[Docker - Container Orcas](https://play.fresco.me/course/112)

**Docker Course Introduction**

In this course, you will learn the

* Basic concepts of Containers
* Docker Engine
* Docker architecture and workflow
* Docker end-to-end flow with example

Let us begin the course with the basic concepts of Virtualization (Virtual Machines) and Containerization (Containers) and then dive into the concepts of Docker.

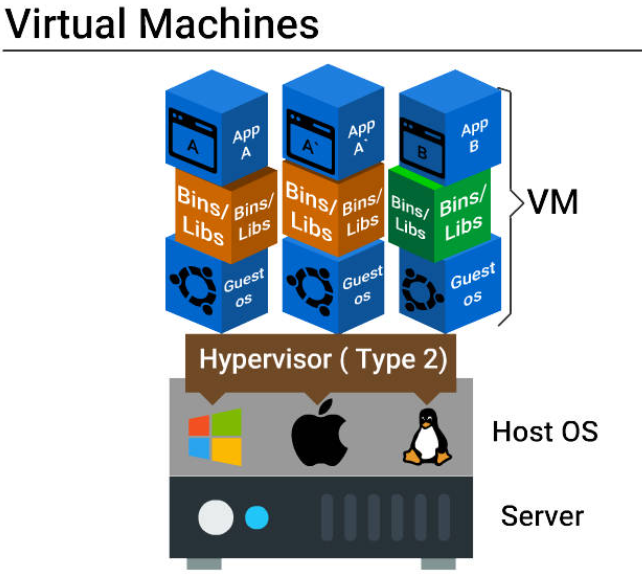
**What is Virtualization?**

Virtualization is a creation of logical object version in place of an actual version.

Few examples are virtual computer hardware platform, virtual storage, and virtual LANs.

Hardware virtualization means creating virtual machine that acts like a real physical computer with an OS.

For example, a virtual machine (VM) hosted on a computer with Microsoft Windows may behave like Ubuntu and Ubuntu supported software may run on the VM.



This block diagram explains virtual machine configuration and how applications are deployed on VMs.

**Need for a Virtual Machine**

Virtual machine setup has the following benefits.

Multiple operating systems can be hosted on the same machine simultaneously with complete isolation.

Multiple VMs can be deployed on the same physical box. This reduces the total number of physical machines.

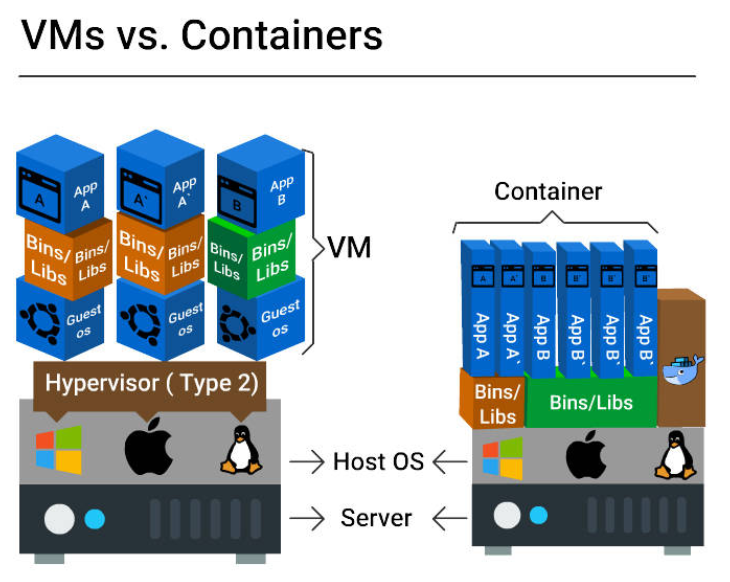
Easy maintenance, app provisioning, and quick recovery.

**Problems with Virtual Machine**

* A lot of wastage of resources like ram, processor, disk space due to fixed space slicing for every application deployed. Hence this is not ideal for a large scale application developed using micro services.
* Inconsistent computing environment across the software delivery life cycle (Prod/Dev/QA).
* Hardware failures like malfunctioning and power supply loss will stop all working servers since many servers run on a single physical server.

**What are Containers?**

* Containers are multiple isolated services that are run on a single control host (underlying infrastructure) and they access a single kernel.
* Container based virtualization is an OS-level virtualization method for deploying and running distributed applications without launching an entire VM for each application.
* They isolate applications from one another.



