Technical Documentation: Docker Installation on AWS

Table of Contents

1. Introduction

Docker, a ground-breaking tool that is revolutionizing the development, shipping, and operation of applications. Regardless of your level of experience with containerization, Docker provides unmatched efficiency and flexibility for software deployment. We'll go over the basics of Docker, covering everything from what containers are to how to create and manage them practically down to the installation process. Let's get started!

2. Getting Started with Docker

a. Benefits of Docker

Docker offers many advantages. Some key advantages include portability, high isolation, resource efficiency, scalability, and fast development and deployment.

- **Portability**: Docker containers encapsulate applications and their dependencies which makes it easier to seamlessly migrate applications between environments. Regardless of where you are working, the docker container will consistently run the same way.
- **Isolation:** Docker containers offer a high level of isolation between applications and their dependencies. This means that since each container runs independently, it will not interfere with other containers on the same machine or server.
- **Resource efficiency:** Containers share the underlying OS kernel, which makes them more efficient than traditional virtual machines. This can lead to a more efficient use of resources, and for cloud computing, cost savings because you can run more containers on the same existing hardware.
- **Scalability:** Docker makes applications scalable with docker images which include everything required to run and even deploy an application. Docker images can be pulled and run on containers across different systems and devices, increasing the scalability of the applications run and developed via Docker.
- **Fast Development and Deployment:** Containers can help improve the speed and reliability of deployments, whether on traditional infrastructure or via the cloud. By packaging applications into containers, developers can be confident that their applications will run consistently on any cloud platform and on any different system. This can help to reduce the time it takes to deploy new applications and to minimize the risk of problems.

b. Docker Components

To understand Docker, we must first understand some of its key components. Docker uses a client-server architecture where the Docker client communicates with the Docker daemon.

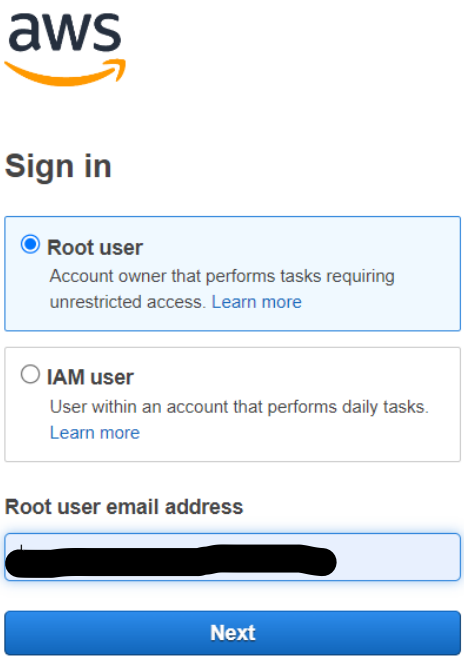
- **Docker Client:** The client is the first component of Docker. It allows the users to communicate with Docker. For example, when a user gives a command to docker, the docker client sends the desired command to the host, which further fulfills the command using Docker API. The client can interact with multiple hosts.
- **Docker Image:** Docker images are used to build containers. These images are read-only binary templates in YAML.

- **Docker Daemon:** Docker Daemon is one of the most essential components of Docker as it is directly responsible for fulfilling the actions related to containers. The Docker daemon mainly runs as a background process responsible for creating, running and distributing containers.
- **Docker Networking:** Docker networking helps in establishing communication between containers.
- **Docker Registry:** Docker images require a location where they can be stored, and the Docker registry is that location. Docker Hub is the default storage location of images that stores the public registry.
- **Docker Container:** A Docker container is the instance of an image that can be created, started, moved, or deleted through a Docker API. Containers are a lightweight and independent method of running applications. They can be connected to one or more networks and create a new image depending on the current state.

