What Is Docker And Why Do We Need It?

The Problem Statement

Let’s consider that you’re working on an application which requires multiple technologies to be used for developing the different components like the Frontend, Backend, Database etc… Everything seems to be good, but while you’re working with all these technologies on your machine, most often there arises a problem where one technology needs one version of a dependency and some other technology needs a different one to work. Adding to this, it might also happen that your application may not work the same way on someone else’s machine or when you deploy it to different environments with different OS or hardware, it fails to run.

In order to solve all these frequent problems we use something called virtualization, let’s discuss this in more detail

The Solution: Virtualization

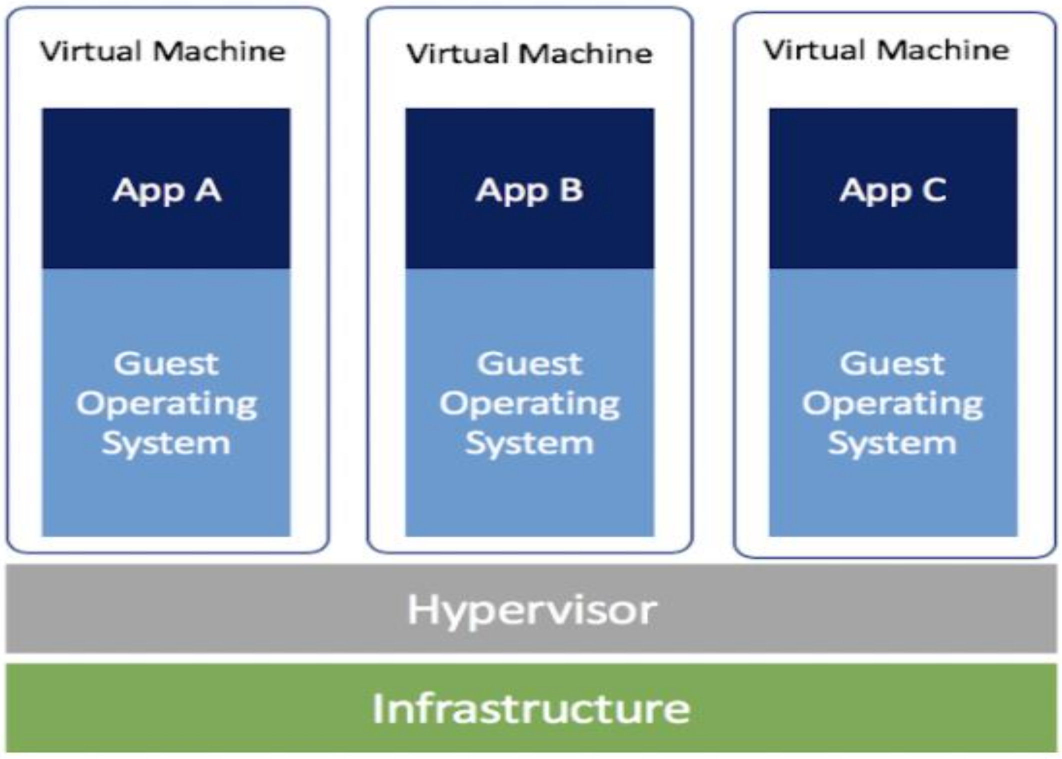
Virtualization is a process whereby a software is used to create an abstraction layer over computer hardware that allows the hardware elements of a single computer to be divided into multiple virtual computers. The main idea of virtualization would be to isolate the components of our application and their dependencies into individual self-contained units that can run anywhere without any dependency or OS conflicts.

Now that we know what virtualization means, we can use it in order to solve our above stated problem. There are two ways to achieve this by leveraging the concept virtualization:

1. Virtual Machines

A Virtual Machine is essentially an emulation of a real computer that executes programs like a real computer. Virtual machines run on top of a physical machine using a ***Hypervisor***. A hypervisor, in turn, runs on a host machine.