This block diagram clearly depicts the difference between VMs and Containers on how an application is deployed.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | Here in case of Virtual machine each of software needs to be installed in each VM while in case of container all the container will share same set of software or library to run the container. |

**Size:**

VMs - Resource wastage is high

Containers - No wastage

**Start up**:

VMs - Starts slow

Containers - Starts really quick

**Integration with DevOps tools:**

VMs - Complex (infrastructure wastage, repetitive configuration and minimal scalability)

Containers - Very simple

**Container - Benefits**

* Improved portability
* Better performance
* Optimum RAM/disk space/cloud utilization
* Suited for agile environment
* Facilitates approaches such as micro services, continuous integration, and delivery.
* Eliminates environment inconsistencies

**Benefits from Container Orchestration Tools**

Container Orchestration tools like Docker Swarm, Amazon ECS, and Azure Container Service:

* Facilitate auto deployment
* Scale application easily
* Quickly push application from one environment to another
* Enable automated rollbacks and backups
* Support load balancing and service healing

**What is virtualization?**



Creation of an actual version of something rather than a logical object.



No hardware at all.



Just another word for virtual storage.



**Creation of a logical object version of something rather than an actual version.**

------------------------

**Which of the following is the criteria to be fulfilled by an ideal VMM?**



VM should perform exactly the same way as it would perform on the physical machine.



VMM must have total control of the system resources.



**All the options**



The environment created for the VM should be the same as the original physical machine.

----------------------------------------------------------------------------------------------------------------------------------

**The following are examples of Virtualization, except \_\_\_\_\_\_\_**



Memory virtualization



**None of the options**



Data virtualization



Desktop virtualization

-----------------------------------------------------------------------------------------------------------------------------------

**What is a Hypervisor?**



**Software running the VM**



None of the options



The VM itself



Collection of VMs

-----------------------------------------------------------------------------------------------------------------------------------

**Another name for Hypervisor is\_\_\_\_\_\_\_\_\_\_\_.**



Virtual Machine



**Virtual Machine Monitor**



Images



Containers

**Which of the following is the criteria to be fulfilled by an ideal VMM?**



The environment created for the VM should be the same as the original physical machine.



VMM must have total control of the system resources.



**All the options**



VM should perform exactly the same way as it would perform on the physical machine.

Docker - Introduction

Docker is a tool intended to make the process of creating, deploying and running applications easier by using container based virtualization technology.

Docker is an open source container technology that provisions far more apps running on the same old servers compared to traditional VMs.

Docker Engine

Docker engine is the Docker core component that is responsible for creating Docker Images and running them as services.

Let us learn in detail about Docker Images in the next topic.

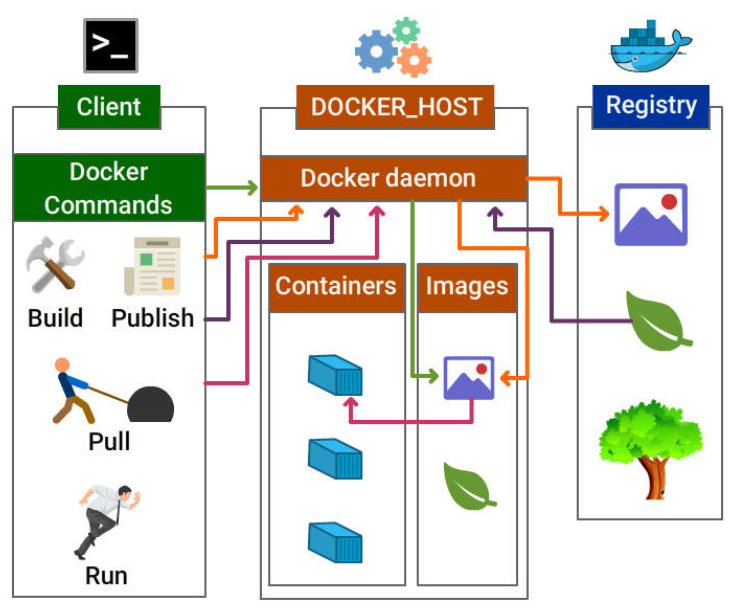
##### Docker Core Components

|  |  |
| --- | --- |
|  | Docker Engine Core Components:  **Docker Daemon**   * Continuous running program (daemon process) that manages the service and other docker objects tied to it.   **REST API**   * Specifies interfaces that programs can utilize to speak to the daemon and direct it what to do.   **Docker Client**   * CLI is utilized to interact with the daemon (docker command). |