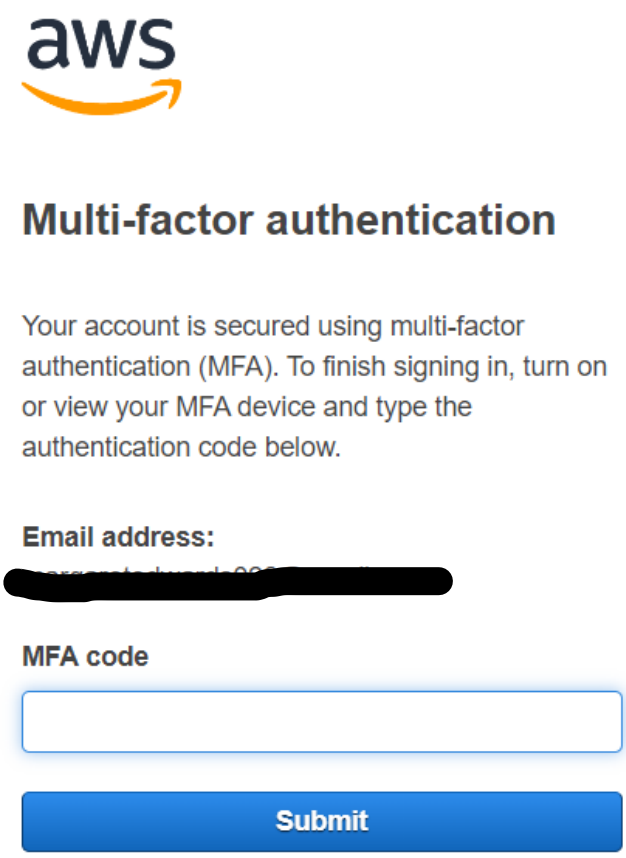
3. Installation Steps

a. Pre-installation Steps

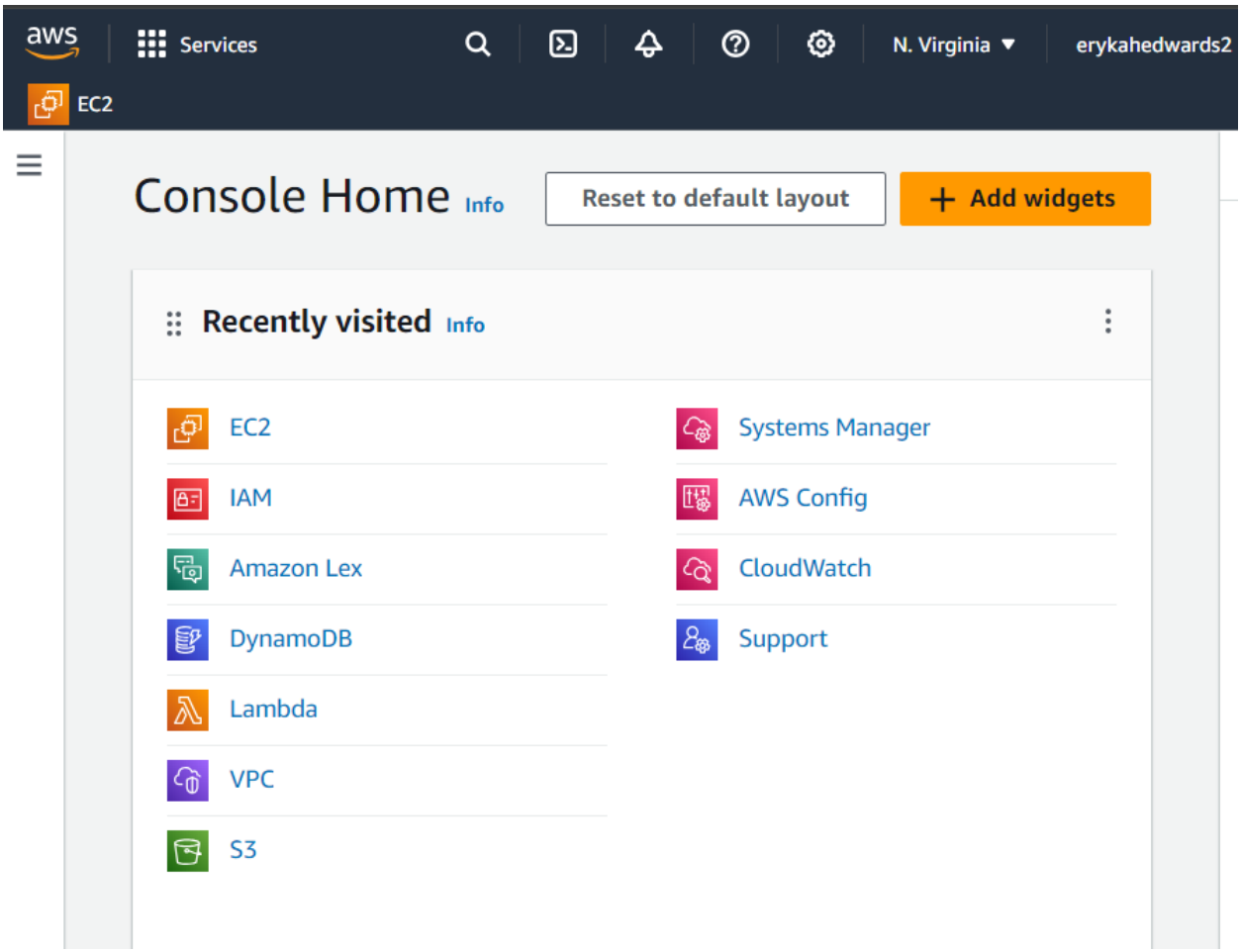
Login to your AWS Console



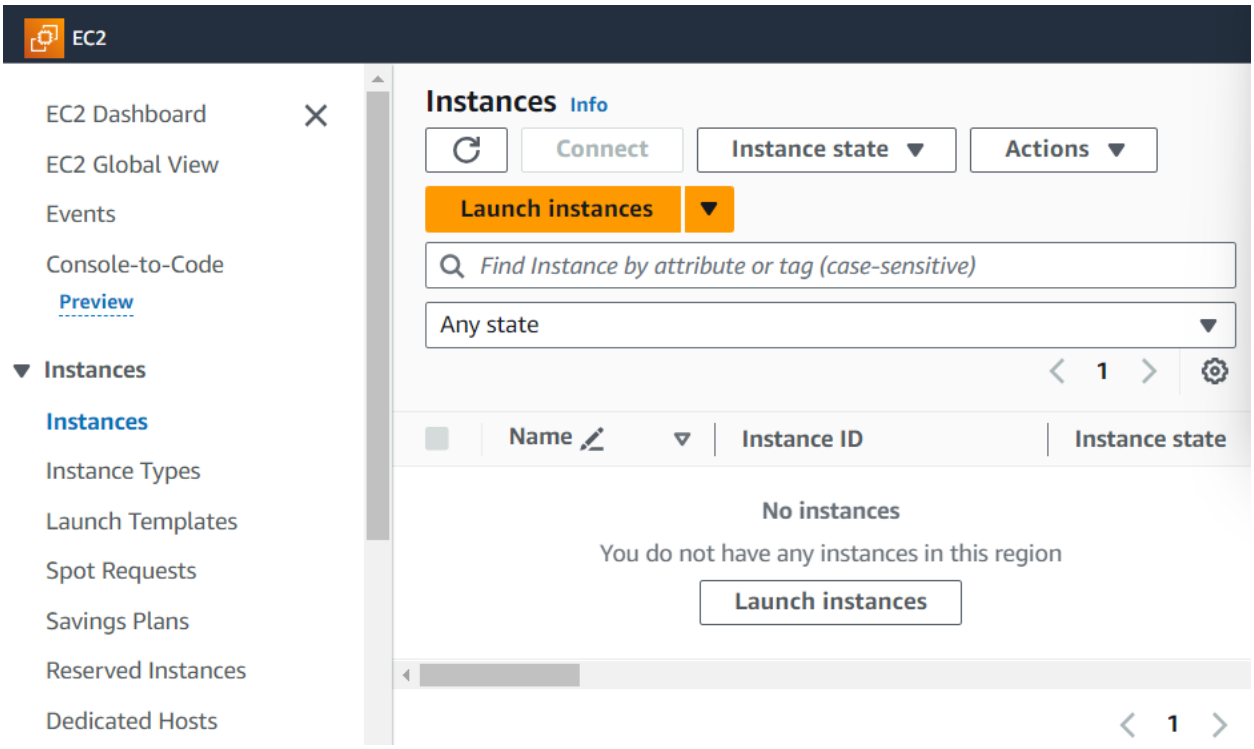
Here I am logging in as a Root user. It is not recommended to regularly login as root user as it can pose security risks. IAM user login suggested.



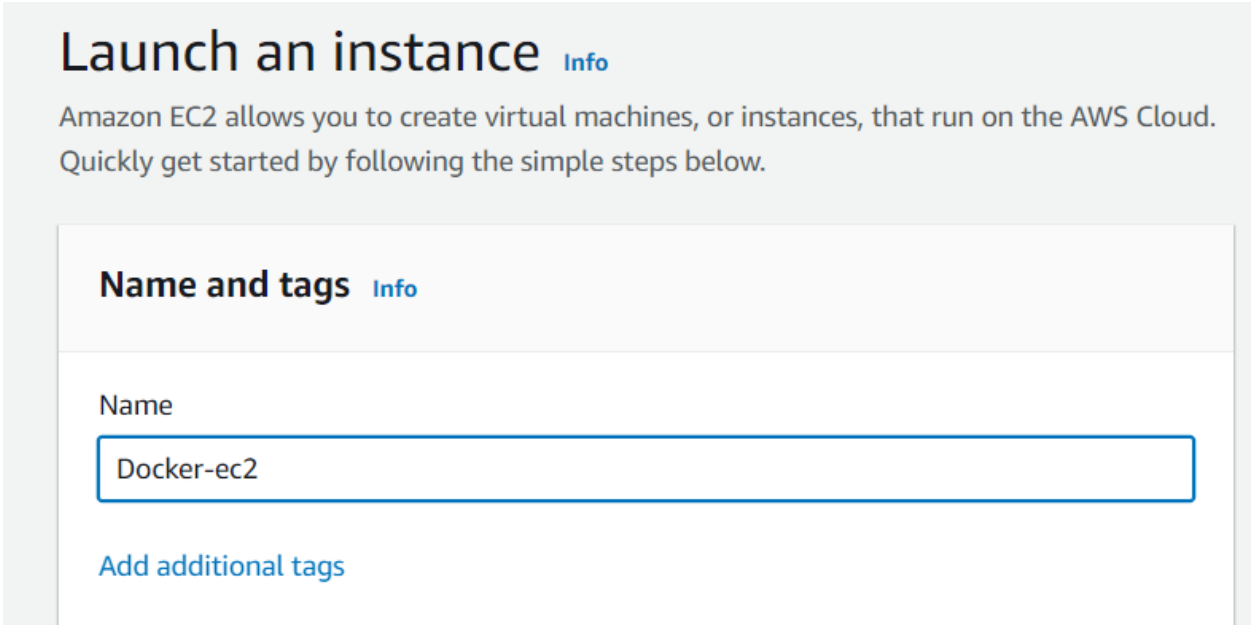
Here I am using my authenticator app code as a method of MFA.



