Internship 2019 - Cybozu Viet Nam

**RESEARCH DOCKER**

dòng ngang

# See the source image

**Team 3:**  Phoenix

**Mentor:** Vu Duc Huy

**Members:** Truong Thanh Danh

Ho Ngoc Dinh

Nguyen Quoc Huy

Vu Pham Thanh Huy

# 

# Contents:

1. **Introduction.**
2. **Concept of docker.**
3. What is Docker?
4. Advantage and disadvantage.
5. **Docker Components.**
6. Docker engine.
7. Docker Daemon (server).
8. Docker CLI (client).
9. REST API.
10. Images.
11. Containers.
12. Data volumes.
13. Networks.
14. Registries.
15. **Distribution tools**
16. Docker hub.
17. Use official Repositories.
18. Docker command.
19. Docker compose.
20. Registry and other public retristris.
21. **Docker Architecture.**
22. Description.
23. Docker daemon.
24. Docker Client .
25. Docker registry.
26. Docker objects.
27. Distinguish between Docker and Virtual Machine.
28. **Cheat sheet.**
29. **Install docker.**
30. On window.
31. On macOS.
32. **Demo for web.**

**I. Introduction:**

**II. Concept of Docker:**

1. **What is Docker?**

**Docker** is the platform that provides tools and services for developers and admin systems to develop, execute and run applications with containers. It provides ways to build, deploy and run application easily on the virtualization platform - “ Build once, run anywhere”

1. **Advantage and Disadvantage:**

* **Advantage:**

**+ Docker products an API for container management:** in an image format and a chance to use a remote registry for sharing containers.

**+ Fast application deployment:** containers carry the minimal runtime requirements of the application, decreasing their size and enabling them to be deployed instantly.

**+ Transferables across machines:** an application and all its dependencies can be grouped into a separate container that is autonomous from the host version of Linux kernel, platform configuration or deployment type.

**+ Version control and component retain:** pursue succeeding versions of a contain, inspect irregularities of go back to previous versions.

**+ Sharing:**  use a distant repository to share your container with others.

**+ Light and minimal overhead:**  Docker image are typically very small, which promotes rapid delivery and reduces the time to deploy new application containers..

**+ Try running applications with other operating systems easily without setting up a virtual machine again**

**+ Parallel test program:** Docker creates a lot of identical containers, and we divide the task for a while. Continuously apply all available server resources to test.

**+ Share and archive – Setup projects for teams:** With Docker, it's effective to have a share container action. You will reduce more than 1/2 documents for setup for the environment. Convenient, simple and easy for the members of the system when participating in the project.

* **Disadvantage**

**+ Containers don’t work at bare-metal rates:** utilise resources more efficiently than virtual machines. But containers are still subject to performance overhead due to overlay networking, interfacing between containers and the host system and so on. If you want 100 percent bare-metal performance, you need to use bare metal, not containers.

**+ The containers ecosystems is split:**  the core Docker platform is open source, some container products don’t work with other ones.

**+ Data storage is intricate.**

**+ Graphical application do not operate well:** Docker was created as a solution for deploying server applications that don’t need a graphical interface. Docker is Platform-dependent: Though Docker is showcasing that it supports Windows and Mac OS X also, it utilizes virtual machines to run on non-Linux platforms.

**III. Docker Components :**

1. **Docker engine:**

**Docker engine** is a client-server application with these major components:

* **Server**: **Daemon** process (long-running program).
* **REST API**: helps program can talk to the daemon and instruct it what to do.
* **Client**: **CLI** (Command line interface) uses **REST API** to control or interact with **Daemon** through scripting or direct **CLI** commands.



1. **Docker daemon (dockerd):**

**The Docker daemon** listens for **Docker API** requests and manages Docker objects. A daemon can also communicate with other daemons to manage Docker services.

**The Docker daemon** creates and manages Docker objects such as *images*, *containers*, *networks* and *volumes*.

1. **Docker client:**

**Docker client** is the primary way that many Docker users interact with Docker.

When you use command such as ‘ docker run ’, the client uses the **Docker API** to sends these commands to **Docker Daemon**.

**Docker client** can communicate with more than one daemon.

1. **Docker REST API:**

**Docker** provides an **API** for interacting with the **Docker daemon** (called the **Docker Engine API**), as well as SDKs for Go and Python.

The **Docker Engine API** is a **RESTful API** accessed by an HTTP client such as wget or curl , or the HTTP library which is part of most modern programming languages.

1. **Images:**

An ***images*** is a read-only template with instructions for creating a **Docker container**. Often, an image is based on another image with some additional customization. In some cases, you can create your own images.

To build your own image, you create a ***Dockerfile*** with simple syntax for defining steps to create image and run it. Each instruction in ***Dockerfile*** create a layer in the image. Every time you change ***Dockerfile*** and rebuild the image, only layers which have been changed are rebuilt. This is why images so lightweight, small and fast compared to other virtualization technologies.

1. **Containers:**

A ***container*** is a runnable instance of an **image**. You can create, start, stop, move or delete a container using the **Docker API** or **CLI**. You can connect a ***container*** to one or more networks, attach storage to it, or even create a new **image** based on its current state.

A ***container*** is defined by its **image** as well as any configuration options you provide to it when you create or start it. When a ***container*** is removed, any changes to its state that are not stored in persistent storage disappear.

1. **Data volumes:**

**General**

By default all files created inside a **container** are stored on a writable container layer. Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: *volumes*, and *bind mounts*.