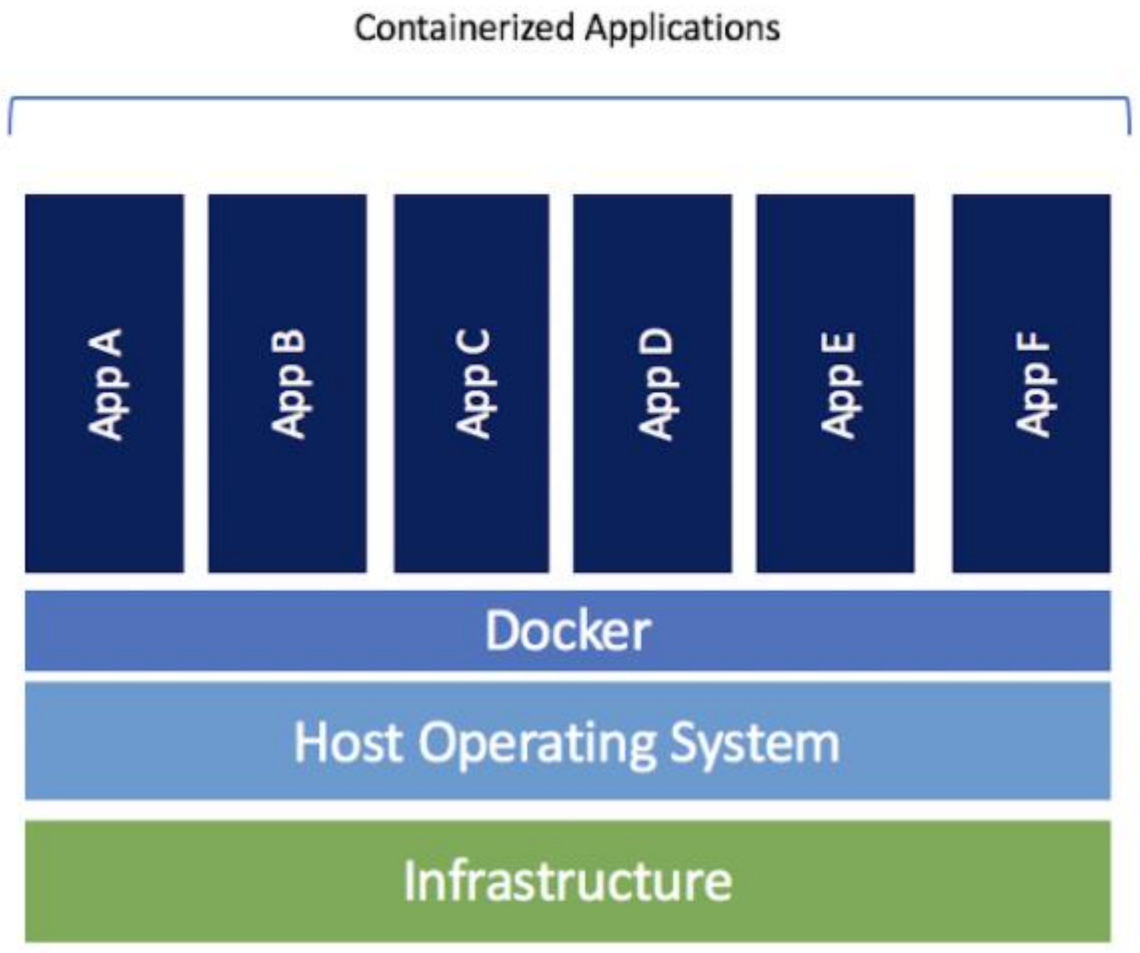
A ***Hypervisor*** is a piece of software, firmware, or hardware that a virtual machine runs on top of. The host machine provides the virtual machines with resources, including RAM and CPU. These resources are divided between virtual machines and can be distributed based on the applications that run on individual virtual machines.



Even though, virtual machines have solved the problem by now making our applications to run in isolation, with their own set of dependencies/libraries and OS requirements, the main issue here is that they're very heavy since each virtual machine has it's own Guest operating system thereby consuming a higher portion of the host machine's resources and so take up a lot of time to start or create. This is where containers come to the rescue.

2. Containers

Containers are a light-weight, more agile way of handling virtualization, and since they don't use a software like hypervisor, you can enjoy faster resource provisioning and speedier availability of new applications. Unlike virtual machines which provide a hardware level virtualization, containers provide an operating-system level virtualization due to which they're very simple and easy to work with.



Since containers are lightweight, they consume a lot less resources of the host machine compared to virtual machines. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way irrespective of version conflicts between dependencies or OS.

