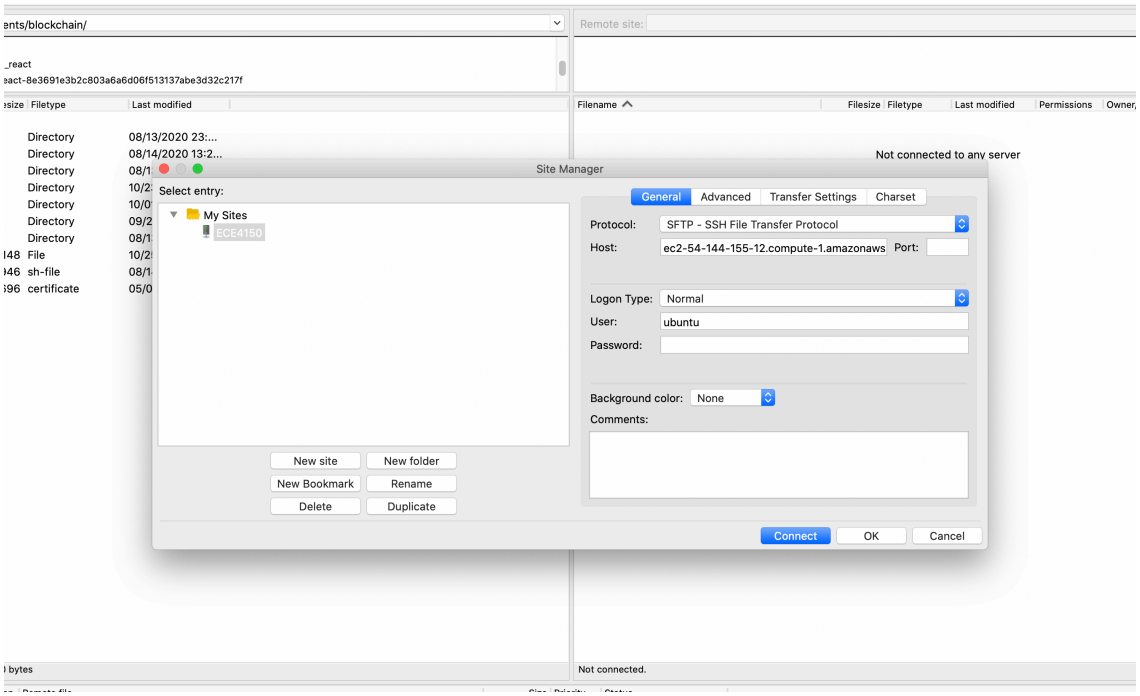
- Close “Settings” and go back to the main interface and click the button to open the “Site manager”, as shown below