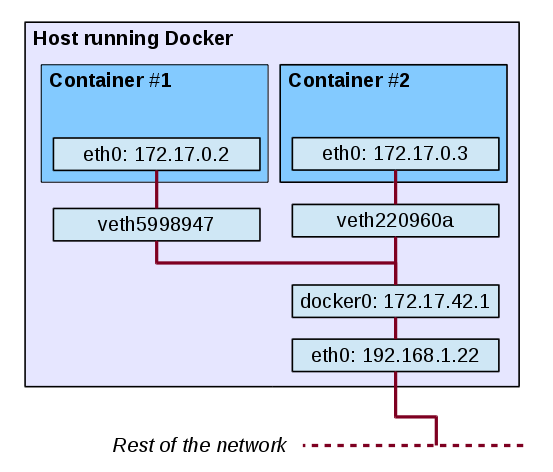
**Volumes** are stored in a part of the host filesystem which is *managed by Docker*. **Volumes** are the best way to persist data in Docker. When you mount the **volume** into a **container**, this directory is what is mounted into the container. This is similar to the way that bind mounts work, except that volumes are managed by Docker and are isolated from the core functionality of the host machine.

## Good use cases for bind mounts

In general, you should use volumes where possible. Bind mounts are appropriate for the following types of use case:

* Sharing configuration files from the host machine to containers.
* Sharing source code or build artifacts between a development environment on the Docker host and a container.
* When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

1. **Networks:**



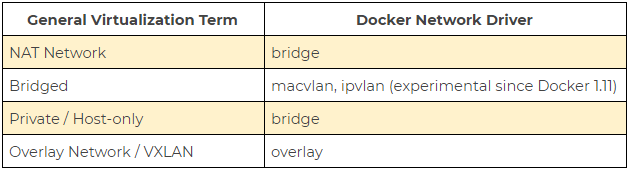
When we setup Docker, the following settings will be made:

* Virtual docker0 bridge will be created
* Docker find a subnet which is not used on the host and assign an address to docker0

Then, when we start a container (with bridge network), a veth (Virtual Ethernet) will be created which one end is connected to docker0 and one end will be connected to interface eth0 on the container.

**8.1 Default Networks:**

By default, when installing, Docker will create three default network cards: bridge - host - none. Docker also have different **network drivers** for alternating other virtualized algae backgrounds.



**8.2 User-defined networks:**

Beside using the default network provided by docker. We can define network ranges that are suitable for our work.

The **container** that you run on this network must belong to the same Docker host. Each network container can communicate with other **containers** in the same **network**.

1. **Registries:**

A **Docker Registry** stores **images.** Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.

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.

If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR). Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker.

**IV. Distribution tools:**

A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions.

A Docker registry is organized into **Docker repositories(**a docker repository is where you can store 1 or more versions of a specific Docker image. An image can have 1 or more versions**)** , where a repository holds all the versions of a specific image. The registry allows Docker users to pull images locally, as well as push new images to the registry (given adequate access permissions when applicable).

By default, the Docker engine interacts with **Docker Hub**, Docker's public registry instance. However, it is possible to run on-premise the open-source Docker registry/distribution, as well as a commercially supported version called **Docker Trusted Registry**. There are other public registries available online.

To pull an image from an on-premises registry, you could run a command similar to:

|  |
| --- |
| docker pull my-registry:9000/foo/bar:2.1 |

where you pull the version of foo/bar image with tag 2.1 from our on-premise registry located at my-registry domain, port 9000 .

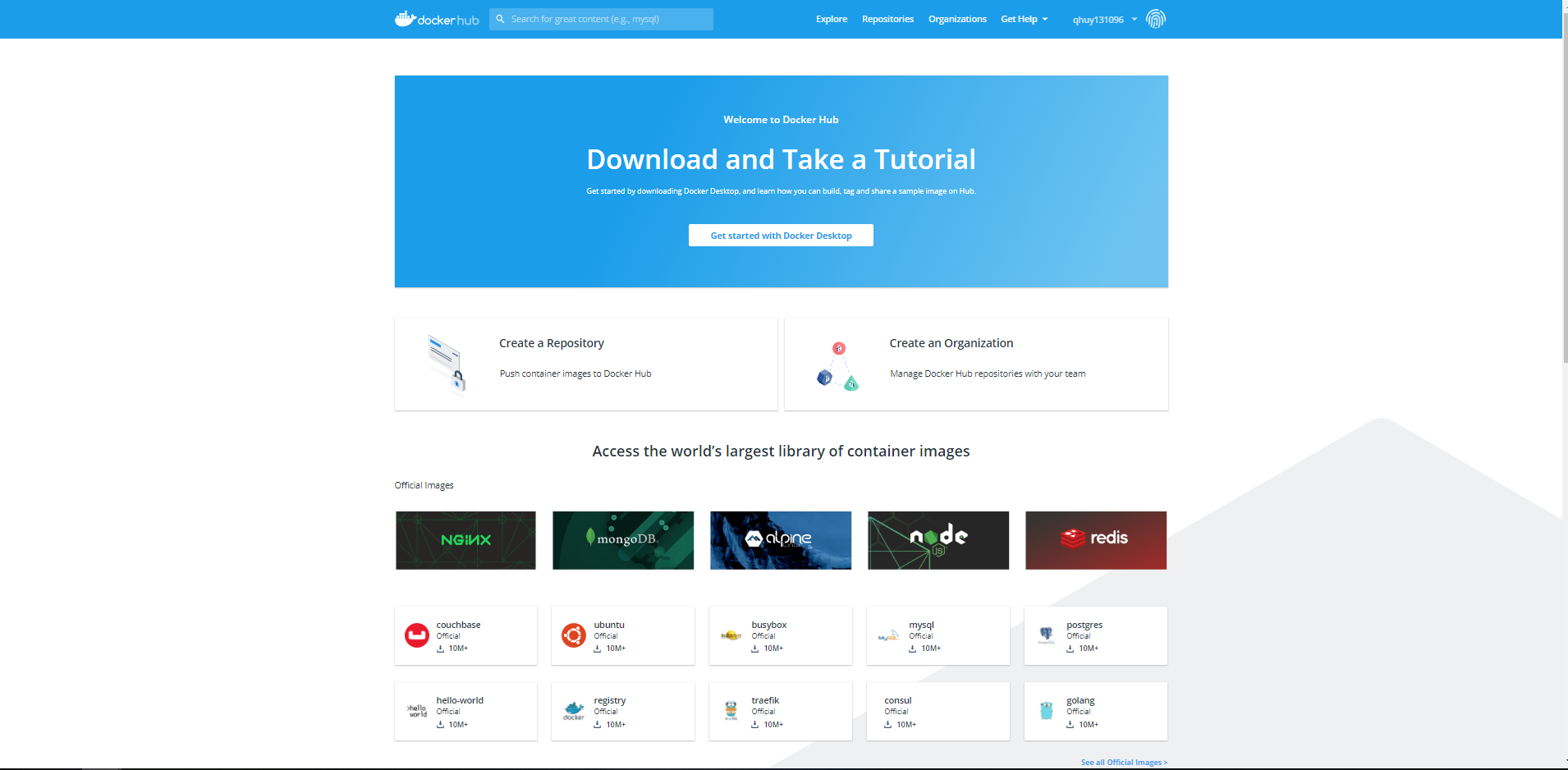
If you used DockerHub instead, and 2.1 was also the latest version, you could run this command to pull the same image locally:

|  |
| --- |
| docker pull foo/bar |

1. **Docker hub:**

[Docker Hub](https://hub.docker.com/) is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to [Docker Cloud](https://docs.docker.com/v17.12/docker-cloud/) so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, [user and team collaboration](https://docs.docker.com/v17.12/docker-hub/orgs/), and workflow automation throughout the development pipeline.

Log in to Docker Hub and Docker Cloud using [Docker ID](https://docs.docker.com/v17.12/docker-hub/accounts/).



Docker Hub provides the following major features:

* [Repositories](https://docs.docker.com/docker-hub/repos/): Push and pull container images.
* [Teams & Organizations](https://docs.docker.com/docker-hub/orgs/): Manage access to private repositories of container images.
* [Official Images](https://docs.docker.com/docker-hub/official_images/): Pull and use high-quality container images provided by Docker.
* [Publisher Images](https://docs.docker.com/docker-hub/publish/customer_faq/): Pull and use high-quality container images provided by external vendors. Certified images also include support and guarantee compatibility with Docker Enterprise.
* [Builds](https://docs.docker.com/docker-hub/builds/): Automatically build container images from GitHub and Bitbucket and push them to Docker Hub
* [Webhooks](https://docs.docker.com/docker-hub/webhooks/): Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

1. **Use official Repositories.**

Docker Hub contains a number of [Official Repositories](http://hub.docker.com/explore/). These are public, certified repositories from vendors and contributors to Docker. They contain Docker images from vendors like Canonical, Oracle, and Red Hat that you can use as the basis to build your applications and services.

With Official Repositories you know you’re using an optimized and up-to-date image that was built by experts to power your applications.

1. **Docker command.**

### Docker itself provides access to Docker Hub services via the [docker search](https://docs.docker.com/v17.12/engine/reference/commandline/search/), [pull](https://docs.docker.com/v17.12/engine/reference/commandline/pull/), [login](https://docs.docker.com/v17.12/engine/reference/commandline/login/), and [push](https://docs.docker.com/v17.12/engine/reference/commandline/push/) commands.

1. **Docker compose.**

### Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.

Install docker compose:

Docker Compose relies on Docker Engine for any meaningful work, so make sure you have Docker Engine installed either locally or remote, depending on your setup.

* On desktop systems like Docker Desktop for Mac and Windows, Docker Compose is included as part of those desktop installs.
* On Linux systems, first install the [Docker](https://docs.docker.com/install/#server) for your OS as described on the Get Docker page, then come back here for instructions on installing Compose on Linux systems.
* To run Compose as a non-root user, see [Manage Docker as a non-root user](https://docs.docker.com/install/linux/linux-postinstall/).

The features of Compose that make it effective are:

* [Multiple isolated environments on a single host](https://docs.docker.com/compose/overview/#Multiple-isolated-environments-on-a-single-host): Compose uses a project name to isolate environments from each other. You can make use of this project name in several different contexts.
* [Preserve volume data when containers are created](https://docs.docker.com/compose/overview/#preserve-volume-data-when-containers-are-created): Compose preserves all volumes used by your services. When docker-compose up runs, if it finds any containers from previous runs, it copies the volumes from the old container to the new container.
* [Only recreate containers that have changed](https://docs.docker.com/compose/overview/#only-recreate-containers-that-have-changed): Compose caches the configuration used to create a container. When you restart a service that has not changed, Compose re-uses the existing containers.
* [Variables and moving a composition between environments](https://docs.docker.com/compose/overview/#variables-and-moving-a-composition-between-environments): Your configuration options can contain environment variables. Compose uses the variable values from the shell environment in which docker-compose is run.

1. **Other Public Repositories:**

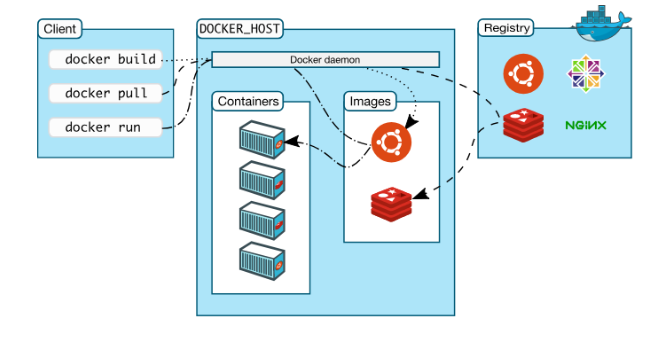
Other companies host paid online Docker registries for public use. Cloud providers like AWS and Google, who also offer container-hosting services, market the high availability of their registries.