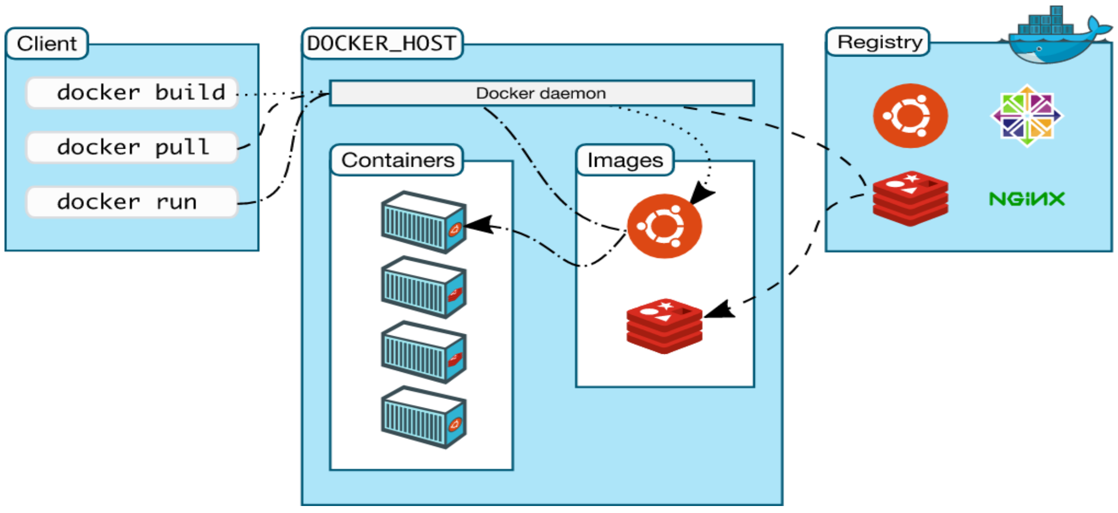
Now What is Docker?

Docker is a tool which helps in developing, shipping, and running your applications on containers and enables you to separate your applications from your infrastructure so you can deliver software quickly.

It provides the ability to package and run an application in an isolated environment i.e; containers. The isolation and security allows you to run many containers simultaneously on a given host.

Docker architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



Docker Desktop
Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (`dockerd`), the Docker client (`docker`), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries
A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

Docker Engine

The **Docker Engine** is the core containerization technology responsible for creating and managing all the containers and other Docker objects. It acts a client-server application consisting of:

1. The **Docker Daemon**, which is a daemon process that runs in the background, keeps listening for any API requests and manages the Docker objects accordingly
2. A set of **APIs** in order to communicate with the Docker daemon
3. The **Docker CLI client** which helps users to communicate with the docker daemon and carry out the user's requests by using these Docker APIs

Docker Image

A Docker image is just a template which includes a set of instructions or commands that are used in order to create the actual container. We can use the same docker image to create multiple docker containers