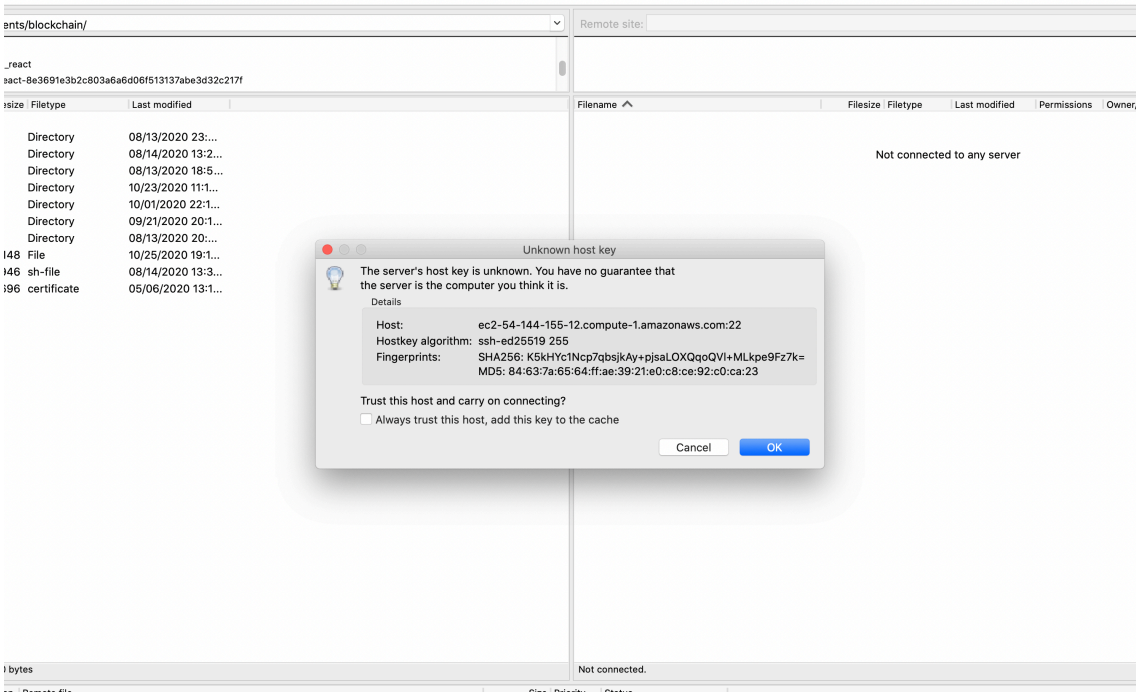
- Click “New Site” and give a name.



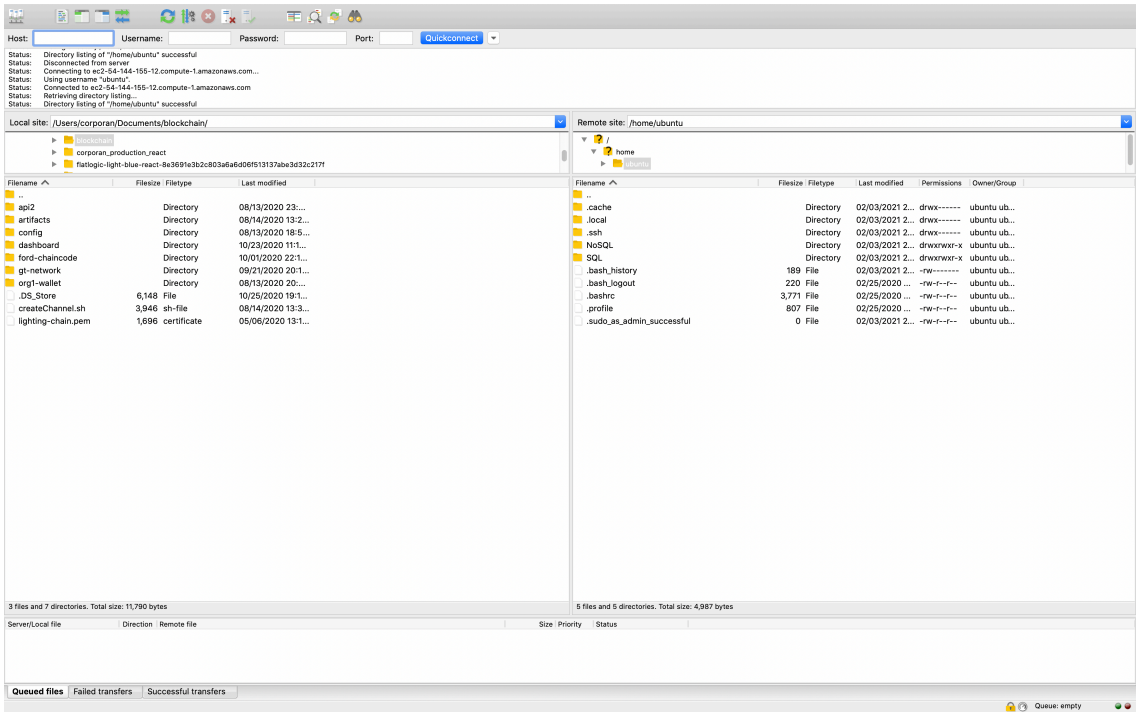
- Put the host URL of the EC2 instance in the “Host” box. Set “Protocol” as “SFTP”, “Logon Type” as “Normal”, “User” as “ubuntu” and leave “Password” as blank. Then click “Connect”.