* [**Amazon Elastic Container Registry**](https://aws.amazon.com/ecr/) **(ECR)**
* [**Google Container Registry**](https://cloud.google.com/container-registry/) **(GCR)**
* [**Azure Container Registry**](https://azure.microsoft.com/en-us/services/container-registry/) **(ACR)**
* [**CoreOS Quay**](https://quay.io/)
* [**Private Docker Registry**](https://private-docker-registry.com/)

**V. Docker architecture:**

Docker uses a client-server architecture.The Docker client and Docker server( daemon) *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.manage Docker services.  
 Docker consists of 3 main components:

* Docker Client.
* Docker Host.
* Docker Registry ( Hub).



1. **Docker daemon:**

The Docker daemon listens for the Docker client through API or CLI and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate

with other daemons to.

1. **Docker client:**

The Docker client is the primary way that many Docker users interact with Docker. When you use commands such as ‘ docker run’, the client sends these commands to the daemon, which carries them out. The Docker client can communicate with more than one daemon.

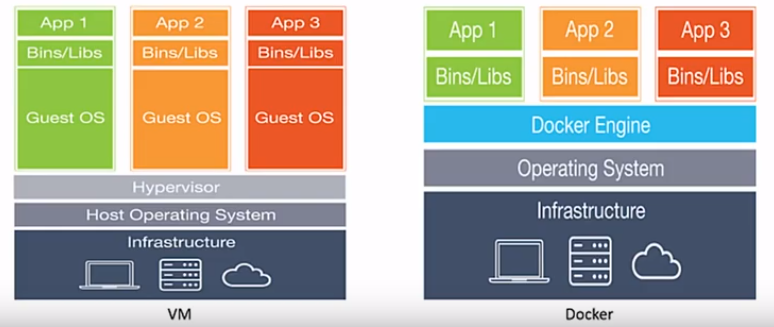
1. **Docker registry:**

The 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.

1. **Docker objects:**

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.

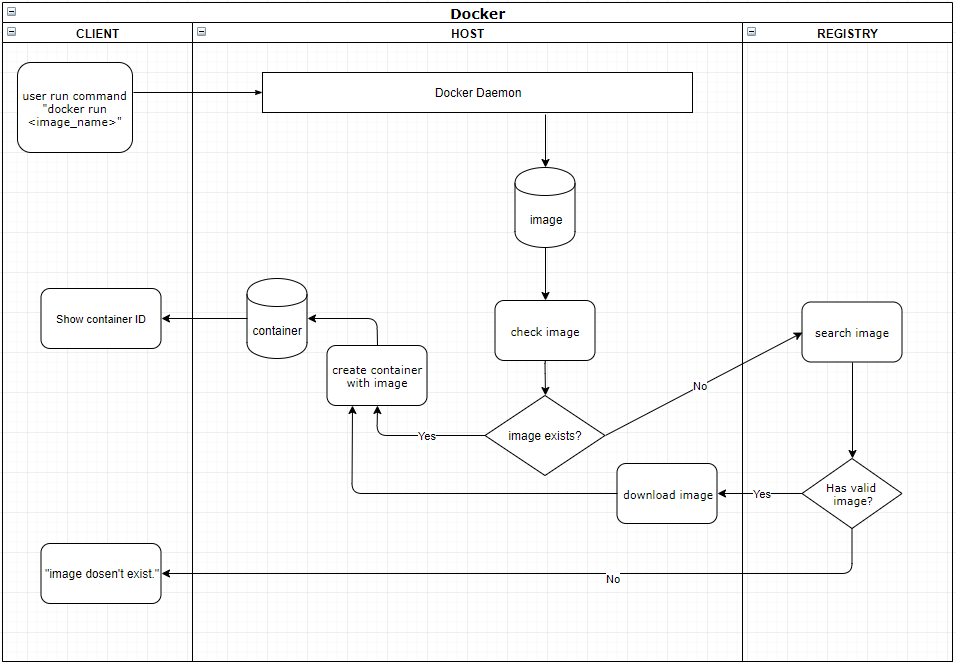
1. **Distinguish between Docker and Virtual Machine:**



|  |  |  |
| --- | --- | --- |
| **The basis of comparison Between Docker vs VMs** | **Dockers** | **Virtual Machines** |
| **Boot time** | Dockers can boot in seconds | It often takes minutes for VMs to boot |
| **Execution** | Makes use of execution engine | Makes use of a hypervisor |
| **Memory** | More memory efficient as no space needed to virtualize | Less memory efficient as the entire OS needs to be loaded before starting the service |
| **Isolation** | No provision for isolation of systems and hence are more prone to adversities | Efficient isolation mechanism and hence interference possibility is less |
| **Ease of deployment** | Deploying through dockers is extremely easy as only one image, containerized, can be used across different operating systems. | Deploying in Virtual Machines is a comparatively lengthy process where separate instances are responsible for the execution |
| **Ease of usage** | Dockers have comparatively complex usage mechanism which consists of both third party and docker managed tools | The tools associated with a VM are comparatively easier to use and simpler to work with. |

**How does Docker work?**

Example “ docker run <image\_name>” command.



**VI. Cheat sheet:**

1. **Installation:**

Lifecycle

* Linux: curl -sSL https://get.docker.com/ | sh
* MacOS: https://download.docker.com/mac/stable/Docker.dmg
* Windows: https://download.docker.com/win/stable/InstallDocker.msi

Check version

* [docker version](https://docs.docker.com/engine/reference/commandline/version/) shows which version of docker you have running.