Docker Container

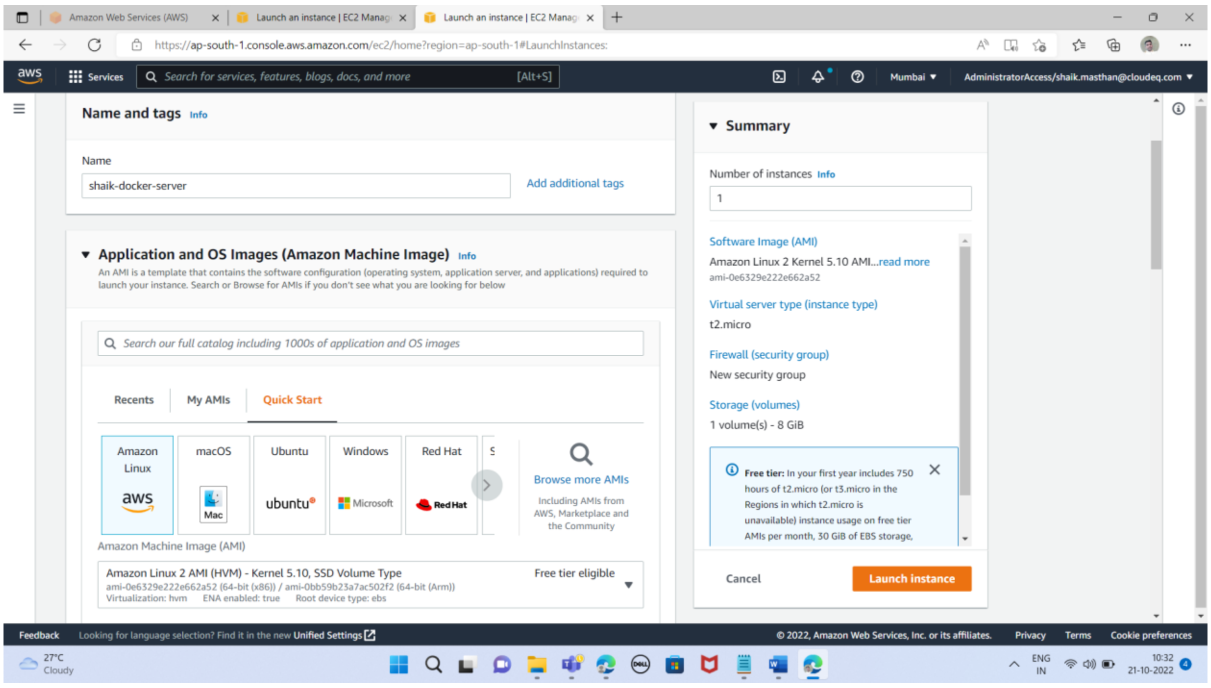
A Docker container is the actual instance which is based off of an image, and packages all the dependencies, libraries that an application needs and runs it in loosely coupled isolation

EC2 creation

Step1:

Now go to AWS Console search on ec2 create one instance

Given name on instance and select image as per requirement and number or instance:1


https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:

Name and tags

Name: shaik-docker-server

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type — Free tier eligible
ami-0e6329e222e662a52 (64-bit (x86)) / ami-0bb59b23a7ac502f2 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Summary

Number of instances: 1

Software Image (AMI): Amazon Linux 2 Kernel 5.10 AMI...read more
ami-0e6329e222e662a52

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

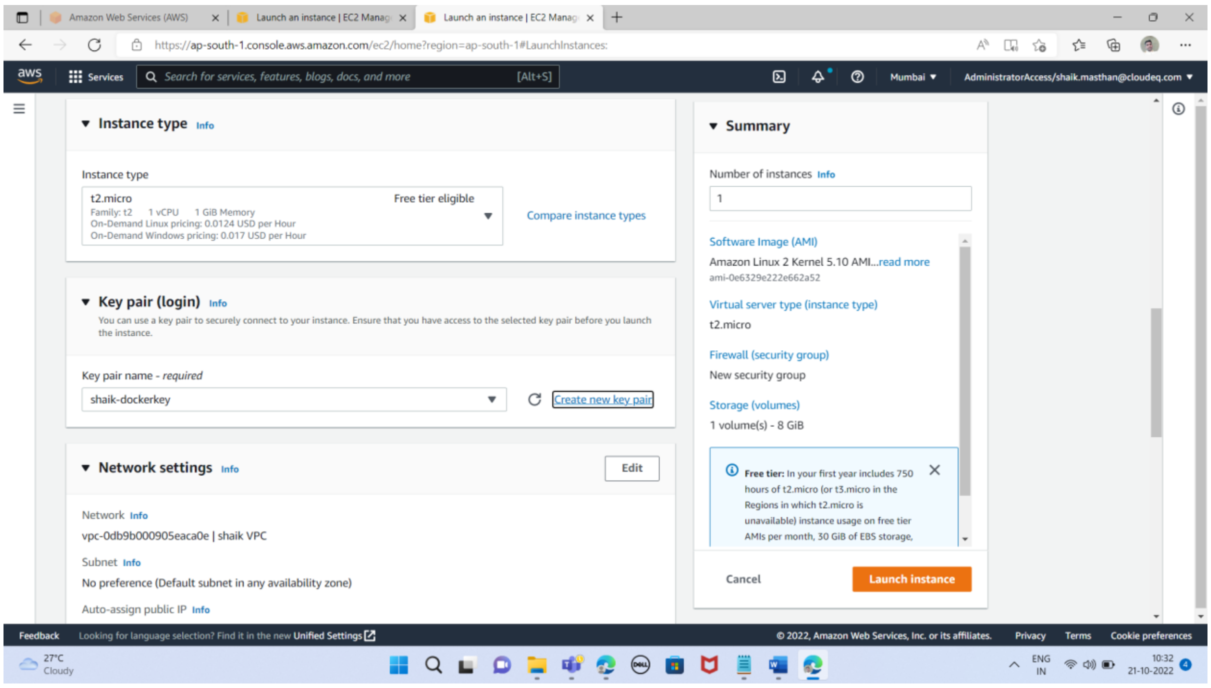
Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage,