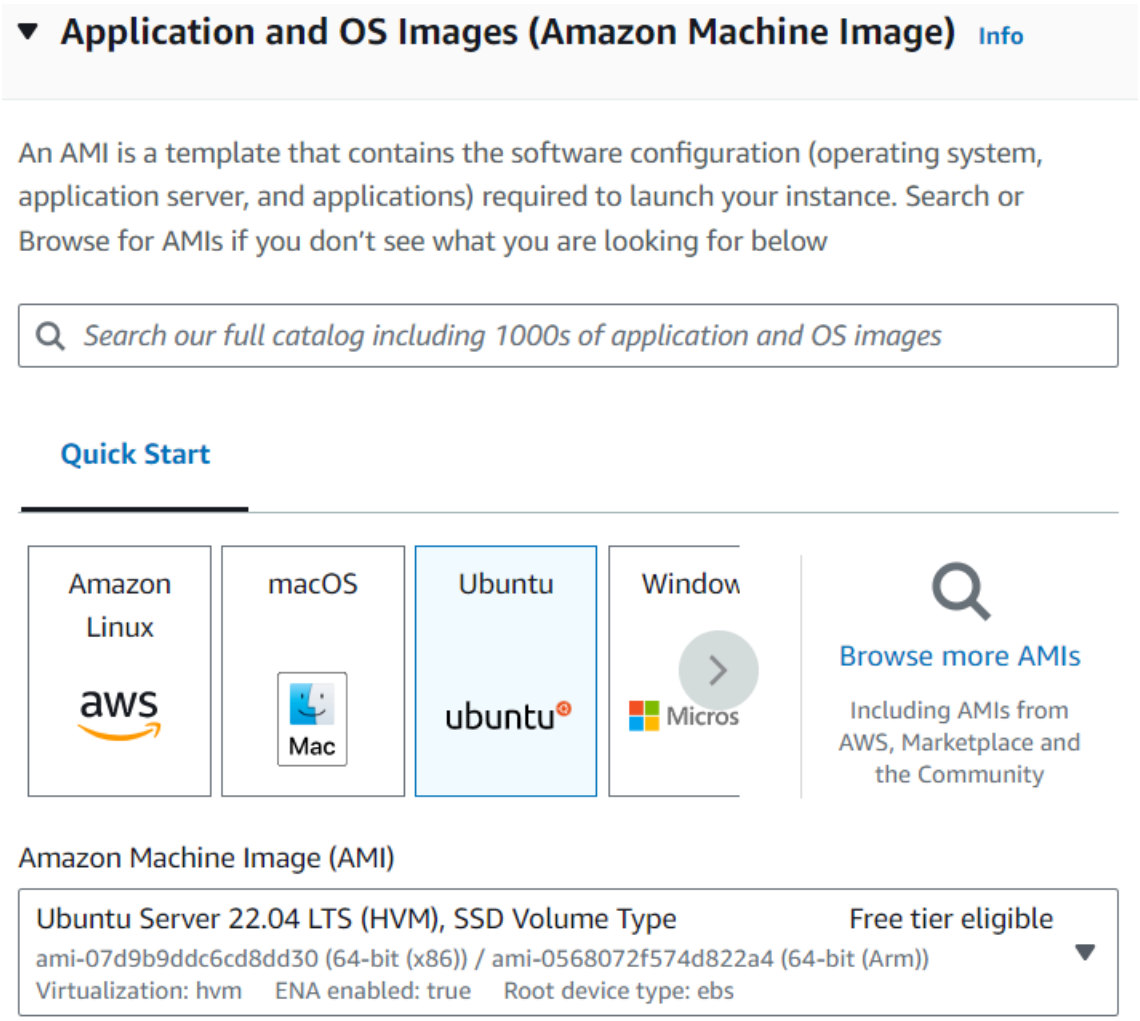
Now, we are at the Console home page. Let's create an EC2 instance. Search and Select EC2.



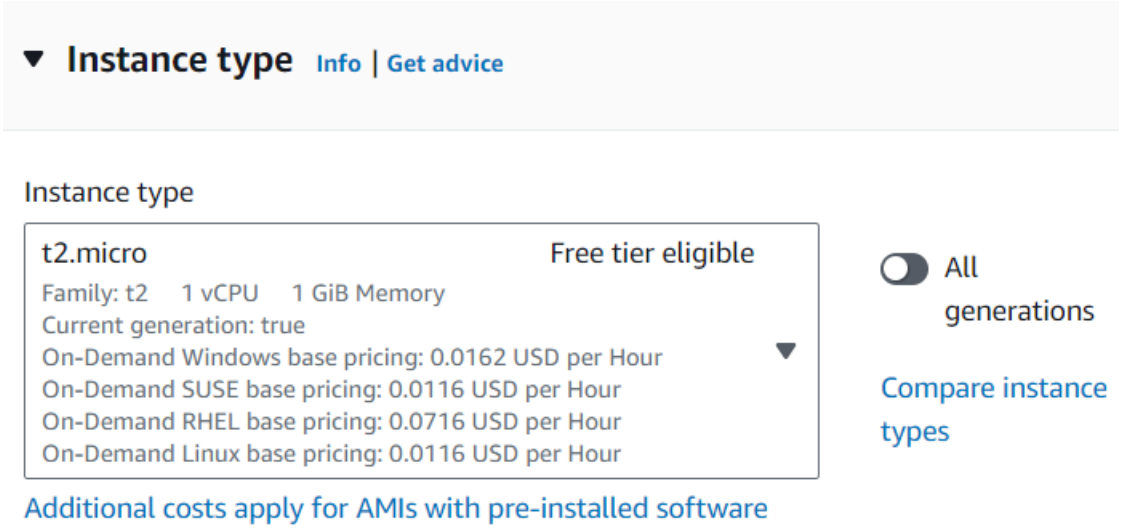
Select Instances and then Launch Instance.



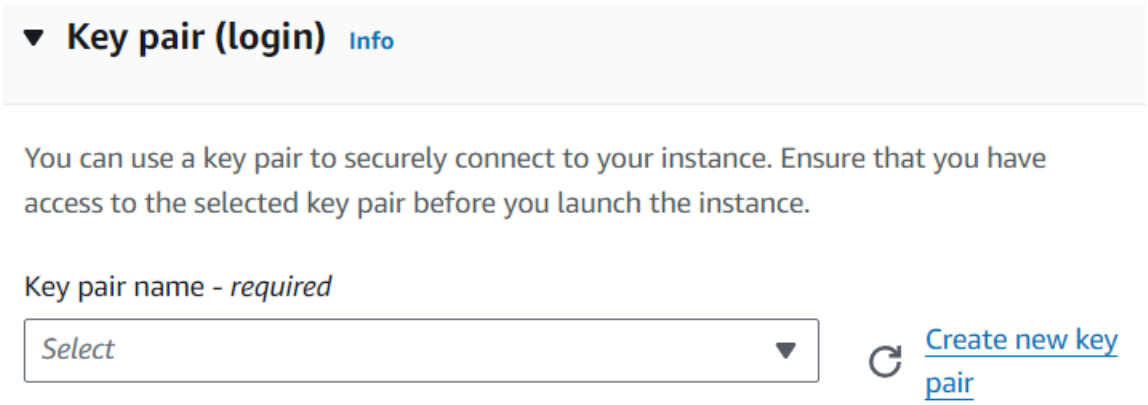
Give your instance a name



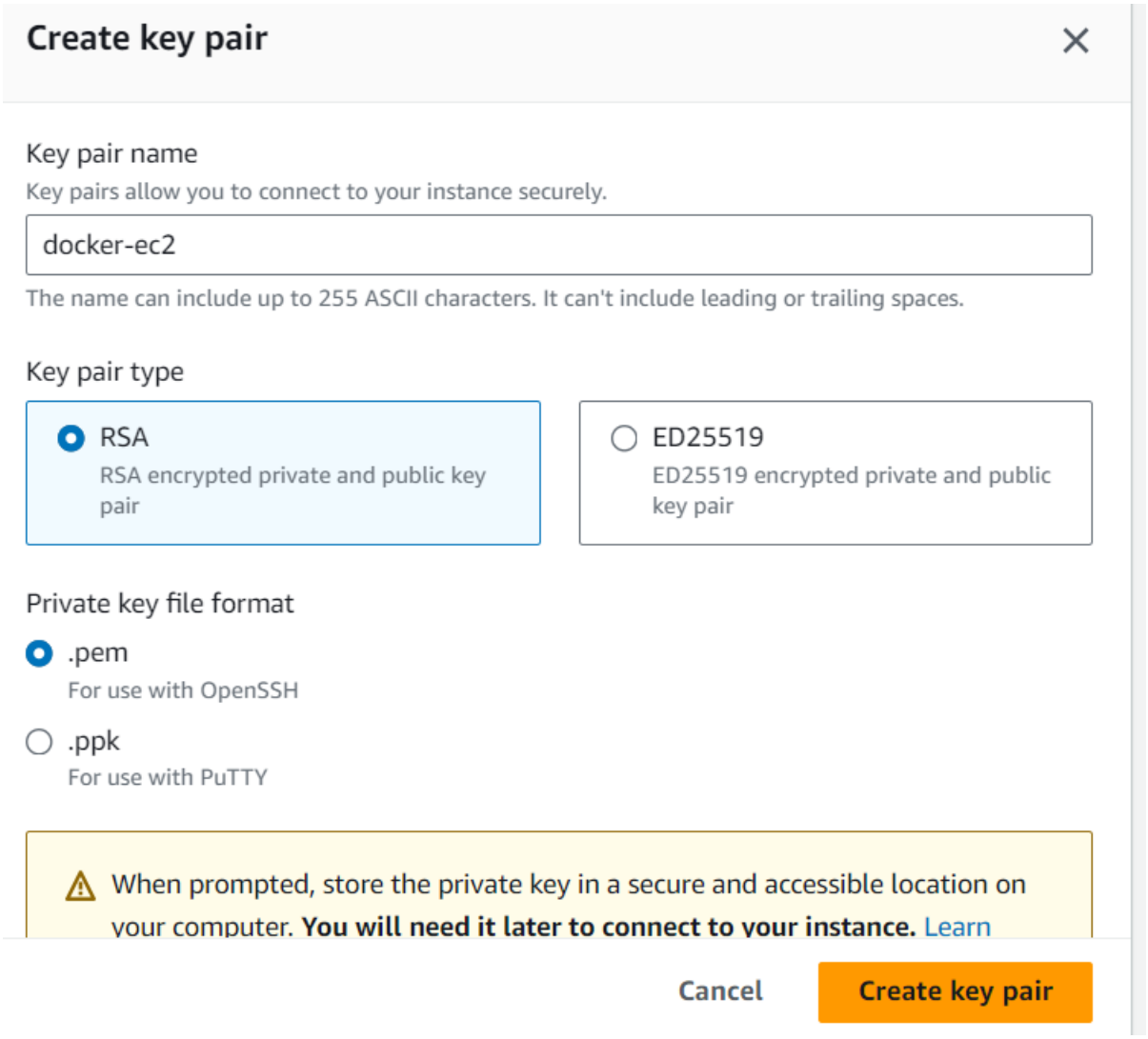
We're going to select the Ubuntu Amazon Machine Image.