1. **Containers:**

Lifecycle

* [docker create](https://docs.docker.com/engine/reference/commandline/create) creates a container but does not start it.
* [docker rename](https://docs.docker.com/engine/reference/commandline/rename/) allows the container to be renamed.
* [docker run](https://docs.docker.com/engine/reference/commandline/run) creates and starts a container in one operation.
* [docker rm](https://docs.docker.com/engine/reference/commandline/rm) deletes a container.
* [docker update](https://docs.docker.com/engine/reference/commandline/update/) updates a container’s resource limits.

Starting & Stopping

* [docker start](https://docs.docker.com/engine/reference/commandline/start) starts a container so it is running.
* [docker stop](https://docs.docker.com/engine/reference/commandline/stop) stops a running container.
* [docker restart](https://docs.docker.com/engine/reference/commandline/restart) stops and starts a container.
* [docker pause](https://docs.docker.com/engine/reference/commandline/pause/) pauses a running container, "freezing" it in place.
* [docker unpause](https://docs.docker.com/engine/reference/commandline/unpause/) will unpause a running container.
* [docker wait](https://docs.docker.com/engine/reference/commandline/wait) blocks until running container stops.
* [docker kill](https://docs.docker.com/engine/reference/commandline/kill) sends a SIGKILL to a running container.
* [docker attach](https://docs.docker.com/engine/reference/commandline/attach) will connect to a running container.

1. **Images:**

Lifecycle

* [docker images](https://docs.docker.com/engine/reference/commandline/images) shows all images.
* [docker import](https://docs.docker.com/engine/reference/commandline/import) creates an image from a tarball.
* [docker build](https://docs.docker.com/engine/reference/commandline/build) creates image from Dockerfile.
* [docker commit](https://docs.docker.com/engine/reference/commandline/commit) creates image from a container, pausing it temporarily if it is running.
* [docker rmi](https://docs.docker.com/engine/reference/commandline/rmi) removes an image.
* [docker load](https://docs.docker.com/engine/reference/commandline/load) loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
* [docker save](https://docs.docker.com/engine/reference/commandline/save) saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

Info

* [docker history](https://docs.docker.com/engine/reference/commandline/history) shows history of image.
* [docker tag](https://docs.docker.com/engine/reference/commandline/tag) tags an image to a name (local or registry).

1. **Networks:**

Lifecycle

* [docker network create docker network create -d overlay MyOverlayNetwork](https://docs.docker.com/engine/reference/commandline/network_create/)
* [docker network rm](https://docs.docker.com/engine/reference/commandline/network_rm/) docker network rm MyOverlayNetwork

Info

* [docker network ls](https://docs.docker.com/engine/reference/commandline/network_ls/)
* [docker network inspect](https://docs.docker.com/engine/reference/commandline/network_inspect/)

Connection

* [docker network connect](https://docs.docker.com/engine/reference/commandline/network_connect/)
* [docker network disconnect](https://docs.docker.com/engine/reference/commandline/network_disconnect/)

1. **Registry & Repository:**

* [docker login](https://docs.docker.com/engine/reference/commandline/login) to login to a registry.
* [docker logout](https://docs.docker.com/engine/reference/commandline/logout) to logout from a registry.
* [docker search](https://docs.docker.com/engine/reference/commandline/search) searches registry for image.
* [docker pull](https://docs.docker.com/engine/reference/commandline/pull) pulls an image from registry to local machine.
* [docker push](https://docs.docker.com/engine/reference/commandline/push) pushes an image to the registry from local machine.

1. **Dockerfile:**

* [.dockerignore](https://docs.docker.com/engine/reference/builder/#dockerignore-file)
* [FROM](https://docs.docker.com/engine/reference/builder/#from) Sets the Base Image for subsequent instructions.
* [MAINTAINER (deprecated - use LABEL instead)](https://docs.docker.com/engine/reference/builder/#maintainer-deprecated) Set the Author field of the generated images.
* [RUN](https://docs.docker.com/engine/reference/builder/#run) execute any commands in a new layer on top of the current image and commit the results.
* [CMD](https://docs.docker.com/engine/reference/builder/#cmd) provide defaults for an executing container.
* [EXPOSE](https://docs.docker.com/engine/reference/builder/#expose) informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.
* [ENV](https://docs.docker.com/engine/reference/builder/#env) sets environment variable.
* [ADD](https://docs.docker.com/engine/reference/builder/#add) copies new files, directories or remote file to container. Invalidates caches. Avoid [ADD](https://docs.docker.com/engine/reference/builder/#add) and use [COPY](https://docs.docker.com/engine/reference/builder/#copy) instead.
* [COPY](https://docs.docker.com/engine/reference/builder/#copy) copies new files or directories to container. By default this copies as root regardless of the USER/WORKDIR settings. Use --chown=<user>:<group> to give ownership to another user/group. (Same for ADD.)
* [ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint) configures a container that will run as an executable.
* [VOLUME](https://docs.docker.com/engine/reference/builder/#volume) creates a mount point for externally mounted volumes or other containers.
* [USER](https://docs.docker.com/engine/reference/builder/#user) sets the user name for following RUN / CMD / ENTRYPOINT commands.
* [WORKDIR](https://docs.docker.com/engine/reference/builder/#workdir) sets the working directory.
* [ARG](https://docs.docker.com/engine/reference/builder/#arg) defines a build-time variable.
* [ONBUILD](https://docs.docker.com/engine/reference/builder/#onbuild) adds a trigger instruction when the image is used as the base for another build.
* [STOPSIGNAL](https://docs.docker.com/engine/reference/builder/#stopsignal) sets the system call signal that will be sent to the container to exit.
* [LABEL](https://docs.docker.com/config/labels-custom-metadata/) apply key/value metadata to your images, containers, or daemons.