Amazon Web Services (AWS)
Launch an instance | EC2 Manag
Services
Search for services, features, blogs, docs, and more [Alt+S]
Mumbai
AdministratorAccess/shaik.masthan@cloudeq.com
Info
Add additional tags
Search our full catalog including 1000s of application and OS images
Recents
My AMIs
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Cancel
Launch instance
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
27°C Cloudy
10:32 21-10-2022

Step-2

Select t2.micro instance ,Create one key pair and remaining are takes default



https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:

Instance type Info

Instance type
t2.micro — Free tier eligible
Family: t2 1 vCPU 1 GiB Memory
On-Demand Linux pricing: 0.0124 USD per Hour
On-Demand Windows pricing: 0.017 USD per Hour

Compare instance types

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*
shaik-dockerkey

Create new key pair

Network settings Info — Edit

Network Info
vpc-0db9b000905eaca0e | shaik VPC

Subnet Info
No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Summary

Number of instances Info
1

Software Image (AMI)
Amazon Linux 2 Kernel 5.10 AMI...read more
ami-0e6329e222e662a52

Virtual server type (instance type)
t2.micro

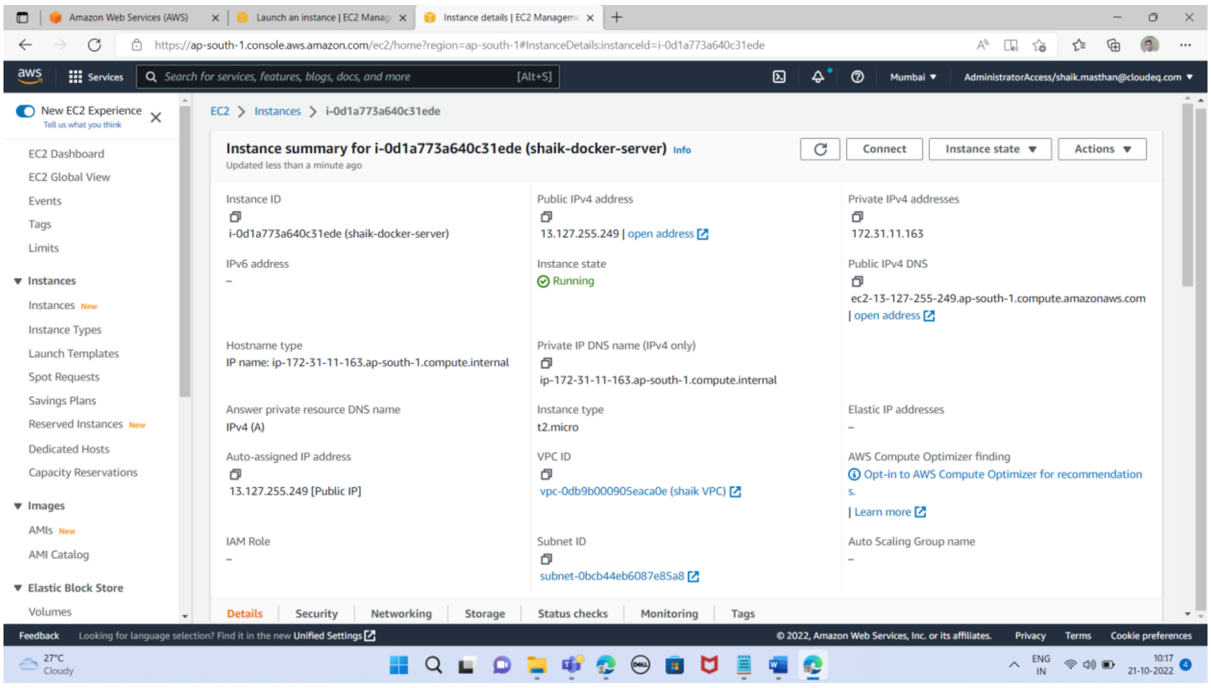
Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage,