- There will be a dialogue box to ask you about “Unknown host key”, just click “Ok”.



- All right. Now you have logged in the EC22 instance. You can drag and drop to transfer the files between you computer and the remote machine.



9. **Getting familiar with the code and installing the necessary libraries inside the EC2 instance (2 points)**

- Update your EC2 instance by running this command once you are inside the instance:

```
sudo apt update
```

- Take a look at the NoSQL and SQL directory and get familiar with the code provided to run the Flask web framework (app.py). To run the framework, we need a couple of libraries using the terminal. First, make sure you have the package installer pip3 and Python3 installed on the instance. You can use this command to instance these dependencies:

```
sudo apt install python3-pip
```

- You can also use Homebrew to install these packages (https://brew.sh) and any other method. After installing the Homebrew, use the following command to install the lastest version of Python3 and pip3:

```
brew install python3
```

The python version used for this lab was 3.9.0 and pip3 20.3

- Now, we will install a couple of libraries that we will use for both SQL and NoSQL variants. After installing python3 and pip3, install the following libraries using the package installer pip

```
pip3 install exifread flask PyMySQL
```

- If you get an error about missing libraries, use pip3 to install those libraries.
- Go back to the Security Group section under the EC2 Console and add another inbound rule for a **Custom TCP** type from port **5000** and configure the source coming from anywhere (**0.0.0.0/0**). This configuration will allow you to access your application inside the EC2 through the 5000 port.

You can also test each of the variants outside your EC2. If you have Python3 and pip3 installed on your computer, install all the dependencies mentioned above and run the following command inside the directory the variant that you would like to test:

```
python3 app.py
```

10. Photo Gallery using NoSQL (2 points)

- For the NoSQL variant, we will store the album and photo records within the same table using the partition key and sort ket for the albumID and photoID, respectively.

DynamoDB Partition Keys and Sort Keys

- Inside the NoSQL directory, run the following command to run the Flask web framework using the NoSQL variant:

```
python3 app.py
```

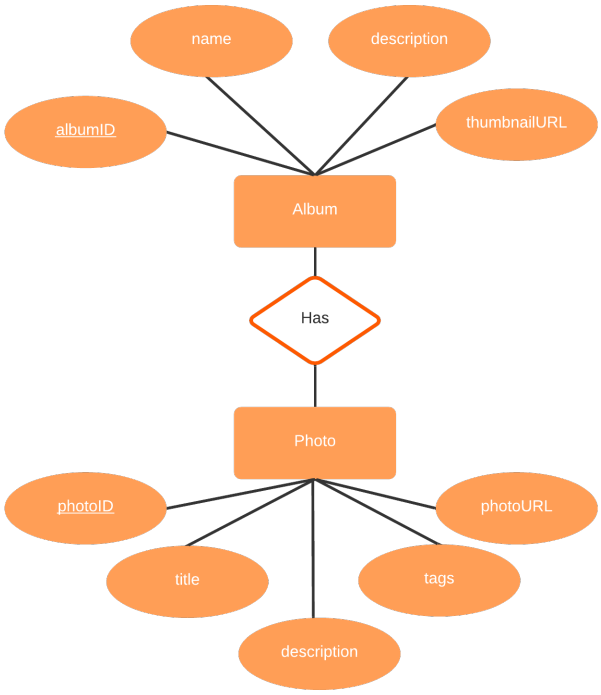
- Copy the hostname (e.g., URL or endpoint) from your EC2 instead of the link provided in the terminal when you run the Flask web framework. Add the port name (5000) and add it to the browser. It is advised to use the following browsers: Chrome and Firefox.

```
yourEC2InstanceHostName.com:5000
```

Remember to turn off your application if you want to use the other variant

11. Photo Gallery using SQL (2 points)

- For the SQL variant, we will store the album and photo records in different tables.