1. **Volumes:**

Lifecycle

* [docker volume create : Create a local volume](https://docs.docker.com/engine/reference/commandline/volume_create/)
* [docker volume rm](https://docs.docker.com/engine/reference/commandline/volume_rm/): destroy a volume.
* [docker run -v:](https://docs.docker.com/engine/reference/commandline/run/) mounting a volume on container start.

Info

* [docker volume ls:](https://docs.docker.com/engine/reference/commandline/volume_ls/)  list volume.
* [docker volume inspect](https://docs.docker.com/engine/reference/commandline/volume_inspect/) : Display detailed information on one or more volumes

1. **Cleaning Docker:**

* [docker-gc](https://github.com/spotify/docker-gc) that will safely clean up images that are no longer used by any containers
* [docker rm nginx](https://docs.docker.com/engine/reference/commandline/rmi/): removing a Running Container.
* [docker rmi Image:](https://docs.docker.com/v17.12/engine/reference/commandline/rmi/) remove an image.
* docker stop: stop docker
* docker rm Id\_or\_Name: Remove Container
* docker rm -v container\_name: Remove a container and its volume

1. **Debug:**

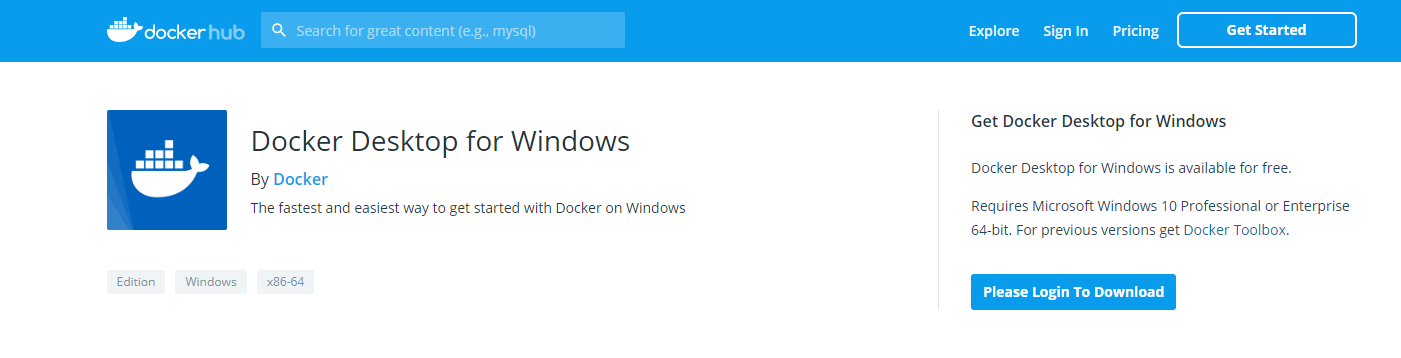
* [docker exec](https://docs.docker.com/engine/reference/commandline/exec/) : run another process in running container.
* docker logs -f: show lives logs of running daemon container.
* [docker port:](https://docs.docker.com/engine/reference/commandline/port/) show exposed ports of a container.

**VII. Install docker:**

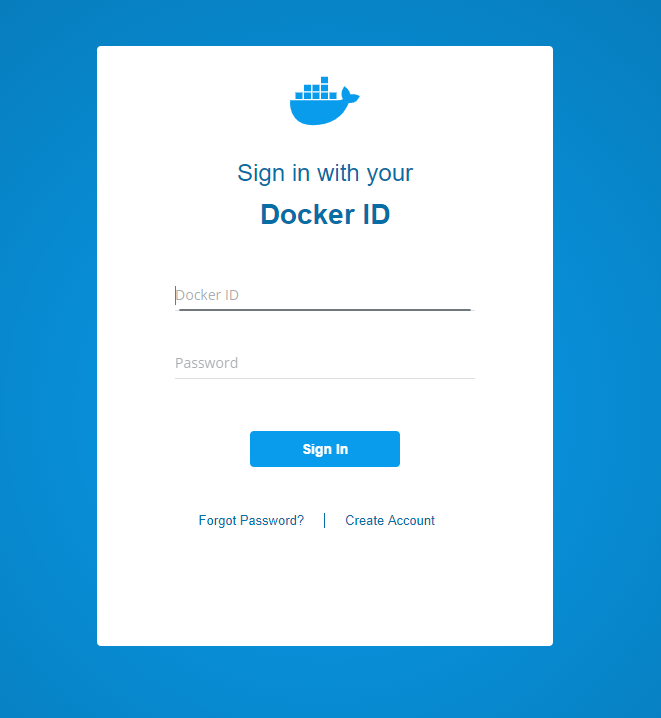
1. **Docker desktop for Windows:**

Step 1: Download Docker

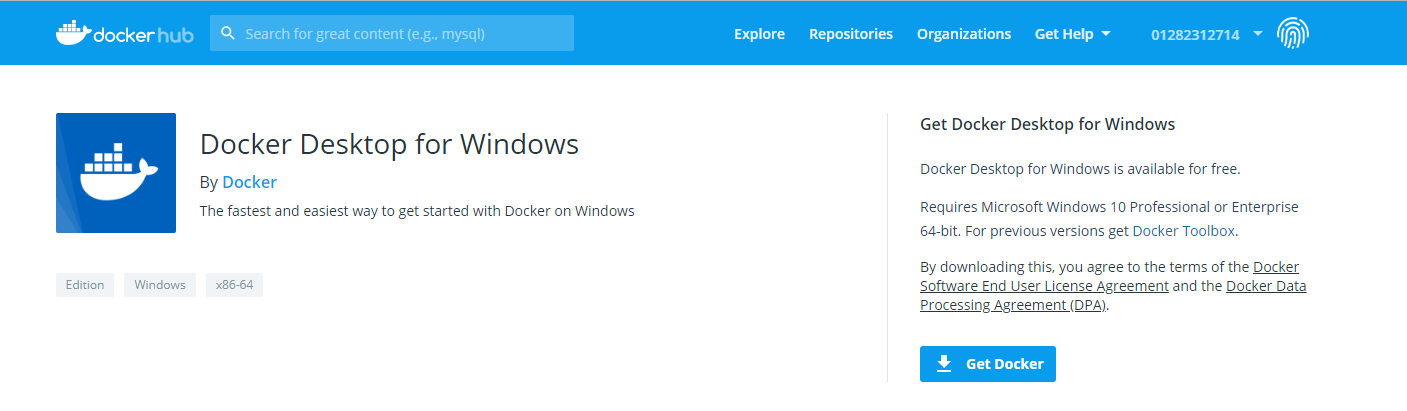
Officially Docker installer can be downloaded from [Docker Store](https://hub.docker.com/editions/community/docker-ce-desktop-windows) only. You will have to create an account to be able to download it. Having an account for Docker is a very good things. It allows you to download docker images in the future.