##### Docker on Linux and non-Linux Kernel

|  |  |
| --- | --- |
|  | **Docker hosted on Linux:**  This requires just a Docker client and Docker daemon.  **Docker hosted on non-Linux:**   * Docker desktop for mac uses [HyperKit](https://github.com/docker/HyperKit/" \t "_blank) VM, which handles virtualization. * Docker desktop for windows uses **Microsoft Hyper-V** to manage virtualization. |

##### Docker Architecture



**Docker Architecture Block Diagram**

Let us discuss in detail on the Docker components and workflow process in the upcoming cards.

##### Docker Components

Docker components include

* Docker daemon
* Docker client
* Docker Objects
* Images
* Containers / Services
* Network
* Volumes
* Docker Registry

##### Docker Daemon and Docker Client

**Docker Daemon**

Docker daemon is the docker process that receives requests from docker client and is responsible for managing docker objects such as containers, images, networks, and volumes.

**Docker Client**

Docker client communicates with the docker daemon through Rest API calls. A docker client can send a request to many docker daemons.

##### Docker Image

Docker image is the collection of all files, libraries, binaries and other dependencies forming an executable software application, which can run everywhere without glitches.

Note: An image is an inert, immutable file.

**Here are the points to be noted about Docker Image:**

* Docker image is read-only, i.e., the image and its content cannot be altered.
* Although the alteration is not allowed in Docker, we are allowed to add the new layer with the changes.
* After having many alterations, a docker image may be visualized as several layers one above another.

**Parent and Child Images**

The layering concept in docker images leads to the addition of required capabilities efficiently by adding a new layer to the existing one resulting in a new image.

Hence, the image has the parent-child relationship where the original image is termed as the **base image** upon which several child images are added.

##### Docker Service / Container

Containers **are run-time instances** of Docker images that can be run using the Docker run command.

The fundamental purpose of Docker is to run containers.

You can run a docker image to create as many docker containers as you want.

**Docker Container Lifecycle**

This example explains the container life cycle stages and traversal from one stage to another.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

##### Docker Network

The concept of networking in Docker comes into account when working with Docker in a real time scenario at a large scale.

Docker Networking helps us to share data across various containers.

Host and containers in Docker are tied with **1:N relationship**, which means one host can command multiple containers.

**Modes of Networking**

Various modes for networking are all about how we manage connections between containers.

* **Bridge mode Networking**
* **Host Mode Networking**
* **Container Mode Networking**
* **No Networking**

##### Docker File

**Dockerfile is a script, formed of different arguments and commands (instructions) listed successively to automatically execute actions on a base image to form or create a new one.**

**A Docker File is a simple text file with instructions on how to build your images.**

##### Docker Registry

Docker Registry (Docker Repository) is **a storage house** for the Docker Images. It can be accessed publicly or privately by developers across the world.

* Docker images can be **sent to registry** by using docker push subcommand.
* Docker images can be **downloaded** from the registry using docker pull subcommand.

**Following are the places where Docker registry can be hosted:**

1. **Docker Hub**
2. **AWS Container Registry**
3. **Google Container Registry** and lot more

##### Docker Hub

**Docker hub is one of the repositories of images which can be accessed at index.docker.io.**

**Docker Hub is the official repository by Docker development community. Any third party images can be pulled from the repository.**

**e.g.:**

docker **pull** **thedockerbook/helloworld**

##### Docker Storage

**Storage drivers**

Container layer contains a **very thin writable layer**, unlike images that have read only layers.

Each container has its storage layer whereas they **share the read only image layer** across containers in the same host.

Docker uses **storage drivers** that will manage the data using copy-on-write mechanism.

**Copy-on-write mechanism**

Docker engine **does not copy the whole image** when we try to launch it. Instead, it uses Copy-on-write mechanism by which it uses a single copy of shared data until the data within the image is modified.

This saves a lot of disk volume, and the startup process is quick.

**Volumes**

Volume is a **directory mounted** in the container that is created using docker command.

They are used to share data between containers by using the same volume across various containers.

Docker Workflow

|  |  |
| --- | --- |
| Client build the docker and create the image | Now publish the image into repository |
| Now pull the image from the respository | Finally put the image into container to run it |

**Docker workflow includes the following components:**

Docker Image - Read only template that stores the application and environment.

