What is virtualization?

Virtualization began in the 1960s, as a method of logically dividing the system resources provided by mainframe computers between different applications (the definition has now changed).

What is hardware virtualization?

Hardware virtualization refers to the creation of a *virtual machine* that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources. For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.

In hardware virtualization, the *host machine* is the machine that is used by the virtualization and the *guest machine* is the virtual machine. The words host and guest are used to distinguish the software that runs on the physical machine from the software that runs on the virtual machine. The software or firmware that creates a virtual machine on the host hardware is called a *hypervisor* or *virtual machine monitor*.

Virtualization refers to importing a guest operating system on the host operating system, allowing developers to run multiple OS on different VMs while all of them run on the same host, thereby eliminating the need to provide extra hardware resources.

What is a snapshot?

A snapshot is a state of a virtual machine, and generally its storage devices, at an exact point in time. A snapshot enables the virtual machine's state at the time of the snapshot to be restored later, effectively undoing any changes that occurred afterwards. This capability is useful as a backup technique, for example, prior to performing a risky operation.

What is migration?

The snapshots described above can be moved to another host machine with its own hypervisor; when the VM is temporarily stopped, snapshotted, moved, and then resumed on the new host, this is known as migration.

What is failover?

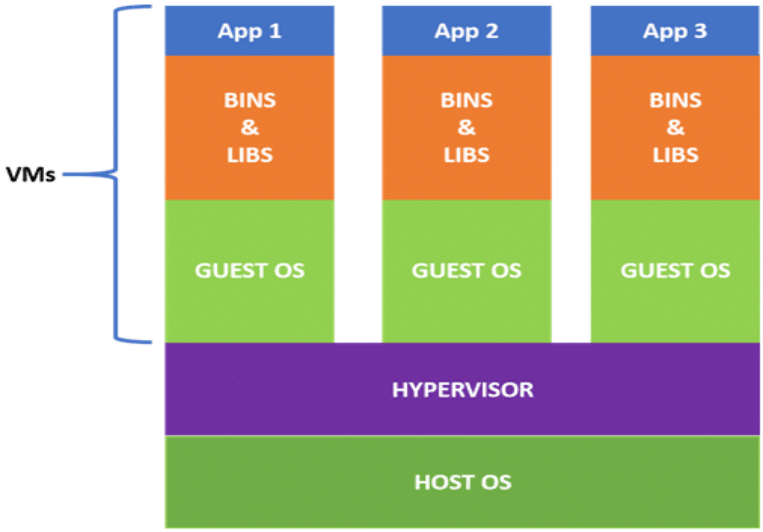
Like the migration mechanism described above, failover allows the VM to continue operations if the host fails. Generally, it occurs if the migration has stopped working.

Advantages of Virtualization:

VMs are being used in the industry in the following ways:

- Enable multiple OS on the same machine
- Cheaper due to less/compact infrastructure setup
- Easy to recover in case of failure
- Easy for maintenance
- Faster provisioning of applications and resources required for tasks
- Increase in IT productivity

Virtualization Architecture:



What is a virtualization host?

In above, the three guest operating systems acting as virtual machines are running on a host operating system. The process of manually reconfiguring hardware and firmware and installing a new OS can be entirely automated; all these steps get stored as data in any files of a disk.

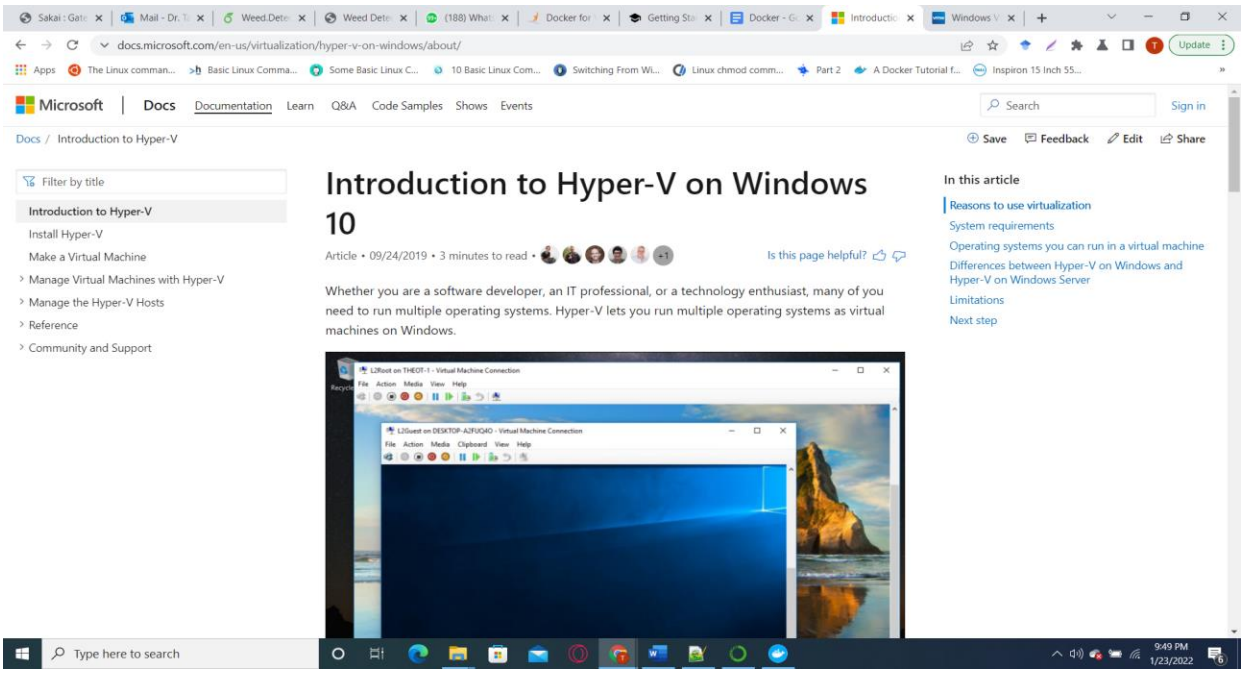
Virtualization lets us run our applications on fewer physical servers. In virtualization, each application and operating system live in a separate software container called VM. Where VMs are completely isolated, all the computing resources like CPUs, storage, and networking are pooled together, and they are delivered dynamically to each VM by a software called a hypervisor.

However, running multiple VMs over the same host leads to degradation in performance. As guest operating systems have their own kernel, libraries, and many dependencies running on a single host OS, it takes up a large occupation of resources such as the processor, hard disk and, especially, its RAM. Also, when we use VMs in virtualization, the bootup process takes a long time that would affect efficiency in the case of real-time applications.