Cancel — Launch instance

Now finally launch the instance, now I have launched ec2 instance successful



https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#InstanceDetails:instanceId=i-0d1a773a640c31ede

EC2 > Instances > i-0d1a773a640c31ede

Instance summary for i-0d1a773a640c31ede (shaik-docker-server) Info
Updated less than a minute ago

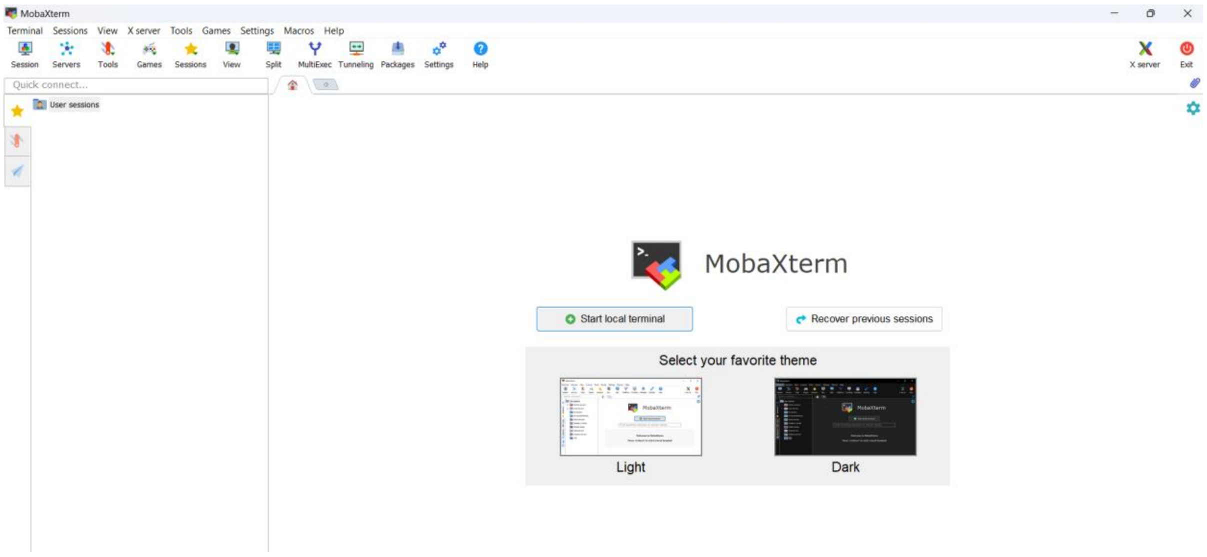
Connect — Instance state — Actions

Instance ID i-0d1a773a640c31ede (shaik-docker-server)	Public IPv4 address 13.127.255.249 | open address	Private IPv4 addresses 172.31.11.163
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-13-127-255-249.ap-south-1.compute.amazonaws.com | open address
Hostname type IP name: ip-172-31-11-163.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-11-163.ap-south-1.compute.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address 13.127.255.249 [Public IP]	VPC ID vpc-0db9b000905eaca0e (shaik VPC)	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID subnet-0bcb44eb6087e85a8	Auto Scaling Group name –