Entity Relationship Diagram that show the relation between albums and photos

- Inside the **utils** directory, you will find a **sqlcommands.sql** that includes the **Album** and **Photo** tables (the **User** table will be used in the last exercise).
- Within the same directory, you will find a python script to create the SQL tables for **Album** and **Photo**. Get familiar with the script's code structure because the method to call the RDS instance is in some way similar to the one used in the web framework. Run the following command to create the **Album** and **Photo** table:

```
python3 album-photo-tables.py
```

- Run the following command within the SQL root directory to run the Flask web framework using the SQL variant:

```
python3 app.py
```

- Copy the hostname (URL) from your EC2 instead of the link provided in the terminal when you run the Flask web framework. Add the port name (5000) and add it to the browser. It is advised to use the following browsers: Chrome and Firefox.

```
yourEC2InstanceHostName.com:5000
```

Remember to turn off your application if you want to use the other variant

12. Exercises: User Management and authentication, besides other CRUD operations, to support the Photo Gallery (60 points)

- Modify both NoSQL and SQL in the following exercises:
    1. Design and implement a new method for user management and authentication without using external services, such as AWS Cognito. **(30 points)**
        1. Allow users to create an account (sign up) and sign in (log in) through the website.
        2. Store the information of the user in a new table called **User** (For SQL variant, we provided a sample schema that you can use. However, you might need include other attributes within that schema. For NoSQL variant, create a new table called **PhotoGalleryUser**). A few requirements for the use case:
            i. Use password hashing when signing up (**Appendix C**).
            ii. The **userID** is a universally unique identifier (UUID).
            iii. UserID and email **must be** unique. A user cannot sign up multiple times with the same email.
            iv. To login the user uses the **email** and **password** to login.
        3. To validate a new account, send an email to the user (See **Appendix A**) with a confirmation token with a URL to the server (See **Appendix B**) (the confirmation token should include the **userID**). When the user clicks the email, the URL should route to a resource in your Flask web framework. The resource will take the incoming request, decode the token string to get the **userID**, and confirm the user in the database. The resource's response should redirect to the login page. A few requirements for the use case:
            i. The user cannot use the website until the user is verified.
            ii. The token should last 10 minutes. If the user does not click the link within that time, the token will be invalid.
        4. When the user logs in, the web server should store a session token in the user's browser. This token will be used to validate the navigation of that particular user in the website. The token should last only 5 minutes. After the token is expired, the user should be redirected to the login page.
    2. Add a new method to allow the user to cancel its account. **(15 points)**
        1. Remove all the albums created by a user when this one cancels its account. A few requirements for the use case:

i. For NoSQL, attached a new attribute when the album is created to identify the creator.

ii. For SQL, update the album schema to add foreign key from the user schema. This strategy will allow you to remove the all albums created by the user.

3. Add a new method to delete a photo in the PhotoGallery. **(5 points)**
4. Add a new method to delete a whole album in the PhotoGallery. This should delete all the photos inside that particular album. **(5 points)**
5. Add a method to update the following information of a photo: title, description, and tags. For NoSQL, whenever you make an update, update the updatedAt attribute as well. In SQL, it is done automatically. **(5 points)**

*****For this exercise, there are designated sections (with comments) where you need to add your functions, routes, and environment variables needed. Make sure you use these sections to insert your code. If you need to modify any other function, route, or environment variable outside this section, put a comment that identifies your modification.*****

Deliverables:

1. A video showing all the full functionalities of your Photo Gallery application, including the ones in the final exercise. Please show your AWS username clearly to indicate that you are using your account. You may briefly explain the functions that you created for the last activity, **but please make sure to limit the total length to 12 minutes.**
2. The complete code with the modifications needed to complete each exercise, including the modified SQL schema (include the modified schema in the **sqlcommands.sql** file).