For instance type, we are going to keep t2.micro



We use the Key pair to securely login to our ec2 instances. You can create or import a key pair that allows us to securely connect to the instance via SSH. Select generate a new key pair. The key pair consists of a public key that you place on the instance and a private key that you keep safe and secure.



Create a name for the Key pair. RSA Key pair type and private key file format leave as .pem and click create new key pair. A .pem file should show in your downloads. We will need to open the file later on.

Network settings Info

VPC - *required* Info

vpc-0c8b101705441d97c (default)
172.31.0.0/16

Subnet Info

subnet-0747cfb59a9e63ba1
VPC: vpc-0c8b101705441d97c Owner: 992382453401
Availability Zone: us-east-1a IP addresses available: 4091
CIDR: 172.31.16.0/20)

Create new subnet

Auto-assign public IP Info

Enable

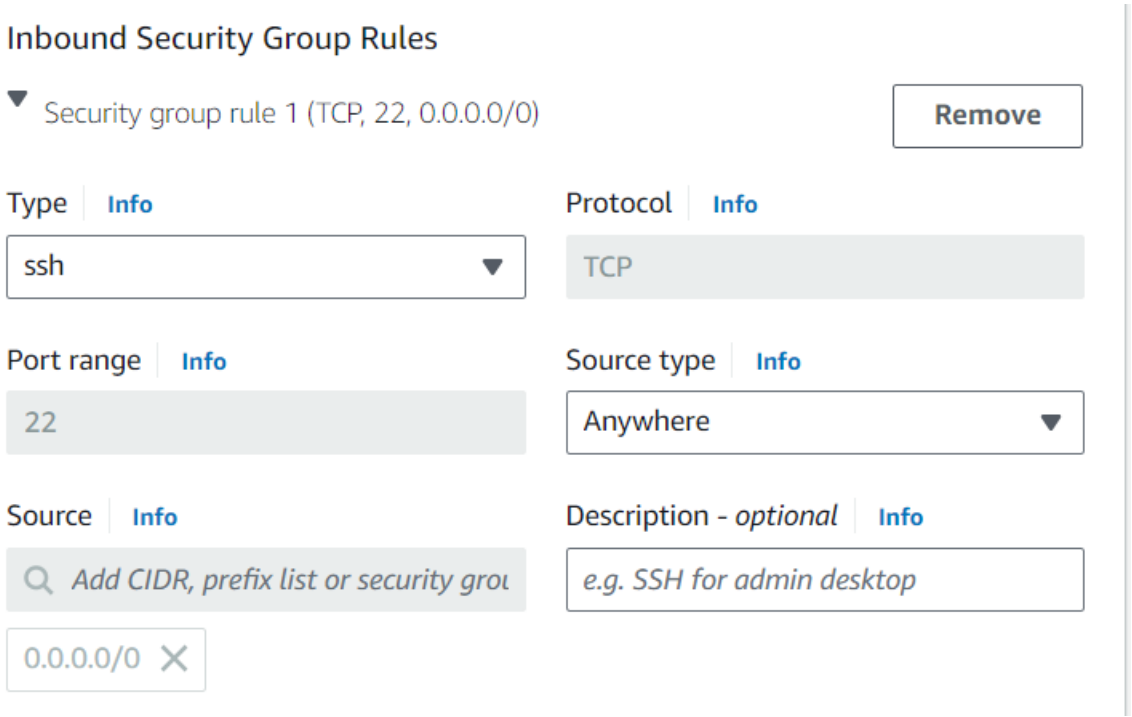
Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

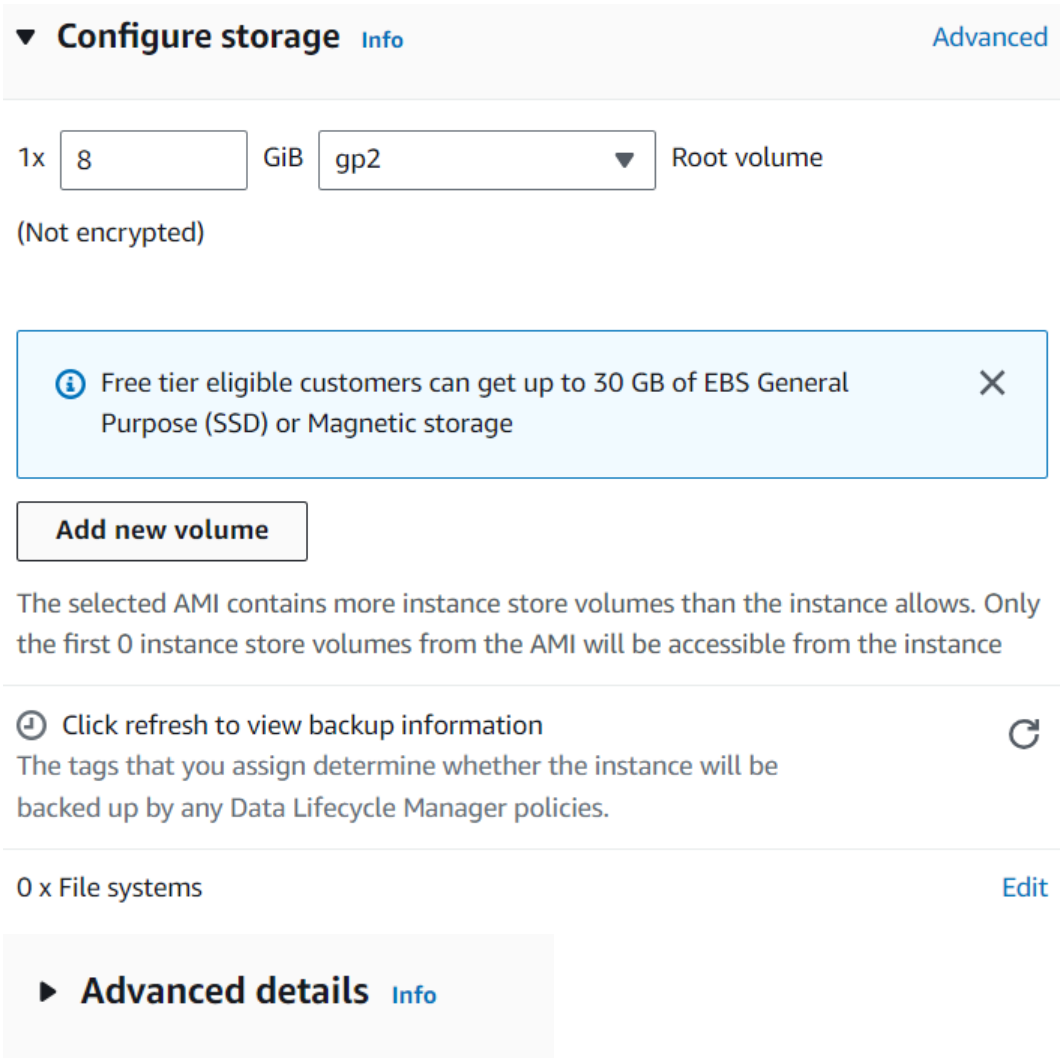
- (●) Create security group
- ( ) Select existing security group