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

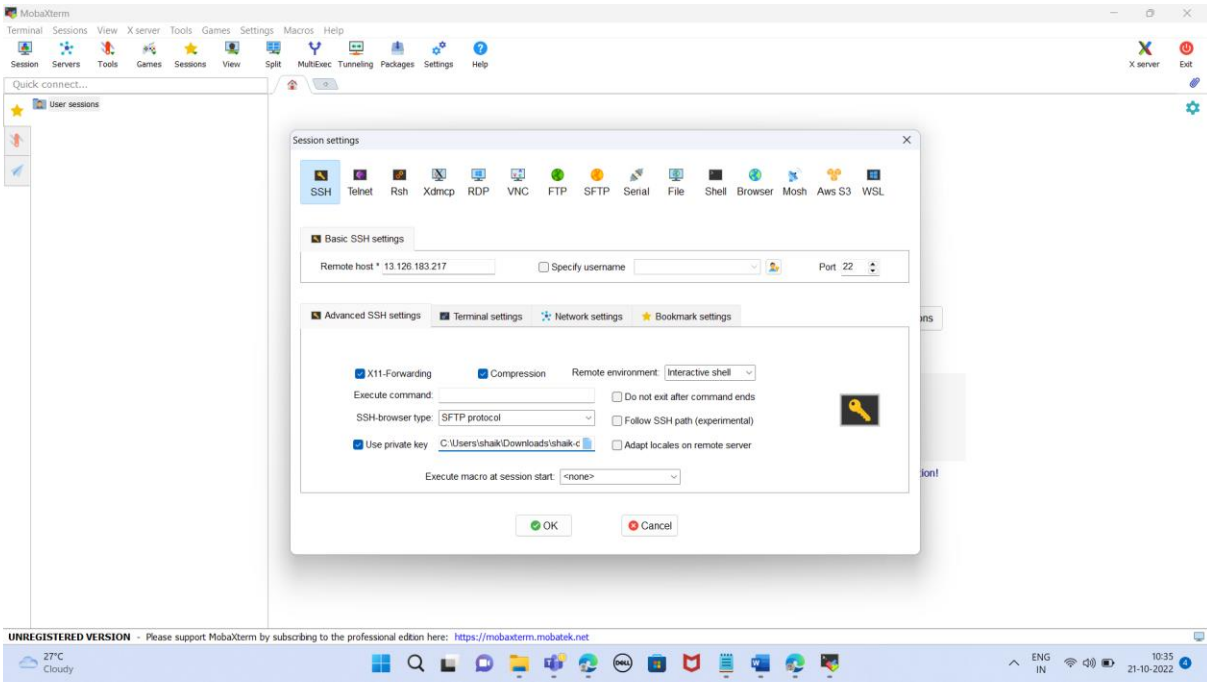
Step-3

Now I have installed MobaXterm


MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
User sessions
MobaXterm
Start local terminal
Recover previous sessions
Select your favorite theme
Light
Dark

Step-4

Now I want to connect that instance using mobaXterm using public ip


MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
User sessions
Session settings
SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL
Basic SSH settings
Advanced SSH settings Terminal settings Network settings Bookmark settings
OK Cancel

Remote host * 13.126.183.217 ☐ Specify username Port 22

☑ X11-Forwarding ☑ Compression Remote environment: Interactive shell

Execute command:

☐ Do not exit after command ends

SSH-browser type: SFTP protocol

☐ Follow SSH path (experimental)