Appendix

A. How to generate confirmation token

The email confirmation should contain a unique URL that a user needs to click to confirm his/her account. Typically, a confirmation URL has this form:

`http://yourEC2InstanceHostName.com/confirm/{id}`

Where the id is an encoded string containing the email of the particular user, a timestamp, and salt (a is random string used in hashed data to safeguard passwords in storage).

Python offered a lot of libraries to create this type of tokens. However, a recommended library for this lab is **itsdangerous** (https://itsdangerous.palletsprojects.com/en/1.1.x/). This library has a function called **URLSafeTimedSerializer** that allows to create a serialize (hash) token for URLs with a record of the time of the signing. It is a good strategy if you want to delegate the validation and expiration of signatures. Here is a simple example:

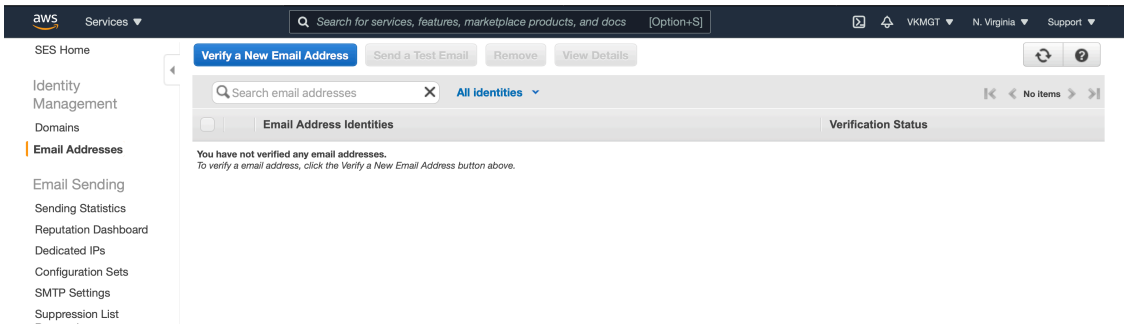
```
from itsdangerous import URLSafeTimedSerializer

email = 'myemailaddress@gatech.edu'

serializer = URLSafeTimedSerializer('some_secret_key')
token = serializer.dumps(email, salt='some-secret-salt-for-confirmation')

print(token)
# eyJpZCI6NSwibmFtZSI6Iml0c2Rhbmdlcm91cyJ9.6YP6T0BaO67XP--9UzTrmurXSmg

try:
    email = serializer.loads(
        token,
        salt='some-secret-salt-for-confirmation',
        max_age=3600
    )

    print(token)

    #myemailaddress@gatech.edu

except Exception as e:
    print('expired token')
```

B. How to send an email (AWS SES)

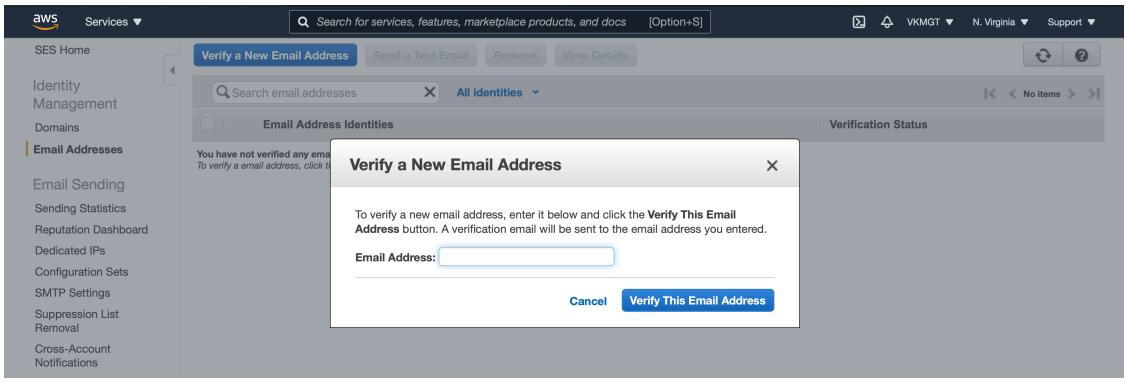
If you are trying to send a user an email to confirm their account, you need a transactional email service. AWS SES is the right solution for your application. Amazon Simple Email Service is a platform within AWS that allows you to send and receive emails using your email address or registered domain.