Enter Docker Store



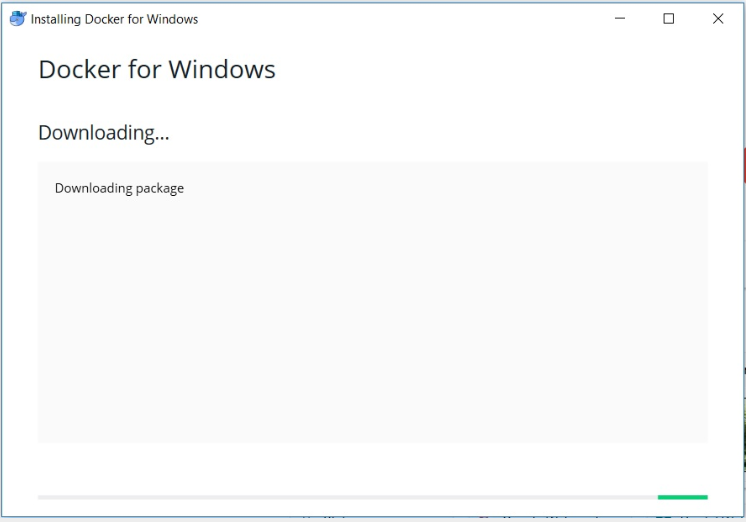
Login to download



Ready to download

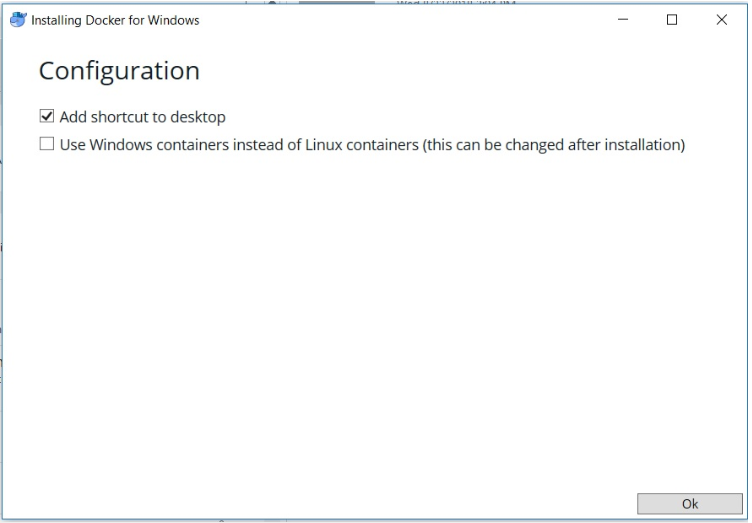
Step 2: Run the installer.

Double click on the downloaded installer file to start the installation wizard. You will see Docker installer downloading additional files required. If you don’t have internet connection, installer will move on to the next step. Wait for the process to complete and you will see the configuration screen.

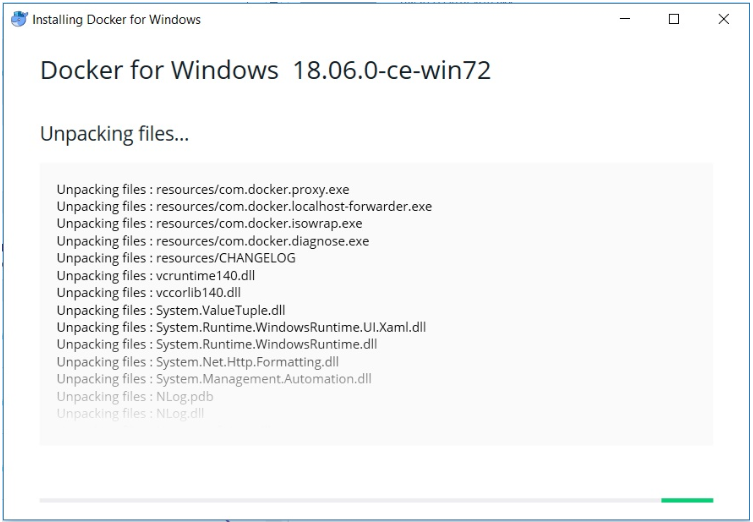


Step 3: Configuration Settings

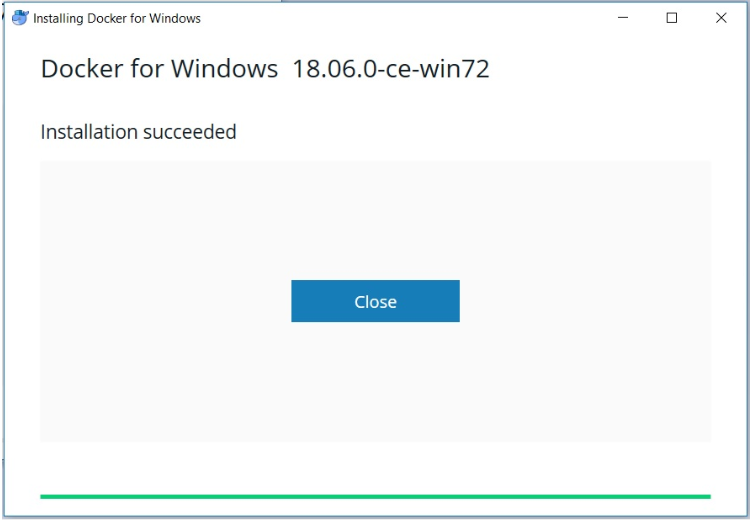
In this dialog box you will be asked if you want to create desktop icon for Docker. I leave this checked. Second option is if you want to use Linux or Windows Container. This option can be changed later on. I leave it as default, that is unchecked.