Docker Container - Runtime instance of a docker image

Docker registry - Public and private repositories to store images

Docker File - Automates Image construction

Docker file Compose - Compose is a tool that can be used to manage multiple containers containing different applications.

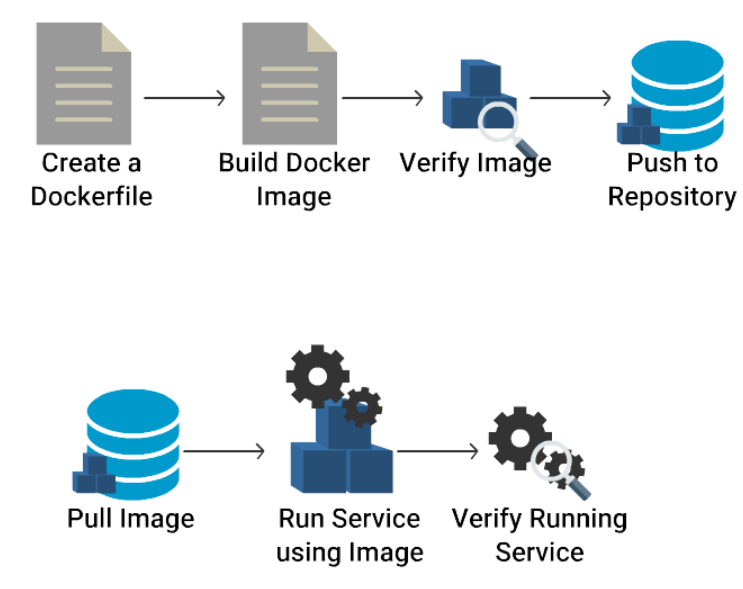
**Docker Workflow - In Detail**

**Part 1**

* Create a docker file that includes details on the base and child images to be built.
* Build the docker file using docker build command and tag a name to the image.
* Verify if the image is built successfully using docker images command and Run docker inspect command to view the complete details of the image.
* Now the image is ready, push the same to the image repository using docker push command.

**Part 2**

* Pull the newly created image from the repository using docker pull command.
* Run the image using docker run command or using Dockerfile compose.
* Now you can verify the running container using docker ps command.



Docker images can be run in \_\_\_\_\_\_\_\_\_.



Physical Machines



Virtual Machines



**All the options**



Clouds

The following statements are true about Docker, except \_\_\_\_\_\_\_\_



It reduces the size of the development.



It lessens the extra work to be done by the developer.



**Containers are non-scalable.**



Running containers can share OS kernel.

------------------------------------------------------------------------------------------------------------------------

Docker Container \_\_\_\_\_\_\_\_\_



Enables quick start up



Eliminates repetitive environment setup on different hosts



Uses Copy-on-write mechanism



**All the options**

-------------------------------------------------------------------------------------------------------------------------

A machine cannot run multiple containers independently.



True



**False**

-------------------------------------------------------------------------------------------------------------------------

You can add new features to an existing image by \_\_\_\_\_\_\_\_\_\_\_



Deleting unnecessary layers and adding modified layers.



**Adding the new layer with the changes.**



Changing the existing layer.



Sharing the layers with other containers.

**Practise Docker Commands**

Now that you have understand about docker architecture and its components, it is time to practice some of the docker commands.

For this purpose, you can have docker installed on your local machine.

Docker Basic Commands

Here are few basic docker commands.

Check the version of Docker:

* docker version

Check the detailed information on the running/stopped containers:

* docker info

Docker images can be downloaded from Docker hub using docker commands.

Lets pull an image from docker hub using pull command.

Download a image from docker hub

* docker pull <<image name>>

e.g. docker pull nginx

Docker Commands - Images

Verify the downloaded docker images:

* docker images ↵

View all the commands that were run with an image via a container.