Security group name - *required*

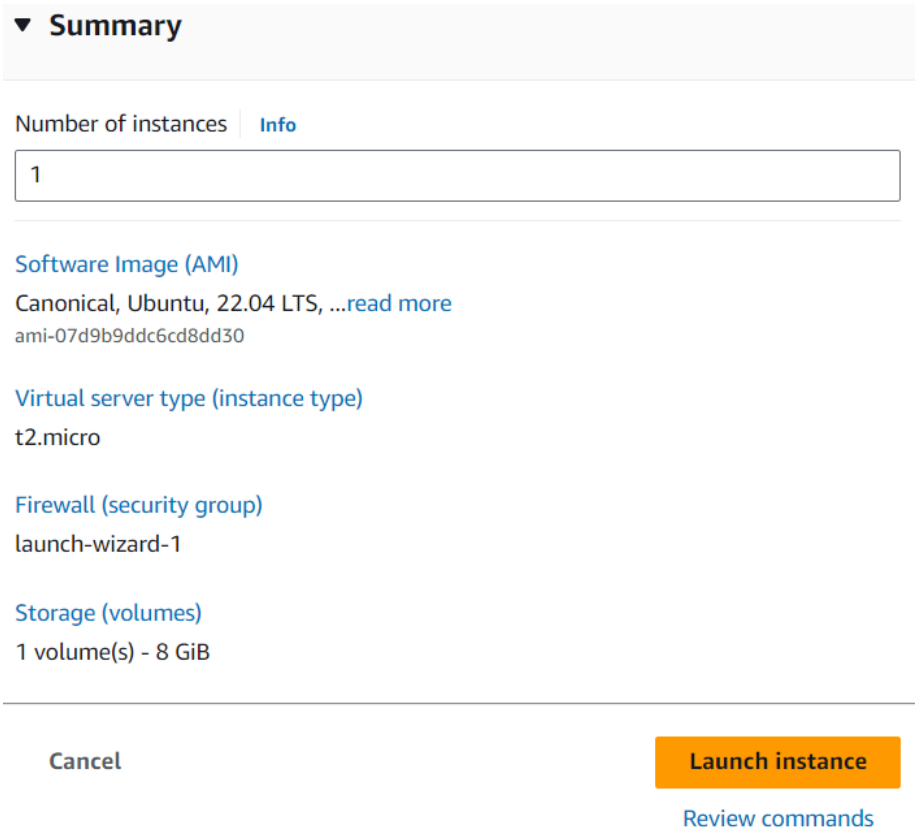
sg-dockerec2



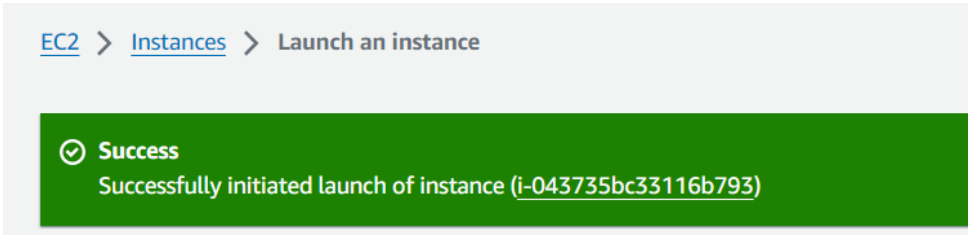
Security groups should be enabled so you can SSH from your local machine. Here we are selecting an existing security group (launch-wizard-1). This security group type is SSH, protocol tcp and is using Port range 22.

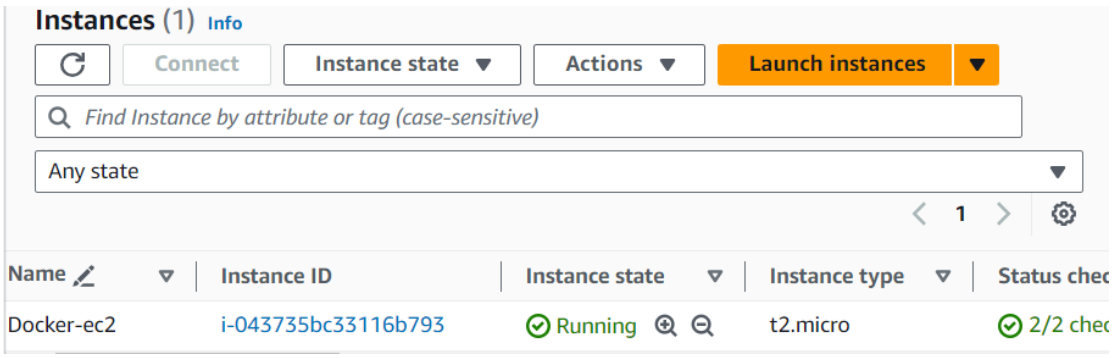


We will leave the storage settings and advanced details as is.

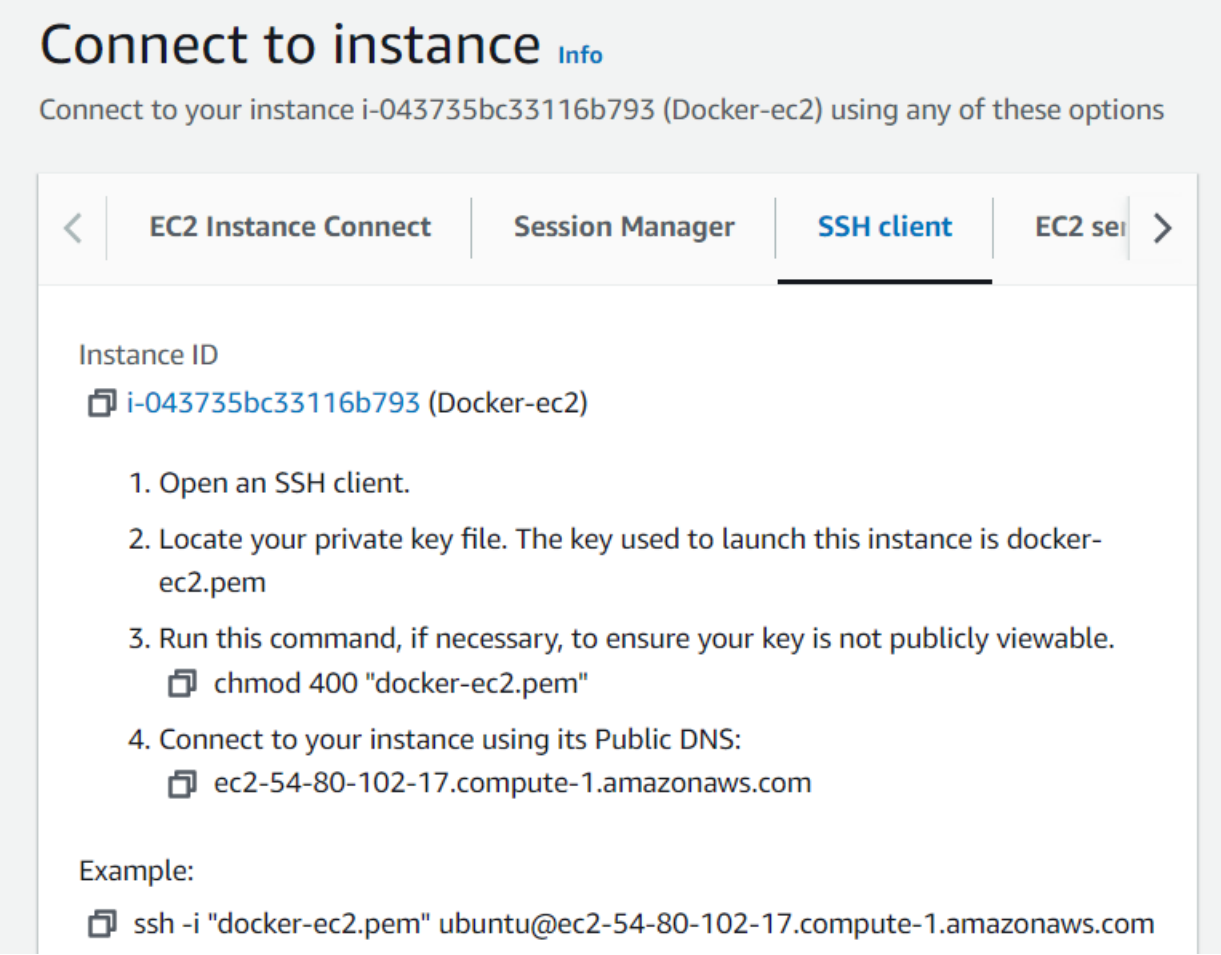


Click on Launch instance.