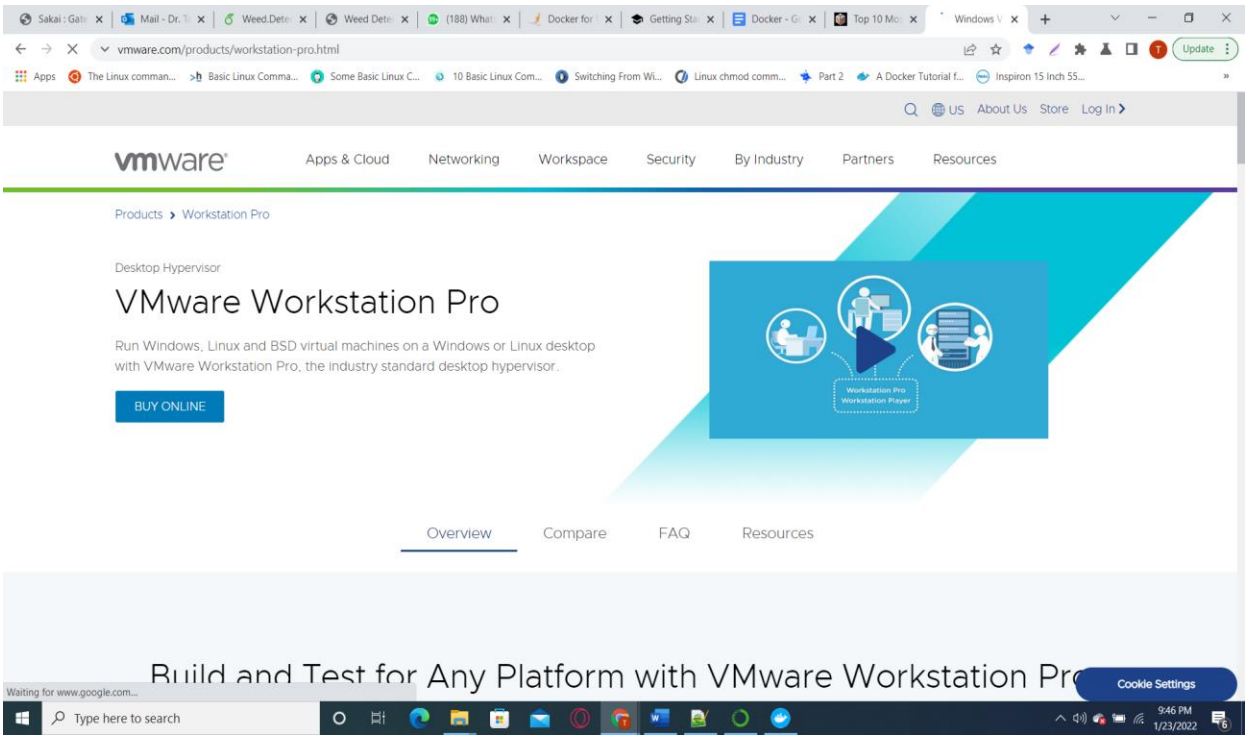
What is the difference between Type-I and Type-II hypervisor?

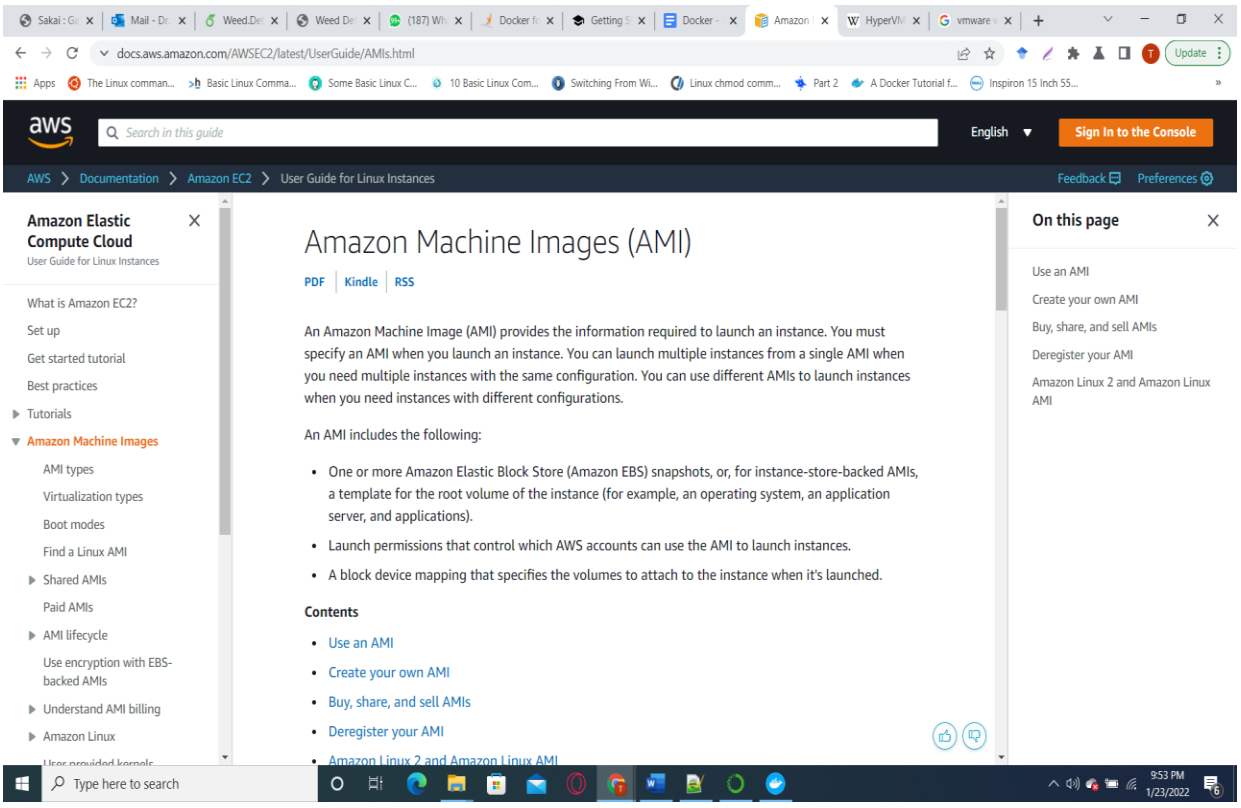
Type 1 hypervisor is a hypervisor that runs directly on the host's hardware to control the hardware and to manage guest operating systems while Type 2 hypervisors run on a conventional operating system just as other computer programs do.

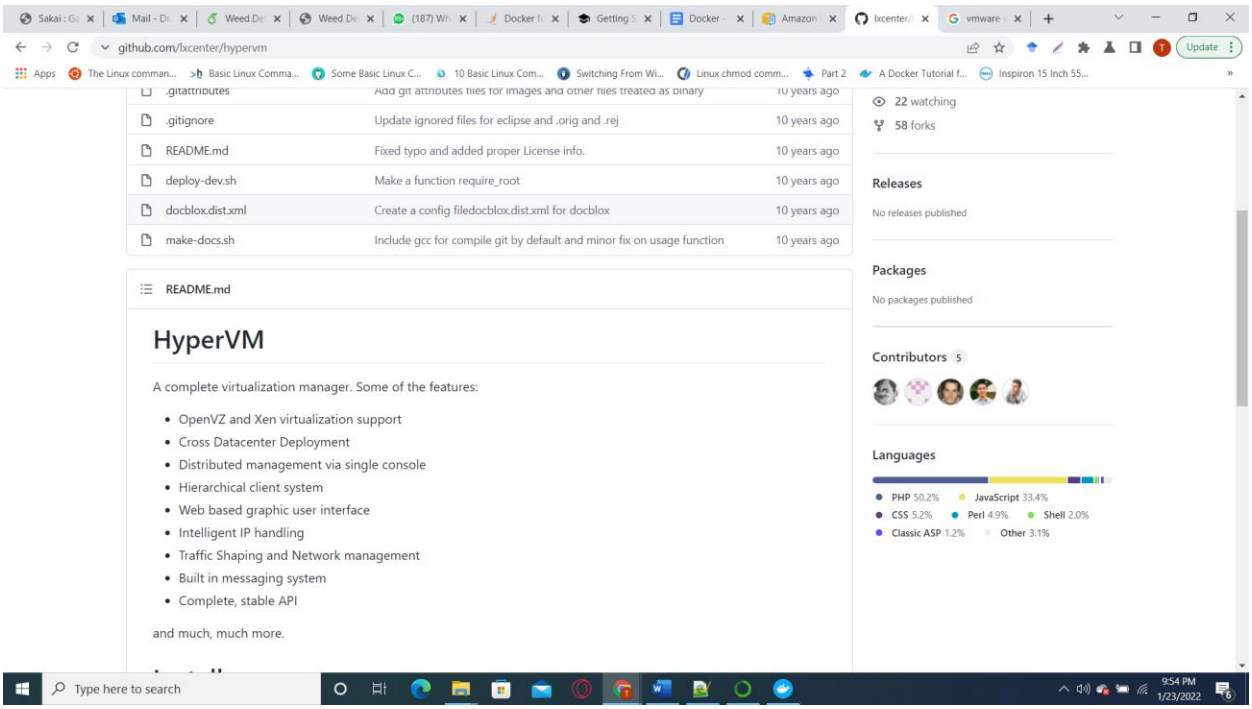
Examples:


docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/
Microsoft | Docs
Documentation Learn Q&A Code Samples Shows Events
Search
Sign in
Docs / Introduction to Hyper-V
Save Feedback Edit Share
Filter by title
Introduction to Hyper-V
Install Hyper-V
Make a Virtual Machine
Manage Virtual Machines with Hyper-V
Manage the Hyper-V Hosts
Reference
Community and Support
Introduction to Hyper-V on Windows 10
Article • 09/24/2019 • 3 minutes to read •
Is this page helpful?
Whether you are a software developer, an IT professional, or a technology enthusiast, many of you need to run multiple operating systems. Hyper-V lets you run multiple operating systems as virtual machines on Windows.
In this article
Reasons to use virtualization
System requirements
Operating systems you can run in a virtual machine
Differences between Hyper-V on Windows and Hyper-V on Windows Server
Limitations
Next step
Type here to search
9:49 PM
1/23/2022


virtualbox.org
VirtualBox
Welcome to VirtualBox.org!
search...
Login Preferences
About
Screenshots
Downloads
Documentation
End-user docs
Technical docs
Contribute
Community
VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.
Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.
VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.
Download VirtualBox 6.1
Hot picks:
Pre-built virtual machines for developers at Oracle Tech Network
Hyperbox Open-source Virtual Infrastructure Manager project site
phpVirtualBox AJAX web interface project site
News Flash
Important January 13th, 2022 We're hiring!
Looking for a new challenge? We're hiring a System Administrator/Quality Engineer (Germany).
Important May 17th, 2021 We're hiring!
Looking for a new challenge? We're hiring a VirtualBox senior developer in 3D area (Europe/Russia/India).
New January 18th, 2022 VirtualBox 6.1.32 released!
Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
New November 22nd, 2021 VirtualBox 6.1.30 released!
Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
New October 19th, 2021 VirtualBox 6.1.28 released!
Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
More information...
ORACLE
Contact – Privacy policy – Terms of Use
Type here to search
9:45 PM
1/23/2022


vmware.com/products/workstation-pro.html
vmware
Apps & Cloud
Networking
Workspace
Security
By Industry
Partners
Resources
Products › Workstation Pro
Desktop Hypervisor
VMware Workstation Pro
Run Windows, Linux and BSD virtual machines on a Windows or Linux desktop with VMware Workstation Pro, the industry standard desktop hypervisor.
BUY ONLINE
Overview
Compare
FAQ
Resources
Build and Test for Any Platform with VMware Workstation Pro
Cookie Settings
Type here to search


docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
aws
Search in this guide
English
Sign In to the Console
AWS › Documentation › Amazon EC2 › User Guide for Linux Instances
Feedback
Preferences
Amazon Elastic Compute Cloud
User Guide for Linux Instances
What is Amazon EC2?
Set up
Get started tutorial
Best practices
Tutorials
Amazon Machine Images
AMI types
Virtualization types
Boot modes
Find a Linux AMI
Shared AMIs
Paid AMIs
AMI lifecycle
Use encryption with EBS-backed AMIs
Understand AMI billing
Amazon Linux
Amazon Machine Images (AMI)
PDF
Kindle
RSS
An Amazon Machine Image (AMI) provides the information required to launch an instance. You must specify an AMI when you launch an instance. You can launch multiple instances from a single AMI when you need multiple instances with the same configuration. You can use different AMIs to launch instances when you need instances with different configurations.
An AMI includes the following:
One or more Amazon Elastic Block Store (Amazon EBS) snapshots, or, for instance-store-backed AMIs, a template for the root volume of the instance (for example, an operating system, an application server, and applications).
Launch permissions that control which AWS accounts can use the AMI to launch instances.
A block device mapping that specifies the volumes to attach to the instance when it's launched.
Contents
Use an AMI
Create your own AMI
Buy, share, and sell AMIs
Deregister your AMI
Amazon Linux 2 and Amazon Linux AMI
On this page
Use an AMI
Create your own AMI
Buy, share, and sell AMIs
Deregister your AMI
Amazon Linux 2 and Amazon Linux AMI
Type here to search



Disadvantages of Virtualization:

- Running multiple VMs leads to unstable performance
- Hypervisors are not as efficient as the host operating system
- Boot up process is long and takes time

These drawbacks led to the emergence of a new technique called Containerization.

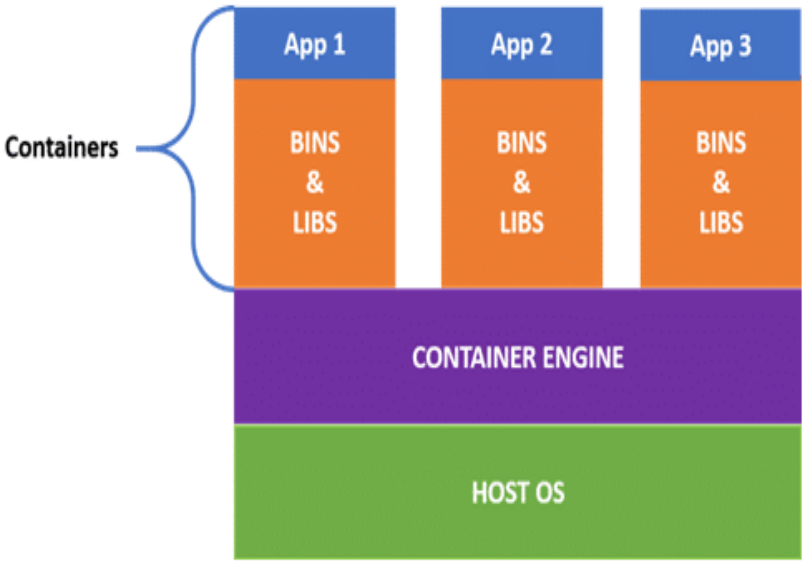
What is Containerization?

Containerization is a technique where the virtualization is brought to an operating system level. In containerization, we virtualize the operating system resources. It is more efficient as there is no guest operating system consuming the host resources; instead, containers utilize only the host operating system and share relevant libraries and resources only when they are required. The required binaries and libraries of containers run on the host kernel leading to faster processing and execution.

In a nutshell, containerization (containers) is a lightweight virtualization technology acting as an alternative to hypervisor virtualization. Any application can be bundled in a container and run without caring about dependencies, libraries, and binaries.

In the case of containerization, all containers share the same host operating system. Multiple containers get created for every type of application making them faster but without wasting the resources, unlike virtualization where a kernel is required for every OS and lots of resources from the host OS are utilized.