* docker history <<Image Name>> ↵

e.g. docker history nginx

Remove Docker Images

* docker rmi <<Image Name>> ↵

e.g. docker rmi nginx

Download and run an image in docker container using run command

* docker run <<Image Name>> ↵

e.g. docker run --name nginxservice -d nginx

--name --> to specify a name for the running service. In this example, it is nginxservice

-d --> to run the service in the background

Docker Version Tag

You must have noticed that on pulling the image from the Registry, the tagged version of the image is displayed:

For example, **docker pull busybox:1.24** will download the corresponding version:

Docker Search

We can also search for the images in the Docker Hub registry by using docker search subcommand. Let us search for ubuntu images in the Docker Hub, and limit the search result only to 20 because we have more than 2000 images on Ubuntu: **docker search ubuntu | head -20**

Docker Commands - Container

Now let’s run list of commands on the service/ container

List running containers

* **docker ps**

---------------------------------------------------------------------------------------------------------------------

Know the IP address of the running container:

* **docker inspect <Container Name>**

**e.g. docker inspect nginxservice**

---------------------------------------------------------------------------------------------------------------------

Print the stats for a running Container

* **docker stats <<Container Name>> ↵**

**E.g. docker stats nginxservice**

---------------------------------------------------------------------------------------------------------------------

Pause the processes in a running container

* **docker pause <<Container Name >> ↵**

**E.g. docker pause nginxservice**

---------------------------------------------------------------------------------------------------------------------

Unpause the processes in a running container

* **docker unpause <<Container Name >> ↵**

**e.g. docker unpause nginxservice**  
---------------------------------------------------------------------------------------------------------------------

Kill the processes in a running container

* **docker kill <<Container Name >> ↵**

**e.g. docker kill nginxservice**

---------------------------------------------------------------------------------------------------------------------

Start the same container:

* **docker start <<Container Name>> ↵**

**e.g. docker start nginxservice**

---------------------------------------------------------------------------------------------------------------------

Stop the running container

* **docker stop <<Container Name >> ↵**

**e.g. docker stop nginxservice**

---------------------------------------------------------------------------------------------------------------------

List all containers (This includes containers in a all states):

We will be able to see the container we just stopped listed here.

* **docker ps -a**

---------------------------------------------------------------------------------------------------------------------

Delete a container:

* **docker rm <<Container Name >> ↵**

**e.g. docker rm nginxservice**

---------------------------------------------------------------------------------------------------------------------

To remove all stopped containers:

* **docker container prune**

Note: Instead of using the Container Name, all the above commands can be executed with the container id as well.

---------------------------------------------------------------------------------------------------------------------

Export a container

**docker export <<Container Name>> <<file\_Name>>.tar ↵**

**e.g.**

**Lets run a service using docker run command.**

**docker run --name newnginxservice -d nginx**

**docker export newnginxservice > test.tar**

---------------------------------------------------------------------------------------------------------------------

Import a container

**docker import <<Remote URL/Image Name.tar>> ↵**

**e.g. docker import test.tar**

---------------------------------------------------------------------------------------------------------------------

Docker daemon Commands

Stop Docker daemon process

* service docker stop

Start Docker daemon process

* service docker start

You may not be able to try these 2 commands since you would not have access to root on Katacoda playground.

Diagnose Run Issues

In case you are having a problem with downloading the images and running them, please follow these steps to check whether the docker service is running on your system or not:

Check the running status of docker:

* service docker status

Restart Docker service in your system:

* service docker restart

-------------------------------------------------------

Docker registry is a \_\_\_\_\_\_\_\_\_\_\_.



Storage house for the Docker images



Storage house for the Docker containers



None of the options



Storage house for the Docker softwares

-----------------------------------------------------------

Which sub-command is used to create a new image?



**docker commit**



docker create

Which sub-command is used to download images from a registry?



**docker pull**



docker extract



docker download



docker push

Which sub-command helps you to verify the downloaded images?



docker rm



docker prune



**docker images**



docker image

An image from the host machine can be deleted using \_\_\_\_\_\_\_\_\_\_\_.



docker -rmi imageName



**docker rm imageName**



docker rmi imageName



docker -rm imageName

The Docker command to run an image is \_\_\_\_\_\_\_\_\_\_\_.



docker push



**docker run**



docker pull



docker save

You can check the history of creation of an image using \_\_\_\_\_\_\_\_\_.



docker imageName show



docker imageName history



**docker history imageName**



docker imageName show-history

The status of the docker running service can be checked using \_\_\_\_\_\_\_\_\_\_.



check docker status



service docker



**service docker status**



docker status

Docker Installation

Let Us Install it!

Looks like you are excited to dive into the Docker World! Aren't you?

Let us quickly explain you the procedure to install Docker in your system and get a quick hands-on with the basics.