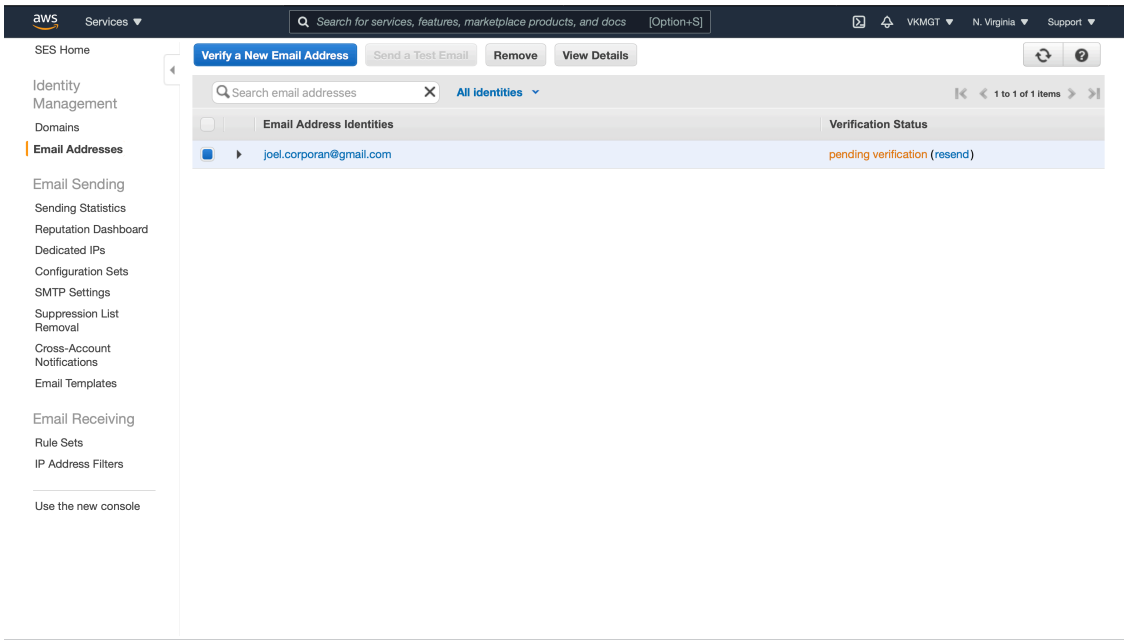
Amazon SES has proven to be a cost-effective email service. It is designed to send both bulk and transactional emails. AWS SES allows you to send emails directly from the SES console, via the Simple Mail Transfer Protocol (SMTP) interface, or through the API (You can use Boto3 to make calls the API).

Before you can send emails with SES, you must first verify that you own the email address you wish to use as a sender. Navigate to Amazon SES, click "Email Addresses" in the sidebar, and then click the "Verify a New Email Address" button.