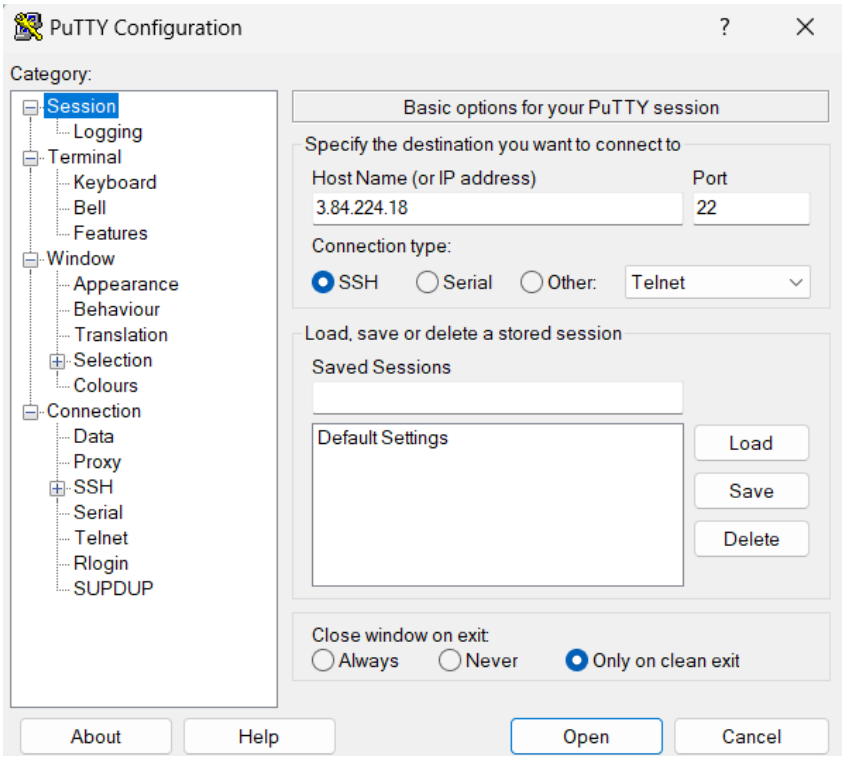




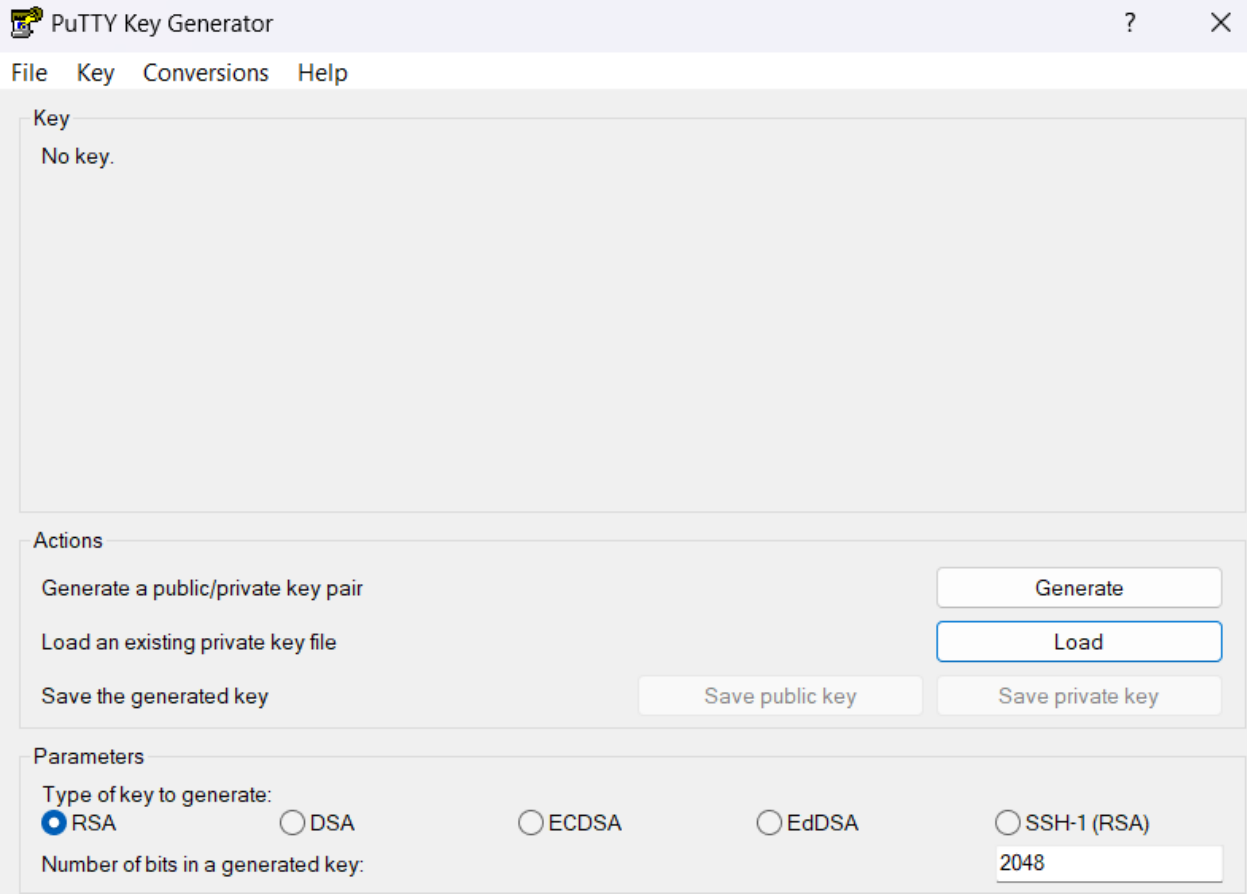
The instance has been successfully created and is running and we can now SSH to our instance from our local machines. Click on Connect.



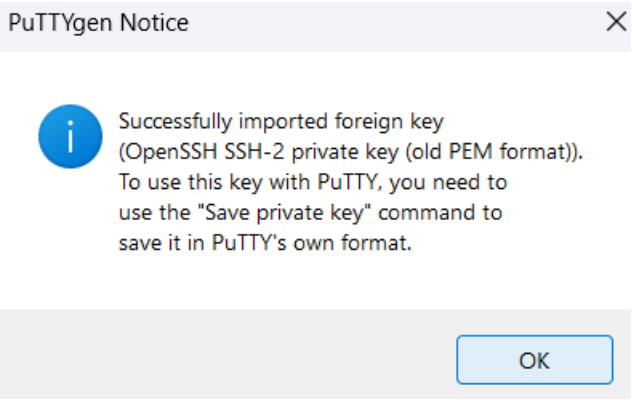
We will use the example command that displays “docker-ec2.pem” which is the name of your key pair file that we created and downloaded onto our local host. The rest of the command shows the name of the user@the publicDNS.

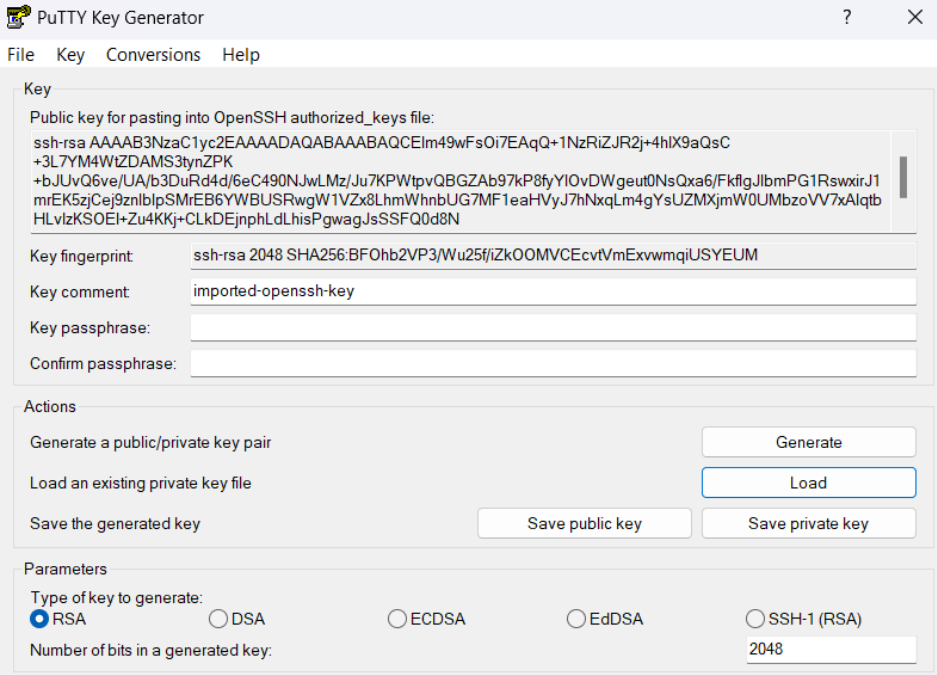


We are going to use Putty as our SSH client. Input your instance IP address. Use this link to install putty onto your host machine if you do not already have it installed https://www.puttygen.com/download.php?val=4