Containerization Architecture:



Advantages of Containers:

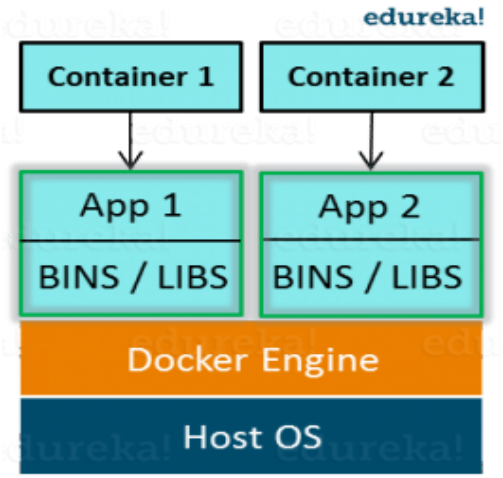
- Containers are small and lightweight as they share the same OS kernel.
- They do not take much time to boot-up (only seconds).
- They exhibit high performance with lower resource utilization

Containerization vs Virtualization:

Containerization	Virtualization
Virtualizes OS	Virtualizes hardware
Installs container only on host OS	Requires complete OS install per VM
Uses only kernel of host OS	Installs a separate kernel per VM
Light-weight	Heavy-weight
Native performance	Limited performance
Process-level isolation	Fully-isolated

Docker:

Docker is a containerization platform that packages your application and all its dependencies together in the form of Containers to ensure that your application works seamlessly in any environment.

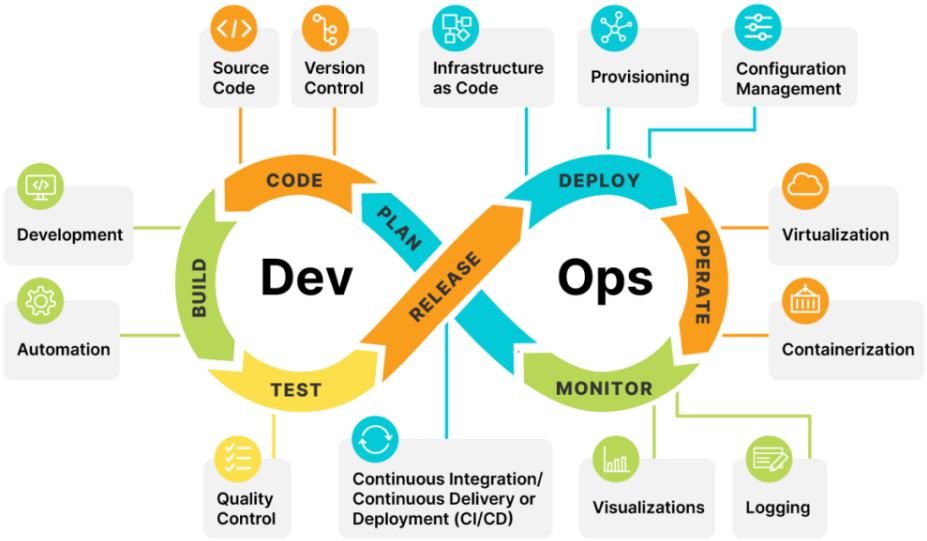


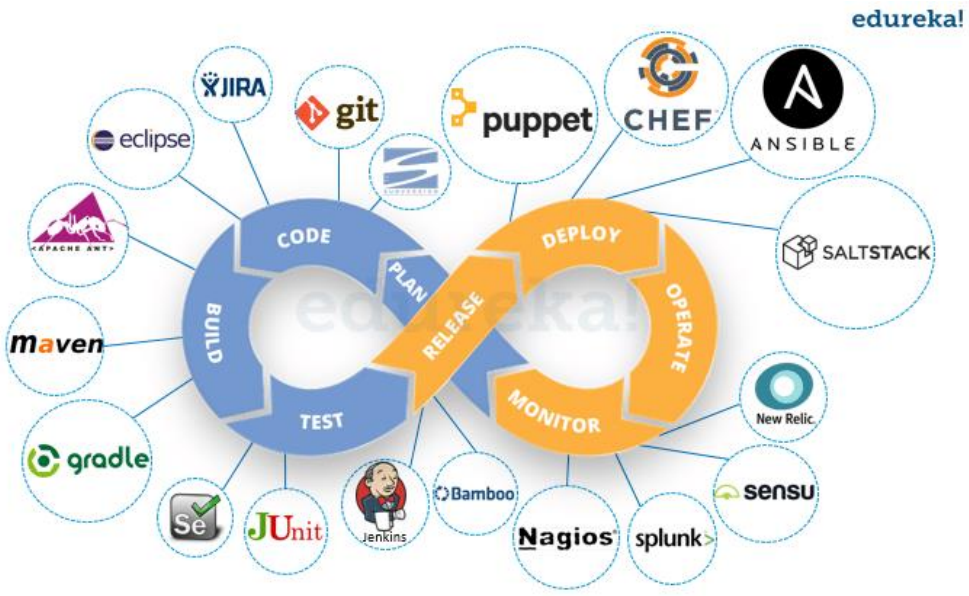
Each application will run on a separate container and will have its own set of libraries and dependencies. This also ensures that there is process level isolation, meaning each application is independent of other applications, giving developers surety that they can build applications that will not interfere with one another.

As a developer, one can build a container which has different applications installed on it and give it to my QA team who will only need to run the container to replicate the developer environment.

Benefits of Docker:

With docker, the QA team need not install all the dependent software and applications to test the code and this helps them save lots of time and energy. This also ensures that the working environment is consistent across all the individuals involved in the process, starting from development to deployment. The number of systems can be scaled up easily and the code can be deployed on them effortlessly.





Docker Architecture:

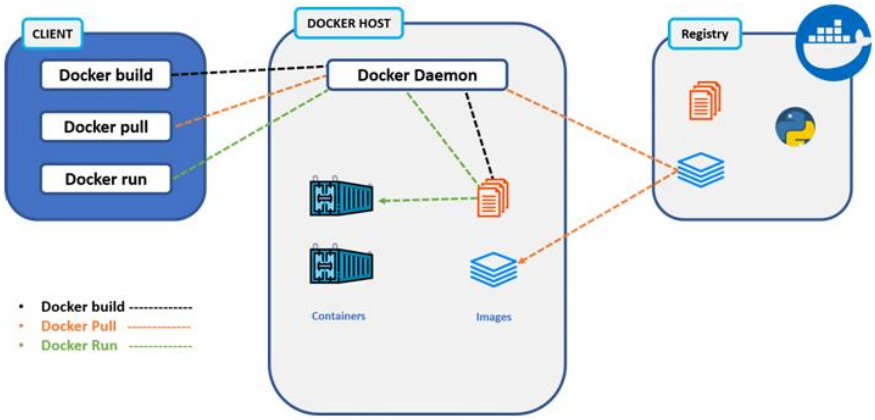
Docker uses a client–server architecture. The Docker client consists of Docker build, Docker pull, and Docker run. The client approaches the Docker daemon that further helps in building, running, and distributing Docker containers. Docker client and Docker daemon can be operated on the same system; otherwise, we can connect the Docker client to the remote Docker daemon. Both communicate with each other using the REST API, over UNIX sockets or a network.

The basic architecture in Docker consists of three parts:

- Docker Client
- Docker Host
- Docker Registry

Docker Client:

- It is the primary way for many Docker users to interact with Docker.
- It uses command-line utility or other tools that use Docker API to communicate with the Docker daemon.
- A Docker client can communicate with more than one Docker daemon.



Docker Host:

In Docker host, we have Docker daemon and Docker objects such as containers and images. First, let's understand the objects on the Docker host, then we will proceed toward the functioning of the Docker daemon.