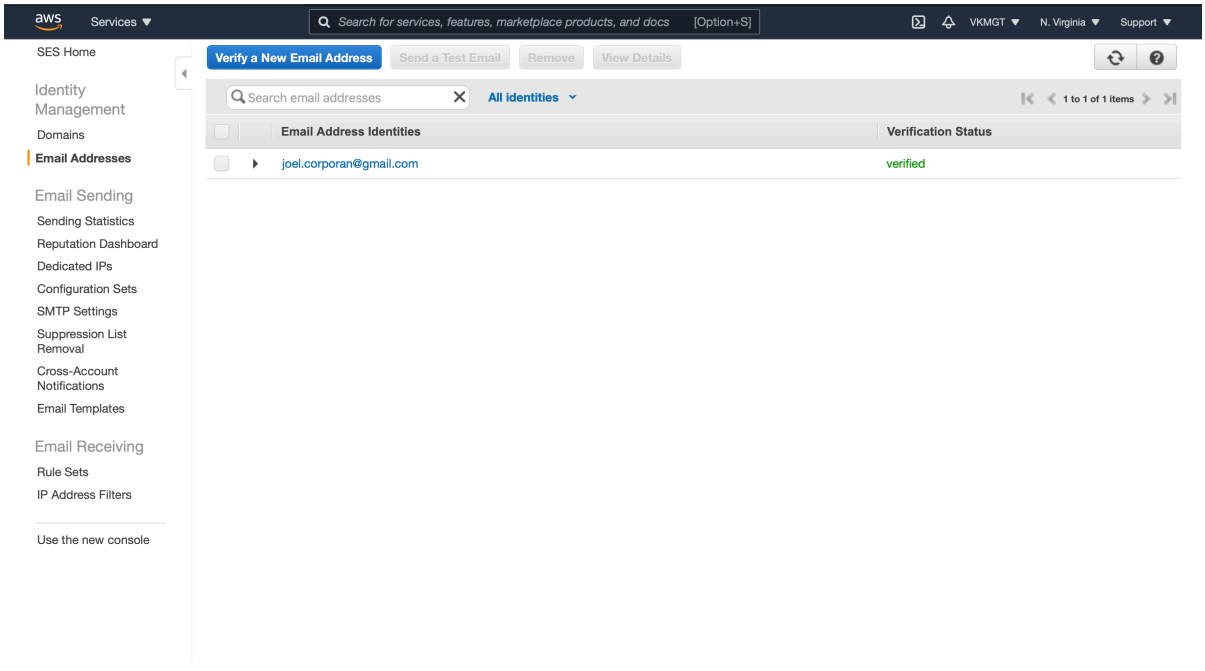


Enter the email you'd like to use and click "Verify This Email address".





Then, after clicking the verification link in your email inbox, you should see your email verified back on SES.



New accounts are automatically placed in a sandbox mode to help prevent fraud. You can only send emails to addresses you have personally verified with Amazon. If you want to remove this restriction, you must request Amazon to move out of the sandbox mode. Fortunately, this is enough since you are not deploying this application to a production environment.

Below is an example of how to send an email using boto3:

```
import boto3
from botocore.exceptions import ClientError

# Create a new SES resource and specify a region.
ses = boto3.client('ses',
                   region_name='AWS_REGION',
                   aws_access_key_id='AWS_ACCESS_KEY_ID',
                   aws_secret_access_key='AWS_SECRET_ACCESS_KEY')

SENDER = 'MyOtherVerifiedEmailAddress@gatech.edu'
RECEIVER = 'myVerifiedEmailAddress@gatech.edu'

# Try to send the email.
try:
    #Provide the contents of the email.
    response = ses.send_email(
        Destination={
            'ToAddresses': [RECEIVER],
        },
        Message={
            'Body': {
                'Text': {
                   'Data': 'This is an email from AWS SES',
                },
            },
            'Subject': {
               'Data': 'Hi, I'm sending this email from AWS SES'
            },
        },
        Source=SENDER
    )

# Display an error if something goes wrong.
except ClientError as e:
    print(e.response['Error']['Message'])

else:
    print("Email sent! Message ID:"),
    print(response['MessageId'])
```

C. Password Hashing

As we know now, it is a terrible idea to stored a password as text without any encryption. For that reason, you do not want to include passwords in your database as plain text, as this would make your users' passwords exposed if your server got hacked and your database vulnerable. That is why you need to protect sensitive data by using a hash method before storing them in your database. This is a simple step to provide a lot of security.

The article below from Dustin Boswell will help you to understand and implement hashing algorithms, such as **bcrypt** (recommended by the author). Also, you will find a flask version of **bcrypt** below.

http://cs.wellesley.edu/~cs304/lectures/bcrypt/dustwell.html

https://pypi.org/project/bcrypt/