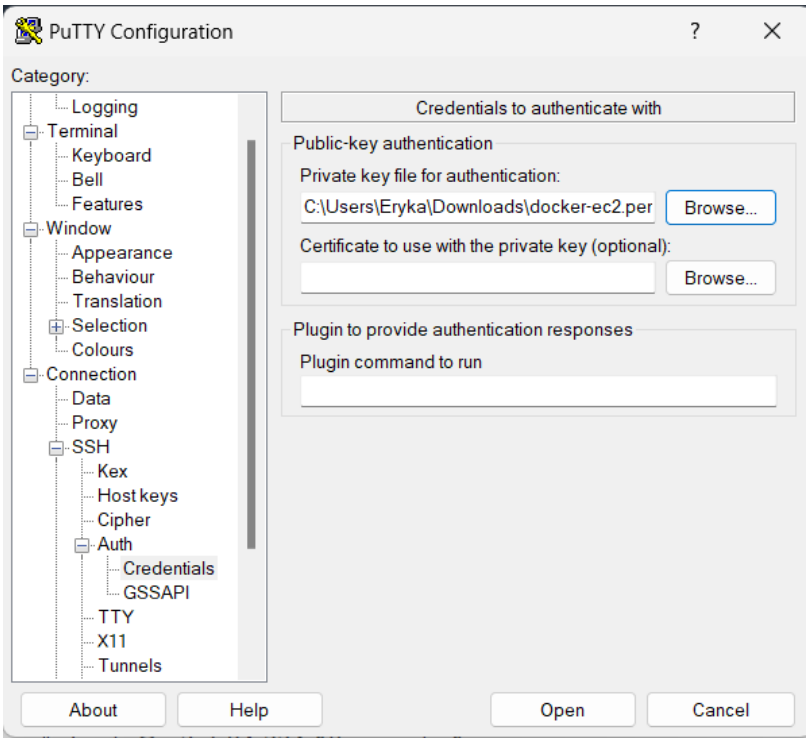


Open the Putty key generator and load your .pem file to convert it into a .ppk file for Putty

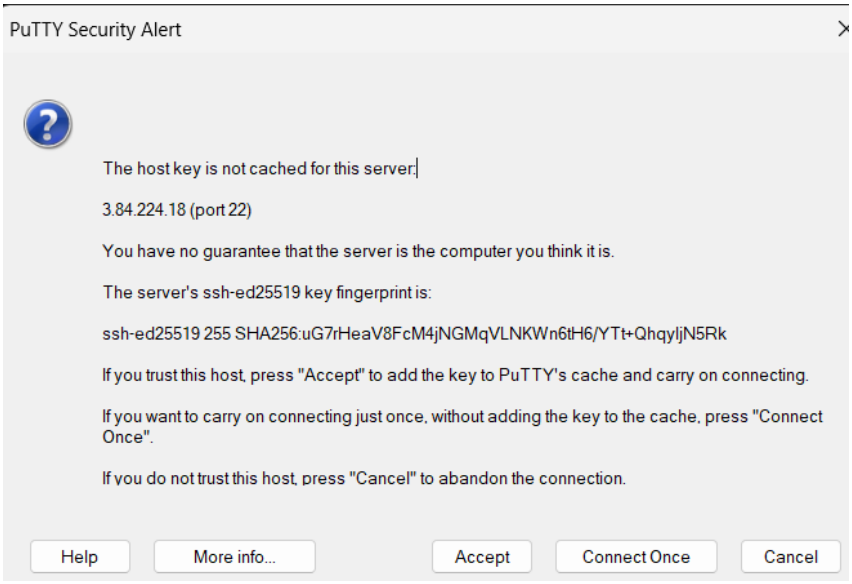




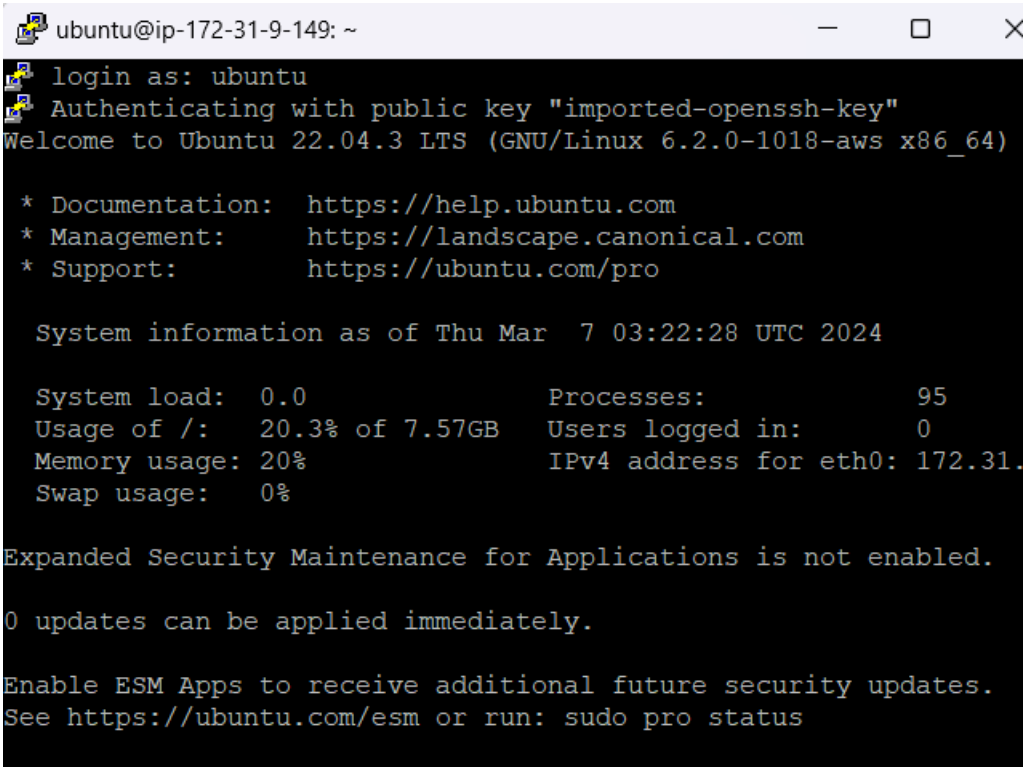
Save the key onto your host machine. You should see the file now show as a .ppk file which can now be uploaded onto Putty



Navigate to SSH ---> Auth ---> Credentials. Then browse for the key pair converted .pem file that was downloaded onto your host machine.