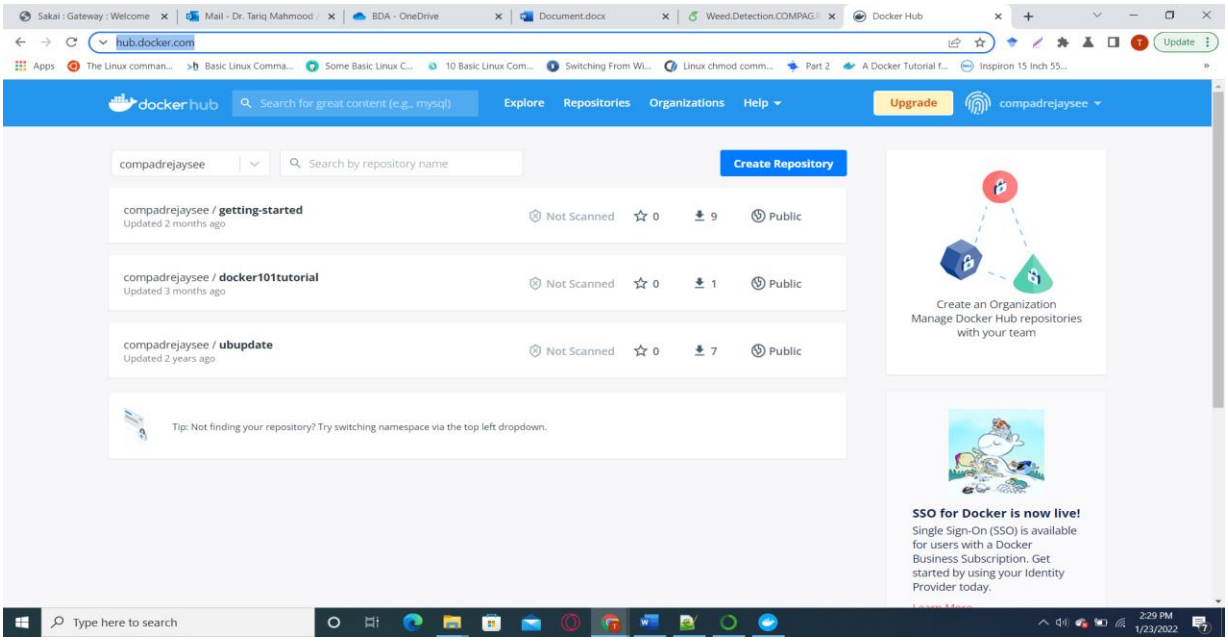
- **Docker Image:** A Docker image is a Docker object. It is a type of recipe/template that can be used for creating Docker containers. It includes steps for creating the necessary software.
- **Docker Container:** A Docker container is also a Docker object. A type of virtual machine created from the instructions found within the Docker image. It is a running instance of a Docker image that consists of the entire package required to run an application.
- **Docker Daemon:**
    - Docker daemon helps in listening requests for the Docker API and in managing Docker objects such as images, containers, volumes, etc. Daemon issues to build an image based on a user's input and then saves it in the registry.
    - In case we don't want to create an image, then we can simply pull an image from the Docker hub (which might be built by some other user). In case we want to create a running instance of our Docker image, then we need to issue a run command that would create a Docker container.
    - A Docker daemon can communicate with other daemons to manage Docker services.

Docker Registry and Docker Hub:

- Docker registry is a repository for Docker images which is used for creating Docker containers.
- We can use a local/private registry or the Docker hub, which is the most popular social example of a Docker repository.

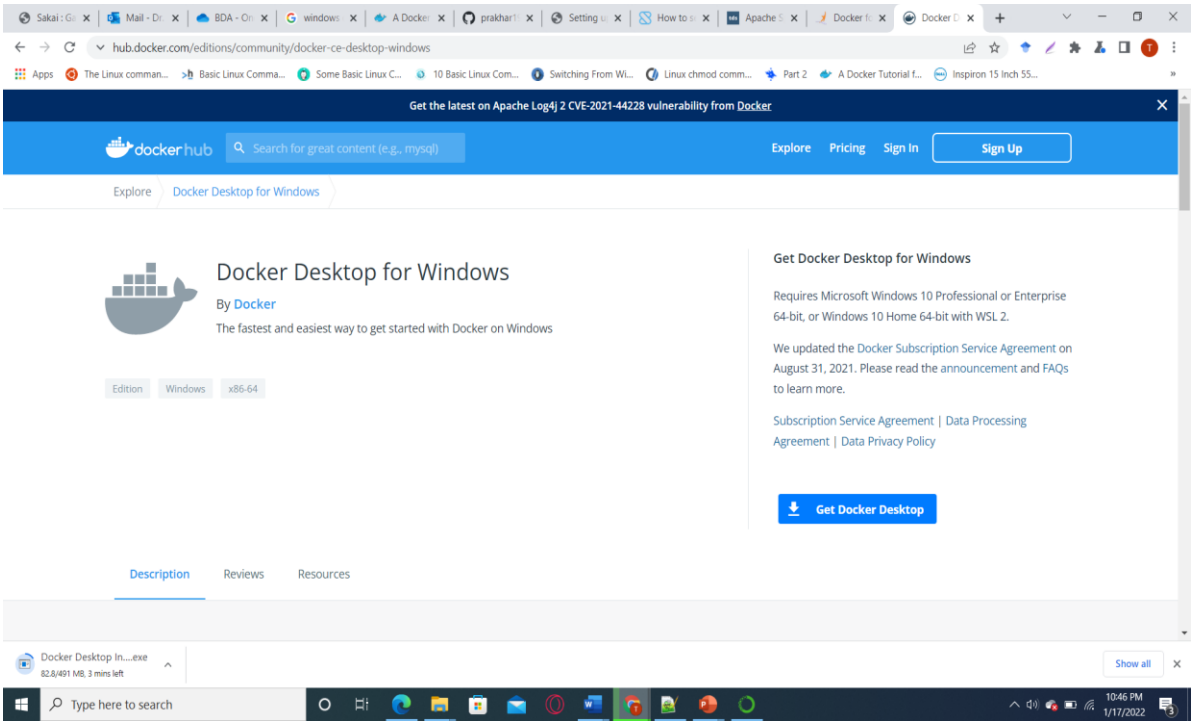
Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

Make an account on Docker hub: https://hub.docker.com/ (mine is compadrejaysee)