Docker Installation

Prerequisite for installation of Docker on Linux are:

* 64-bit architecture Linux
* Linux kernel must be 3.10 or later

**Here are the steps for installing the community edition in Ubuntu 16.04:**

Add the GPG key for the official Docker repository to the system:

* $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Now add the Docker repository to APT sources:

* $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

1. Now update the package database with the Docker packages:
   * + $ sudo apt-get update
2. Check for the policy:
   * + $ apt-cache policy docker-ce
3. Install Docker:
   * + $ sudo apt-get install -y docker-ce
4. Check the installed version of Docker:
   * + $ sudo docker --version

**Congratulations!** You have successfully installed Docker version 17.03.0 community edition.

Installation Using Automated Script

Are you feeling this procedure lengthy? There is a shortcut to this process.

Just run the below command to install Docker

* curl command
* $ sudo curl -sSL https://get.docker.io/ | sh

Uninstall Docker CE

This command is used to uninstall Docker CE package in Ubuntu machine.

* $ sudo apt-get purge docker-ce

**Docker Example Step by step**

Hands-on scenario

***Welcome to the Docker challenge, your task is to follow the below steps and complete them successfully.***

\*Perform the following actions described below by executing the respective commands. Open terminal and execute the commands. \*

1. Check the version of Docker.
2. Check the detailed information on the running/stopped containers.
3. Download tomcat:latest image from docker hub.
4. Verify the downloaded docker images.
5. View all the commands that were run on the tomcat:latest image (check docker image history).
6. Remove tomcat:latest docker image.
7. Download and run an nginx:latest image in the docker container using the run command, name the container nginxservice.
8. Pull busybox:1.24 docker image.
9. Search for ubuntu images in the Docker hub, and limit the search result to 20.
10. List all containers.
11. Identify the IP address of the running container 'nginxservice'.
12. Print the stats for a running container 'nginxservice'.
13. Pause the processes in a running container 'nginxservice'.
14. Start the processes in a running container 'nginxservice'.
15. Kill the processes in a running container 'nginxservice'.
16. Start the same container 'nginxservice'.
17. Stop the running container 'nginxservice'.
18. List all containers (including the containers in all states).
19. Export the container 'nginxservice' and name the tar as 'test.tar'.
20. Delete the container 'nginxservice'.
21. Remove all stopped containers by pruning them.
22. Import the container 'test.tar'

* Create an nginx Dockerfile with index page that returns the string 'Welcome to fresco'
* Build the image and name it 'nginximage', run the image and expose it on port 80 and name it 'nginxcontainer'
* Start a container registry image using following command

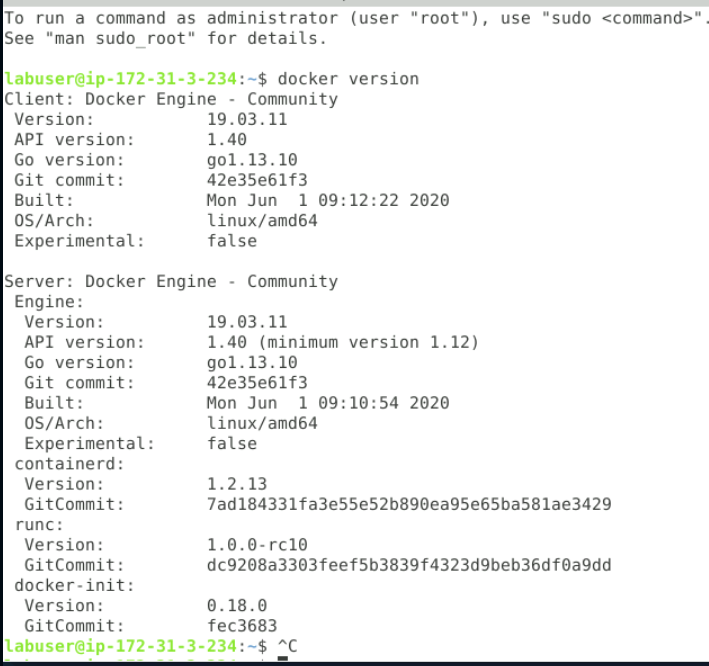
docker run -d -p 5000:5000 --restart=always --name registry registry:2

* Write the following content to /etc/docker/daemon.json file to avoid http security error. Replace the string with registry container ip.