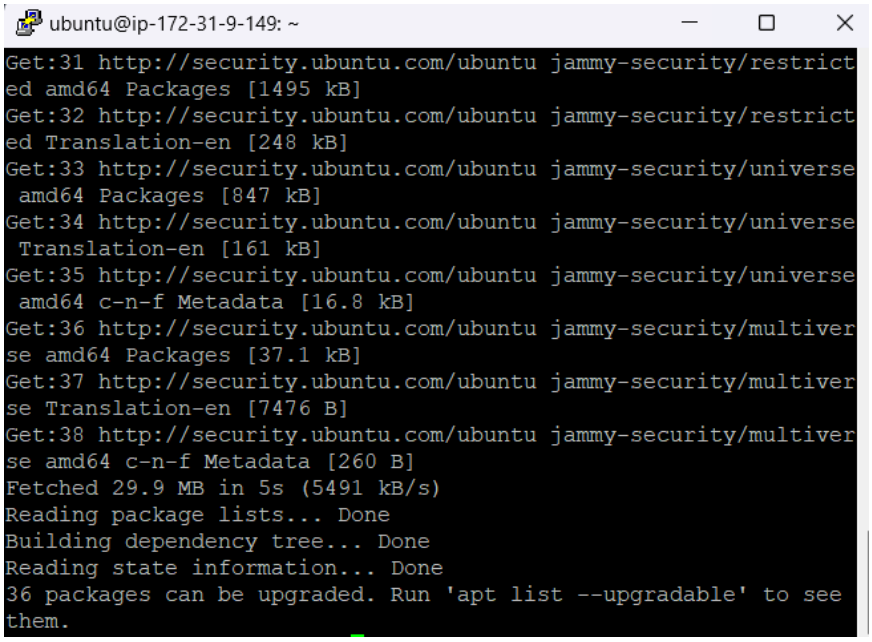
Click Accept

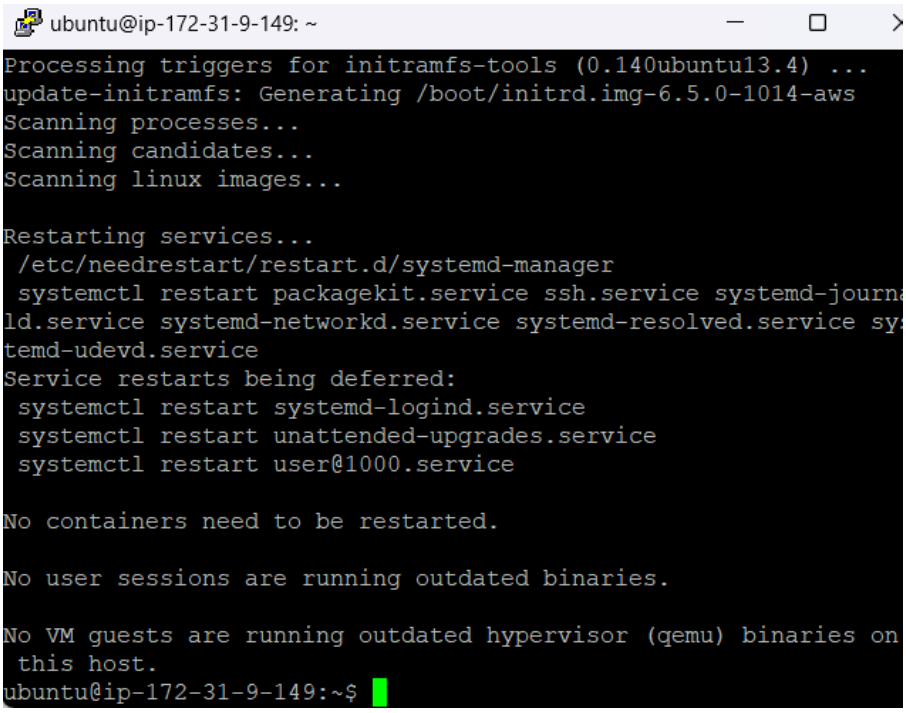


```
The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in t
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitte
applicable law.

To run a command as administrator (user "root"), use "sudo <comm
See "man sudo_root" for details.

ubuntu@ip-172-31-9-149:~$
ubuntu@ip-172-31-9-149:~$
```

```
ubuntu@ip-172-31-9-149:~$ pwd
/home/ubuntu
```

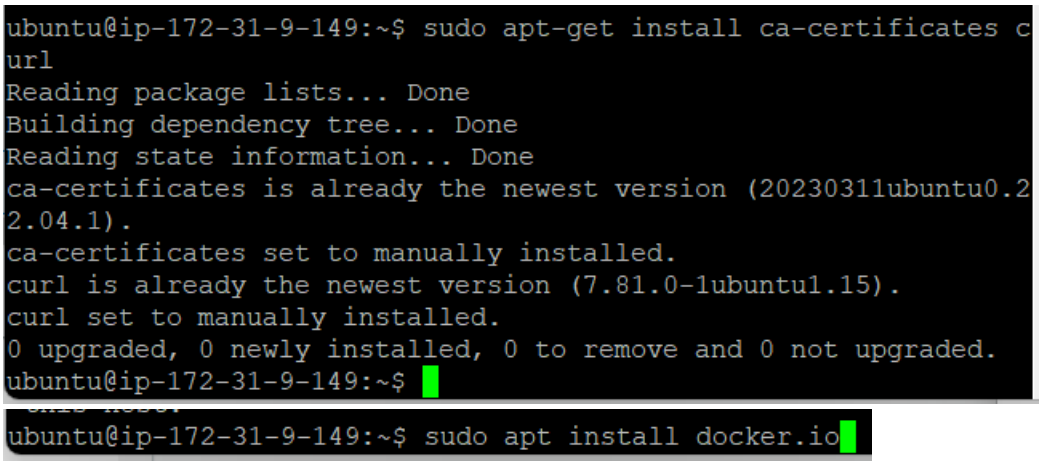
b.



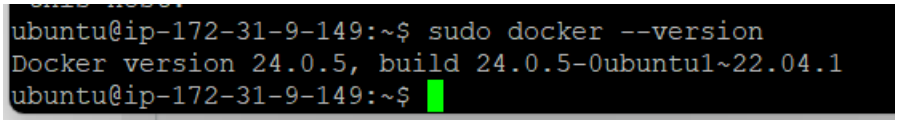


After logging in to the Ubuntu virtual machine, open a terminal window and run the following commands to update the package lists and upgrade existing packages:

sudo apt update
sudo apt upgrade

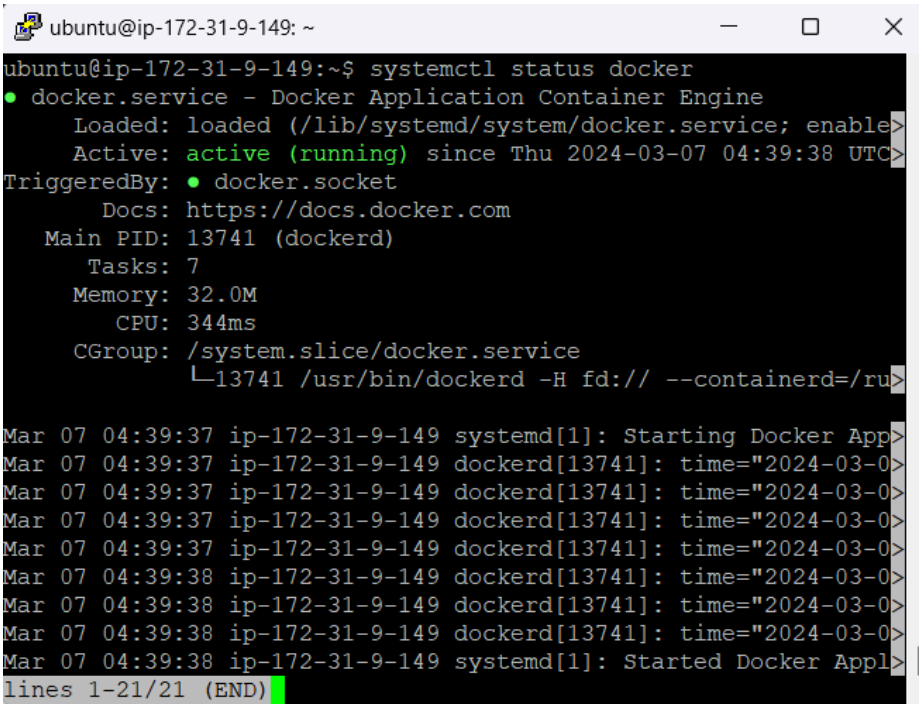


Run the command “sudo apt install docker.io” in order to install docker.



Run the command “sudo docker –version" to display the version of docker installed.

```
ubuntu@ip-172-31-9-149:~$ sudo service docker start
ubuntu@ip-172-31-9-149:~$
```



To start docker we will run the command “sudo service docker start”. Then run “systemctl status docker”. If Docker has started successfully, you should see output indicating that the service is active and running.

```
ubuntu@ip-172-31-9-149:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:d000bc569937abbe195e20322a0bde6b2922d805332fd6d8a
68b19f524b7d21d
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Do
```

This command also shows that docker is successfully running.

```
ubuntu@ip-172-31-9-149:~$ sudo docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55
bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
```

We have successfully pulled the centos image.

```
ubuntu@ip-172-31-9-149:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
bccd10f490ab: Pull complete
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb54023
69e421f44eff17e
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

We have successfully pulled the Ubuntu image.