☑ Use private key C:\Users\shaik\Downloads\shaik-c

☐ Adapt locales on remote server

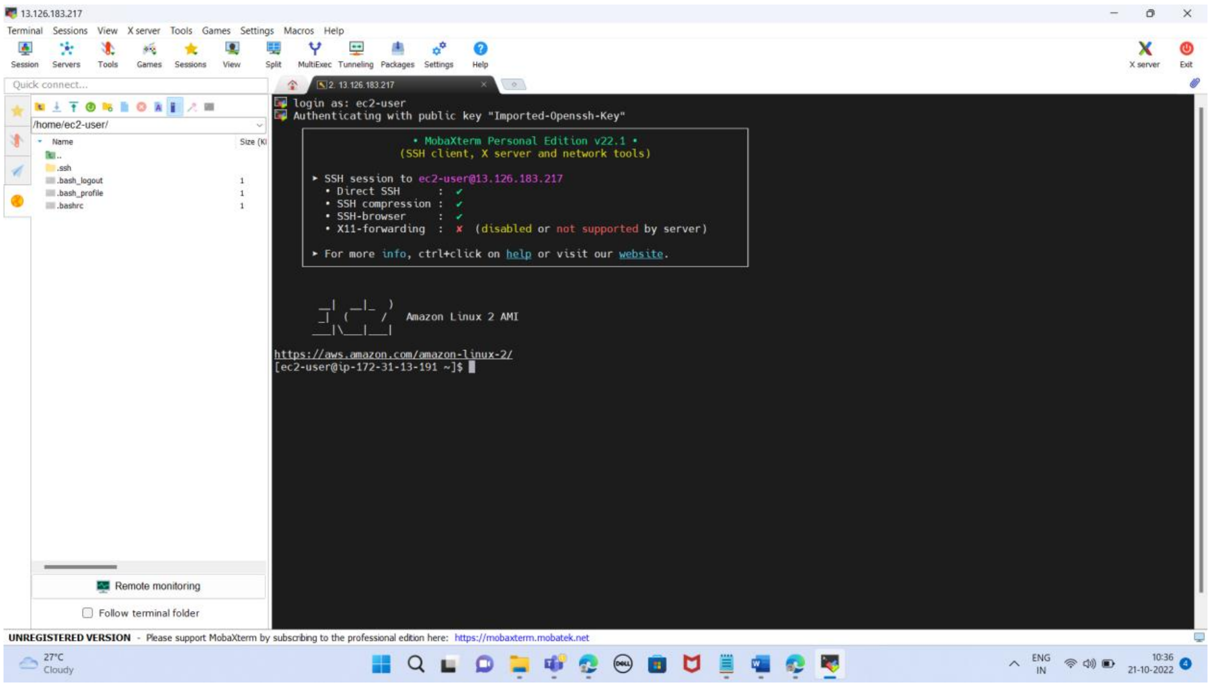
Execute macro at session start: <none>

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

27°C Cloudy ENG IN 10:35 21-10-2022

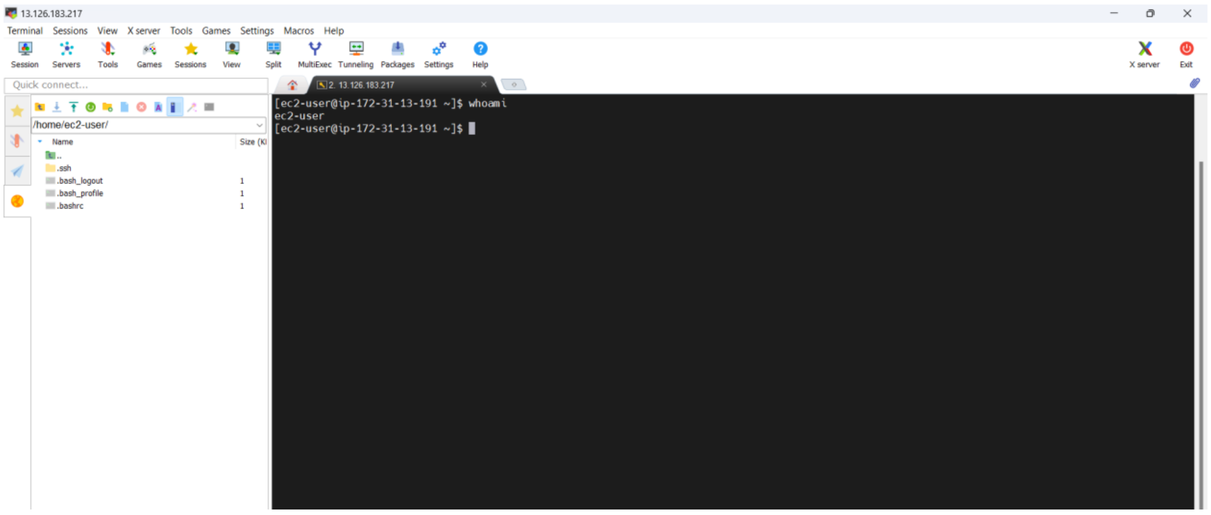
Step-5

After connect asking username-ec2user



```
login as: ec2-user
Authenticating with public key "Imported-Openssh-Key"

        • MobaXterm Personal Edition v22.1 •
      (SSH client, X server and network tools)

 ▸ SSH session to ec2-user@13.126.183.217
   • Direct SSH      :  ✔
   • SSH compression :  ✔
   • SSH-browser     :  ✔
   • X11-forwarding  :  ✘  (disabled or not supported by server)

 ▸ For more info, ctrl+click on help or visit our website.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-13-191 ~]$
```

Now I have successful connected ec2 instance



```
[ec2-user@ip-172-31-13-191 ~]$ whoami
ec2-user
[ec2-user@ip-172-31-13-191 ~]$
```