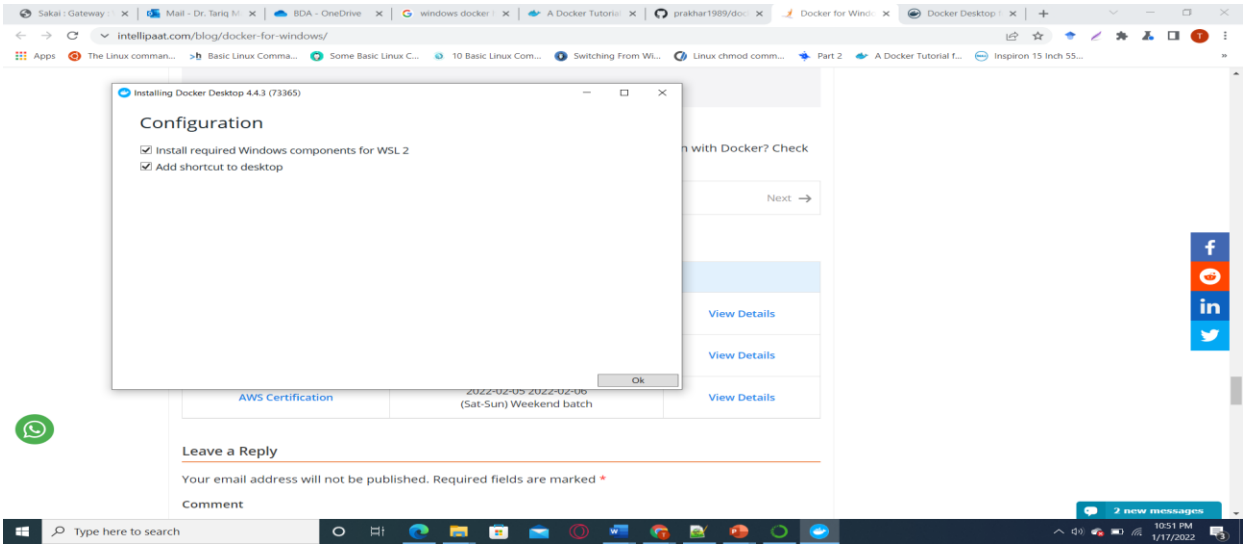


Access: https://hub.docker.com/editions/community/docker-ce-desktop-windows

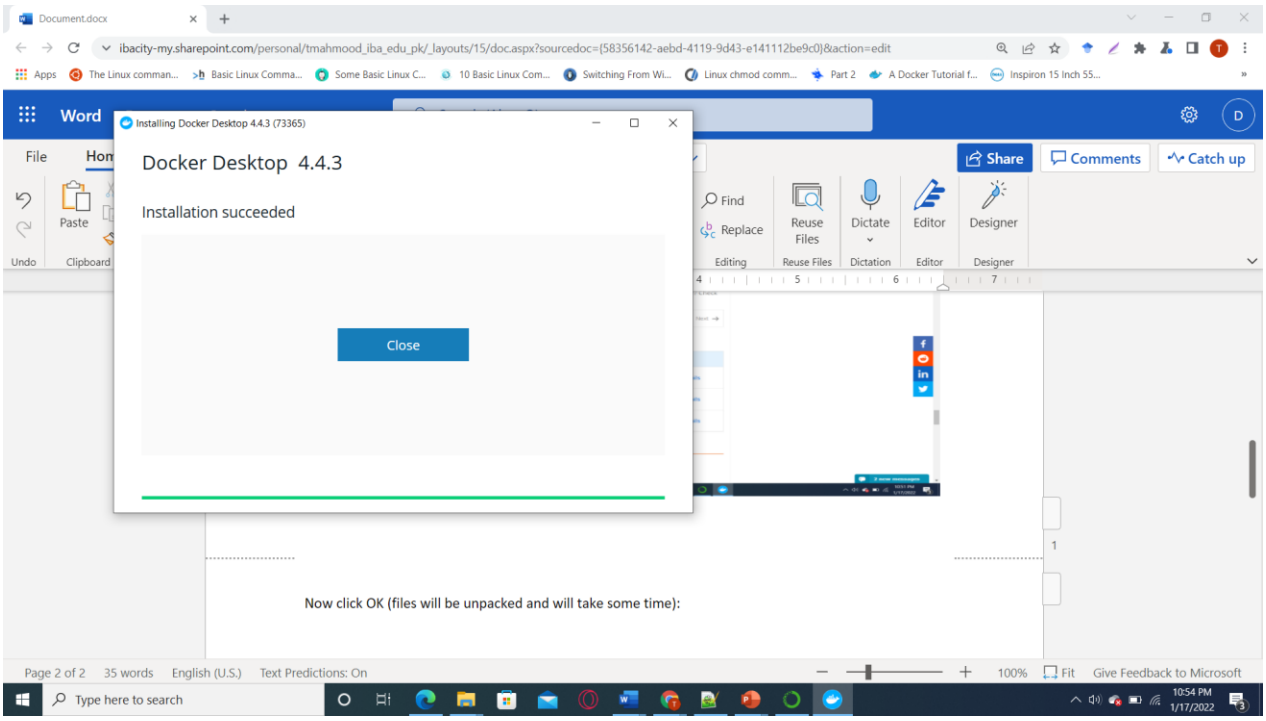
Click: Get Docker Desktop (https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe)



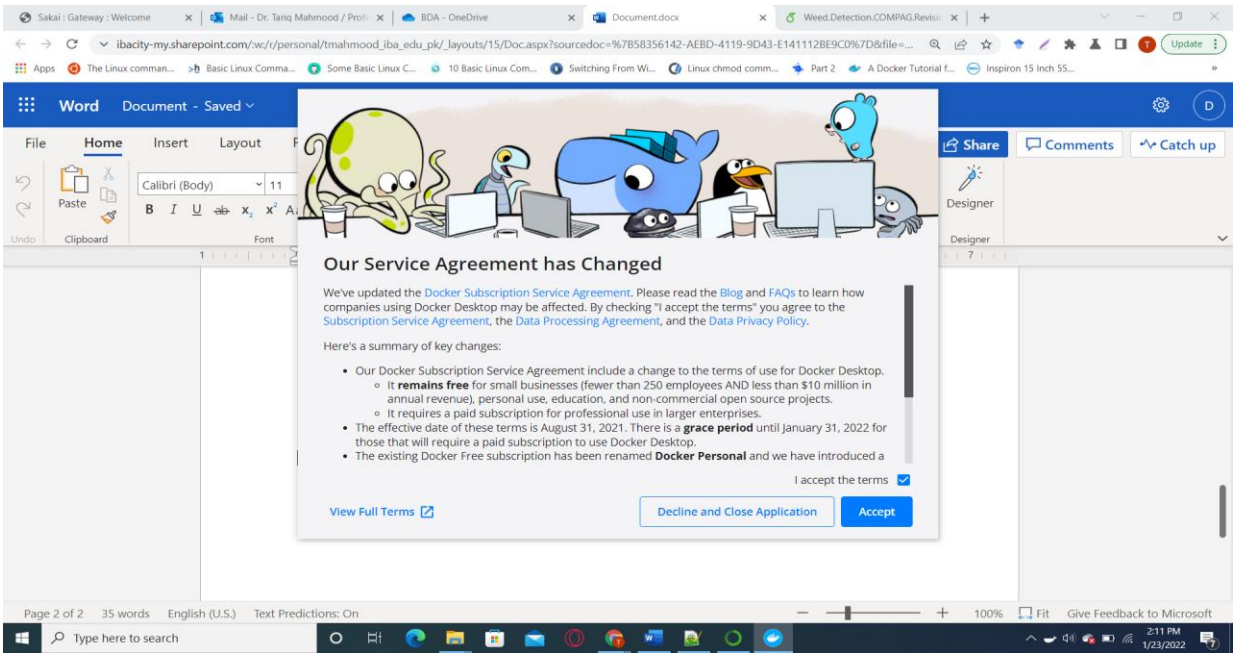
When you click on installer, make sure that “Install required Windows component for WSL2 is checked”