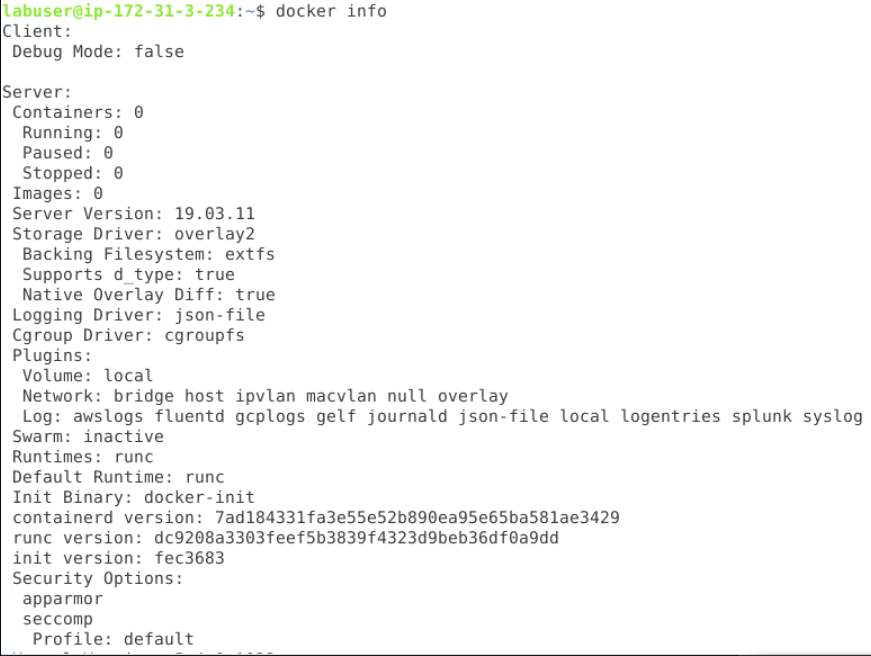
|  |
| --- |
| {  "insecure-registries" : ["replace-with-registry-ip:5000"]  } |

* Restart docker service using the command 'sudo service docker restart'
* Push the nginximage image to registry. (Hint: need to tag the docker image with the registry ip and port)

1. Check the version of Docker.

****

**2. Check the detailed information on the running/stopped containers.**

****

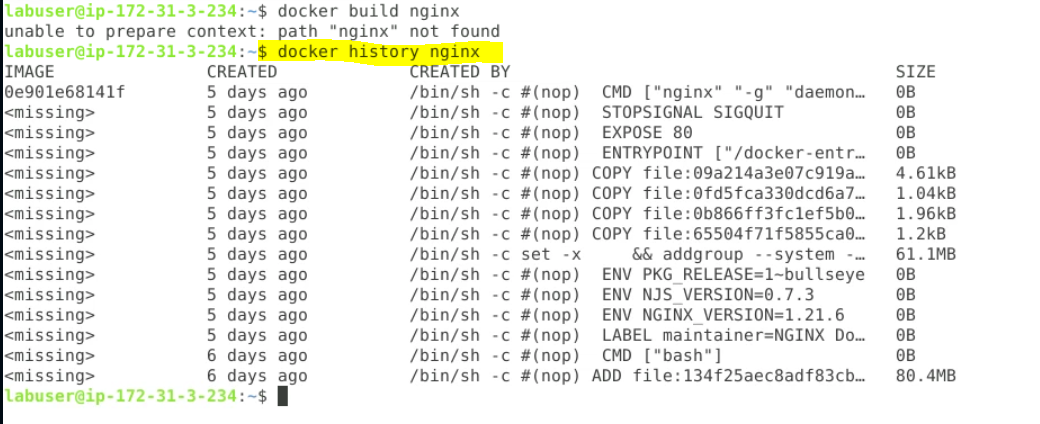
**3. Download tomcat:latest image from docker hub.**

****

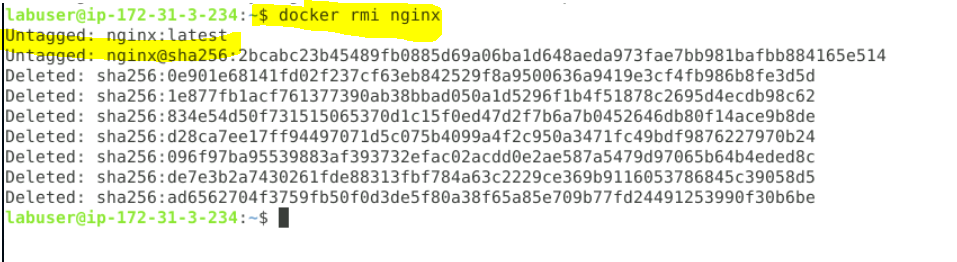
1. **Verify the downloaded docker images.**