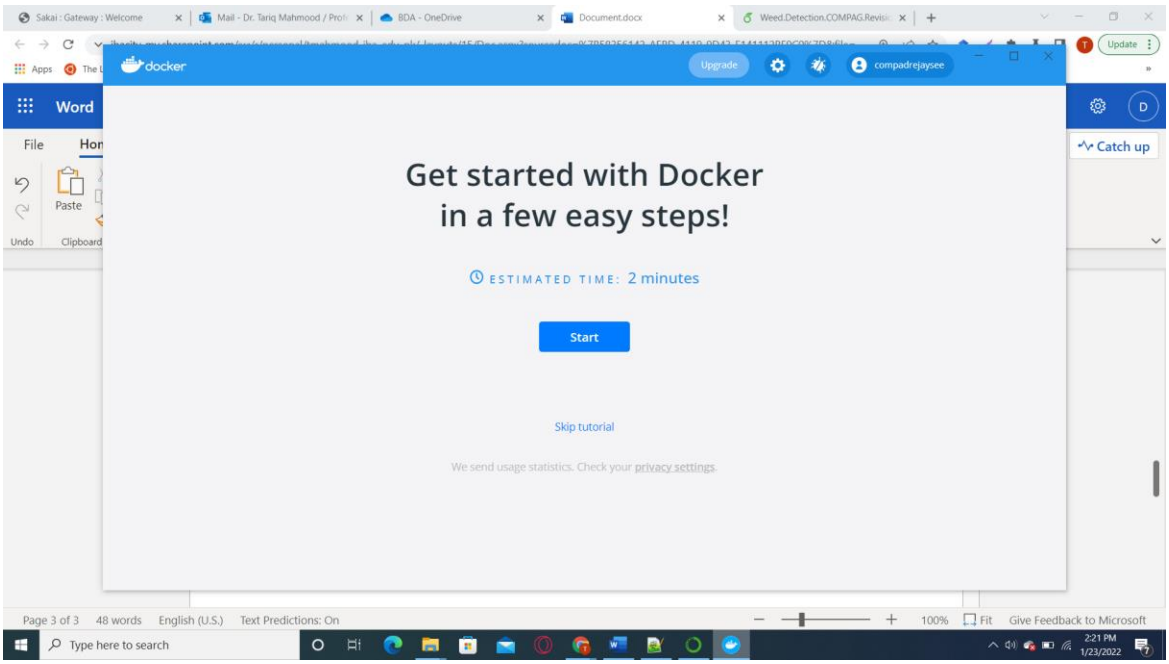
Now click OK (files will be unpacked and will take some time):



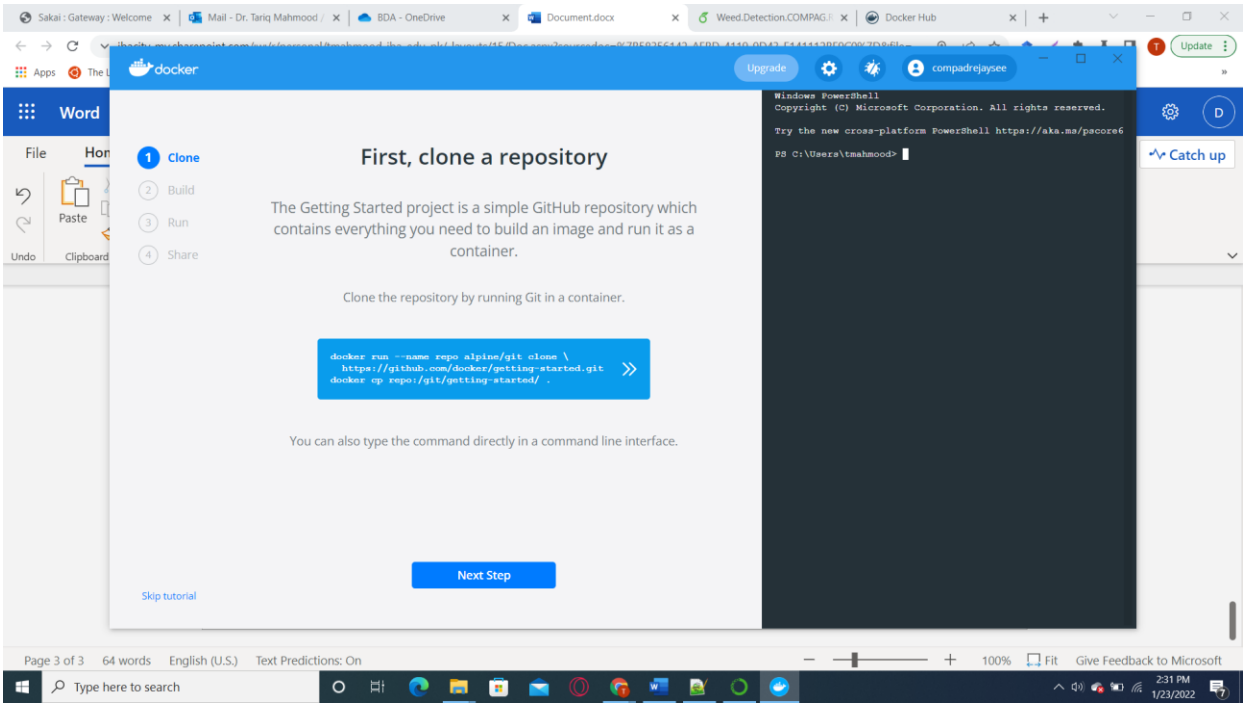
Run Docker Desktop from Windows run bar and Accept



Docker Desktop will open



Start the tutorial and take it:



docker run --name repo alpine/git clone https://github.com/docker/getting-started.git

A Docker image is a private file system just for your container. It provides all the files and code your container needs.

cd getting-started docker build –t docker101tutorial .

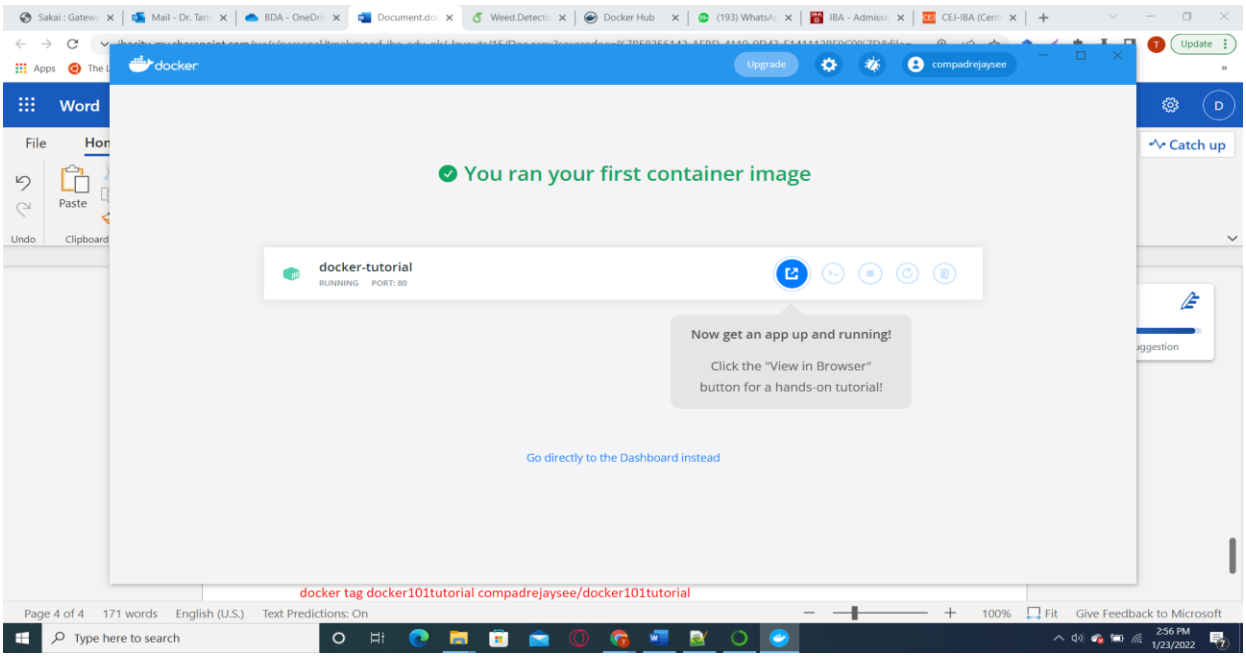
Start a container based on the image you built in the previous step. Running a container launches your application with private resources, securely isolated from the rest of your machine.

docker run -d -p 80:80 \ --name docker-tutorial docker101tutorial

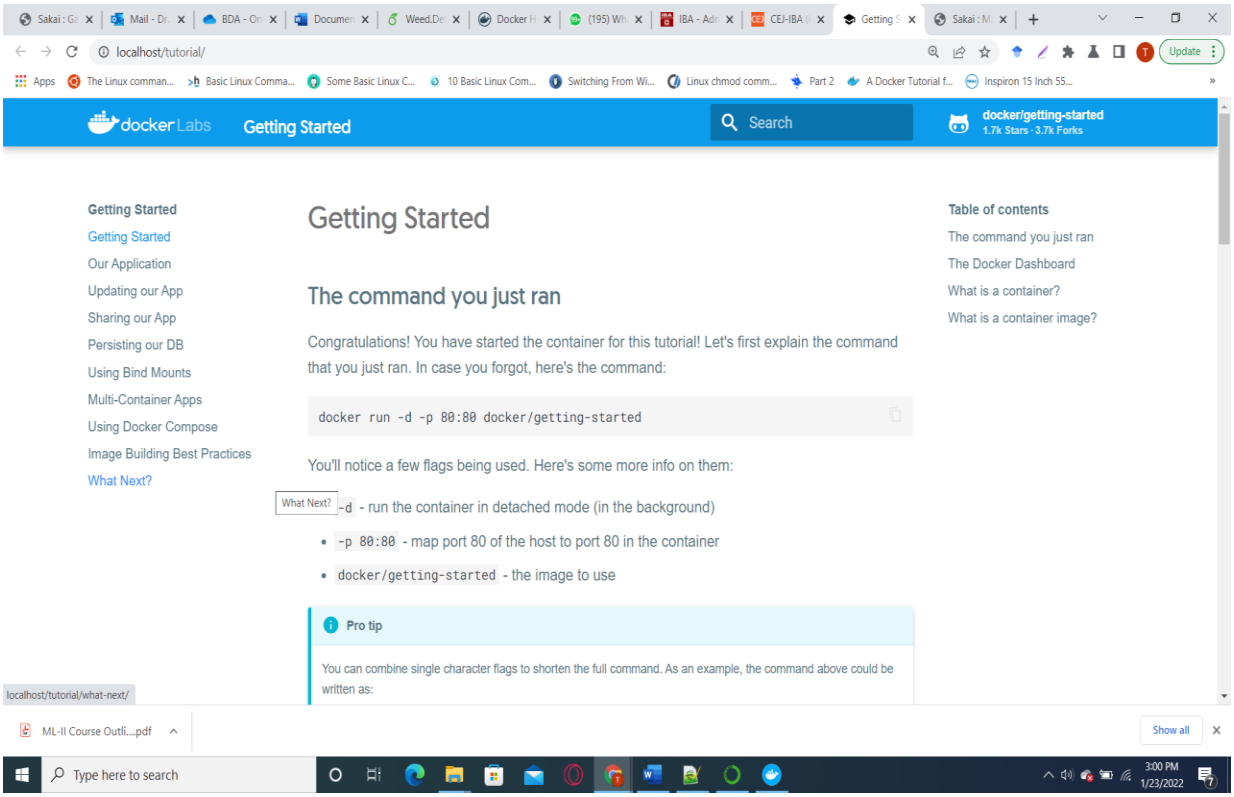
Save and share your image on Docker Hub to enable other users to easily download and run the image on any destination machine.

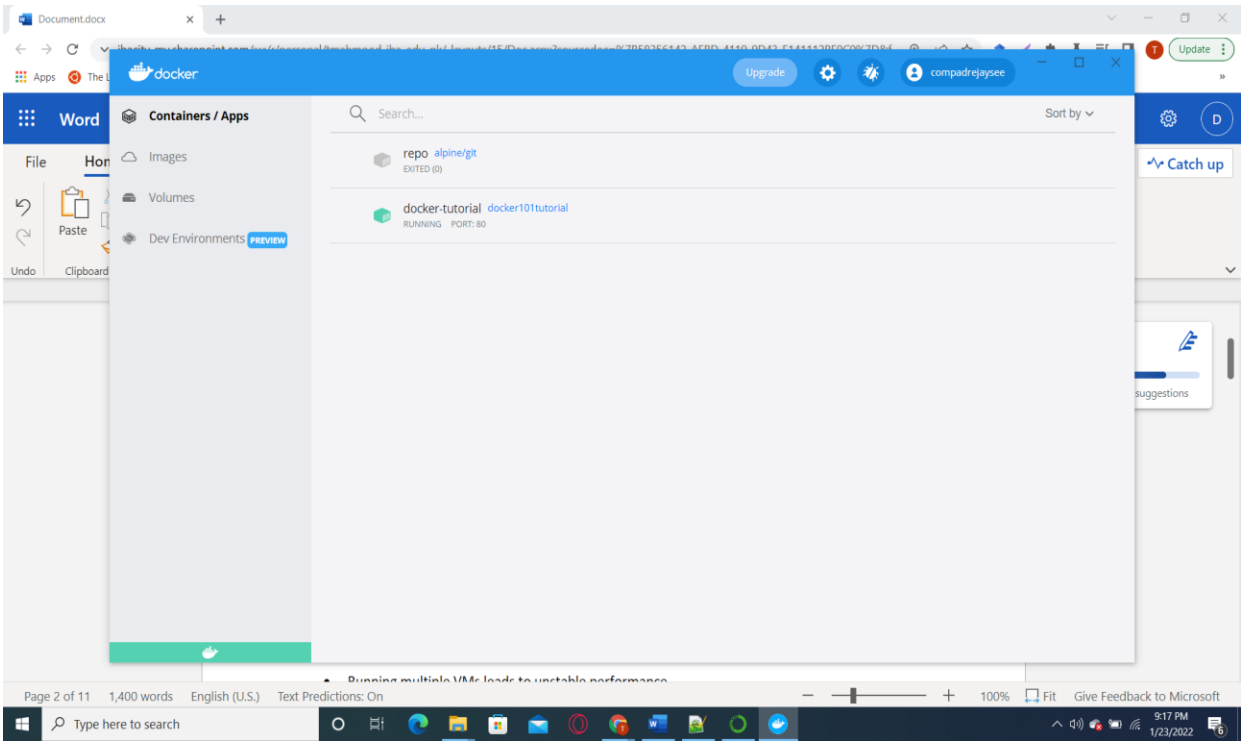
docker tag docker101tutorial compadrejaysee/docker101tutorial

docker push compadrejaysee/docker101tutorial



Click on the blue button. App will deploy on localhost:





docker run -d -p 80:80 docker/getting-started

- -d - run the container in detached mode (in the background)
- -p 80:80 - map port 80 of the host to port 80 in the container
- docker/getting-started - the image to use

Docker Commands:

docker run hello-world

The above will pull the image and execute it and print “Hello from Docker”

docker run docker/whalesay cowsay boo

this will download the image for the first time and then instantiate it through a container

docker run docker/whalesay cowsay tweedledum

Try running the nginx docker container:

docker run nginx

List all running containers (IDs etc.)

docker ps

Every container has an ID (given by docker), its image name, the command used to run it, time it was created, its current status (running or exited), assigned ports (if any by the user) and a name (given by docker)

docker ps -a

Show all prev and current containers

docker stop ID/NAME

I can stop a running docker with either its name or ID. A message will confirm.

docker rm name/id

get rid container permanently:

After stopping, the docker will still remain in the history list….

Remove it completely through the name:

docker rm name

docker images

List all the docker images downloaded from registry

docker rmi

docker rmi nginx

Removes an image but we have to ensure that all of its containers are deleted before:

Let us delete all nginx containers