****

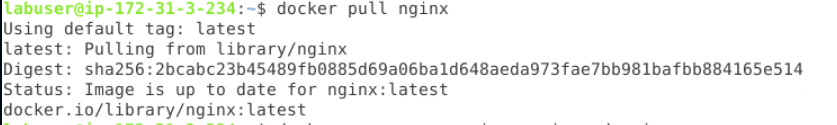
1. **View all the commands that were run on the tomcat:latest image (check docker image history).**

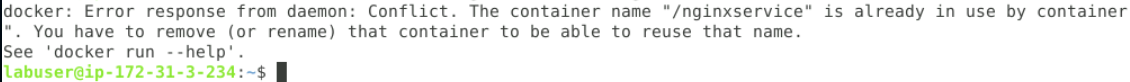
****

1. **Remove tomcat:latest docker image.**

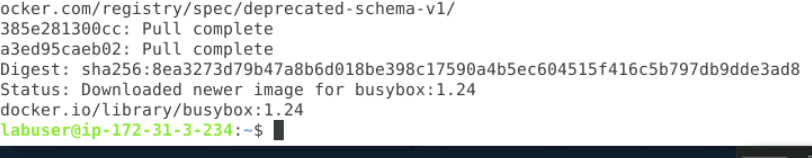
****

1. **Download and run an nginx:latest image in the docker container using the run command, name the container nginxservice.**

** **

** **

1. **Pull busybox:1.24 docker image.**

**  **

Docker Container Orcas Summary

**Ocean in the Capsule!**

**In this course, you studied the following topics:**

* Docker Architecture
* About Dockerfiles, Containers, and Images
* Running and playing with containers
* Creating and pushing Docker images
* Export and Import of Images

##### Further Reading

Are you willing to explore the Docker world further? Here is the list of external links as a reference which will help you a lot:

[Docker Documentation](https://docs.docker.com/) <https://docs.docker.com/>

Docker Daemon does not require root privileges to run.



True



False

You can search for Docker images by using \_\_\_\_\_\_\_\_\_\_.



docker find imageName



docker pull image



docker search imageName



docker searchfor imageName

Which of the following statements is true about containerization?



All the options



Response time of the VM boosts.



Docker containers do not have a complete OS with them.



Container consumes very less memory.

Docker hub images can be accessed at \_\_\_\_\_\_\_\_\_\_\_.



index.docker.io



index.docker



None of the options



docker

Container can alter host filesystem without any restriction.



True



False

Docker images can be sent to registry by using \_\_\_\_\_\_\_\_\_\_\_\_.



docker push



docker pull



docker publish



docker send

An altered Docker image is visualized as \_\_\_\_\_\_\_\_\_\_\_



The final layer that undergoes required changes.



Both the options



Several layers one above another.



None of the options

Containers have shared resources and are lightweight.



False



True

What are all Docker registries?

**Docker Registries**

* Docker Hub.
* Azure Container Registry.
* Google Container Registry.
* Google Artifact Registry.
* Amazon EC2 Container Registry.
* Bintray.io/Artifactory.
* Quay.io.
* Github Container Registry.
* The following are all Docker Registries except \_\_\_\_\_\_\_\_\_\_
* 
* AWS Container Registry
* 
* Google Container Registry
* 
* None of the options
* 
* Docker Hub

Docker Networking is needed while working on Docker \_\_\_\_\_\_\_\_\_\_\_.



in real-time on a large scale



on a small scale

Running Docker containers can be checked by using \_\_\_\_\_\_\_\_\_\_\_.



docker ps



docker cs

To get help on a topic in Docker, you can use \_\_\_\_\_\_\_\_\_\_.



docker --help topicName



docker topicName --help

Application running in a VM is hidden from the host OS with the help of Hypervisor/VMM.



True



False

Which of the following is the correct method of exporting a TAR file?



docker exampleImage.zip > exampleImage.tar



docker exampleImage.tar > exampleImage



docker exampleImage > exampleImage.tar



docker export exampleImage > exampleImage.tar

What is the name of the virtual Ethernet bridge that is created at the start of Docker Daemon?



docker1



docker0



docker2



docker00

The Docker deamon creates “**docker0**” a virtual ethernet bridge that forwards packets between all interfaces attached to it.

The following are components of Docker Engine, except \_\_\_\_\_\_\_



None of the options



Docker Daemon



Command Line Interface



REST API