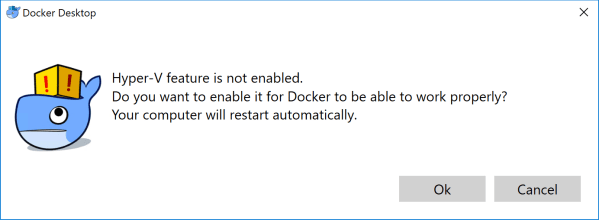
*Click OK to start the installation. Wait for the installation to complete*.



*Once the installation completes, click on Close to complete the installation.*



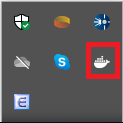
*Click Close to finish install.*



*Click OK to enable Hyper-V for Docker*

Step 4: Run Docker.

Once the installation completes run Docker by double clicking the icon created on the desktop or from start menu. You will see a docker icon appear on your windows taskbar. If you hover your mouse over it, it will say “Docker is Starting”. Wait for Docker to start.

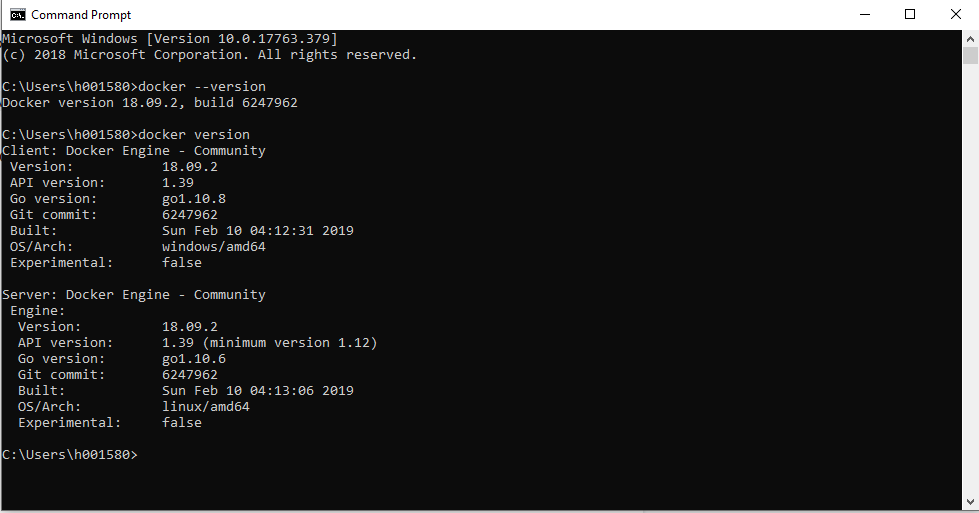


Step 5: Check the version of Docker installed.

Most of the time you will be working with command line to work with Docker. Let this be your first command to check Docker version and see what you get.

Open Powershell or command prompt and enter the command: **docker version**

You should see something like this.



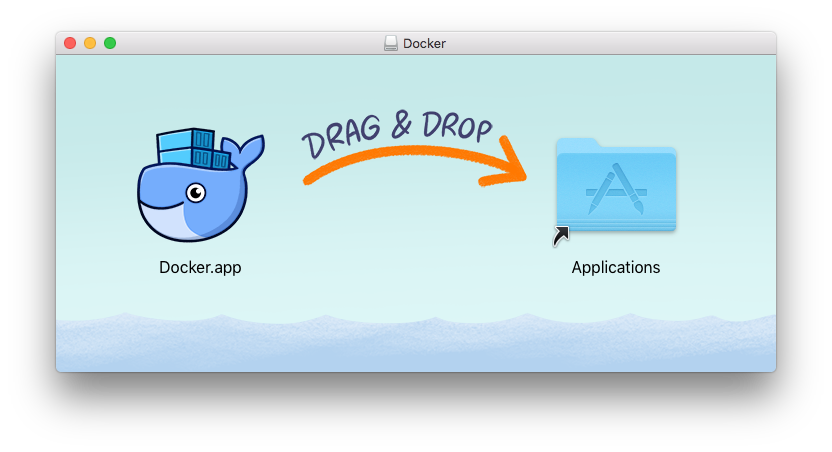
1. **Docker desktop for MAC**

Step 1: Download Docker

Officially Docker installer can be downloaded from [Docker Store](https://hub.docker.com/editions/community/docker-ce-desktop-mac) only. You will have to create an account to be able to download it. Having an account for Docker is a very good things. It allows you to download docker images in the future.

Step 2: Run the installer

Double-click Docker.dmg to open the installer, then drag Moby the whale to the Applications folder



Step 3: Run Docker

## Double-click Docker.app in the Applications folder to start Docker. (In the example below, the Applications folder is in “grid” view mode.)

## Docker app in Hockeyapp

## You are prompted to authorize Docker.app with your system password after you launch it. Privileged access is needed to install networking components and links to the Docker apps. The whale in the top status bar indicates that Docker is running, and accessible from a terminal.

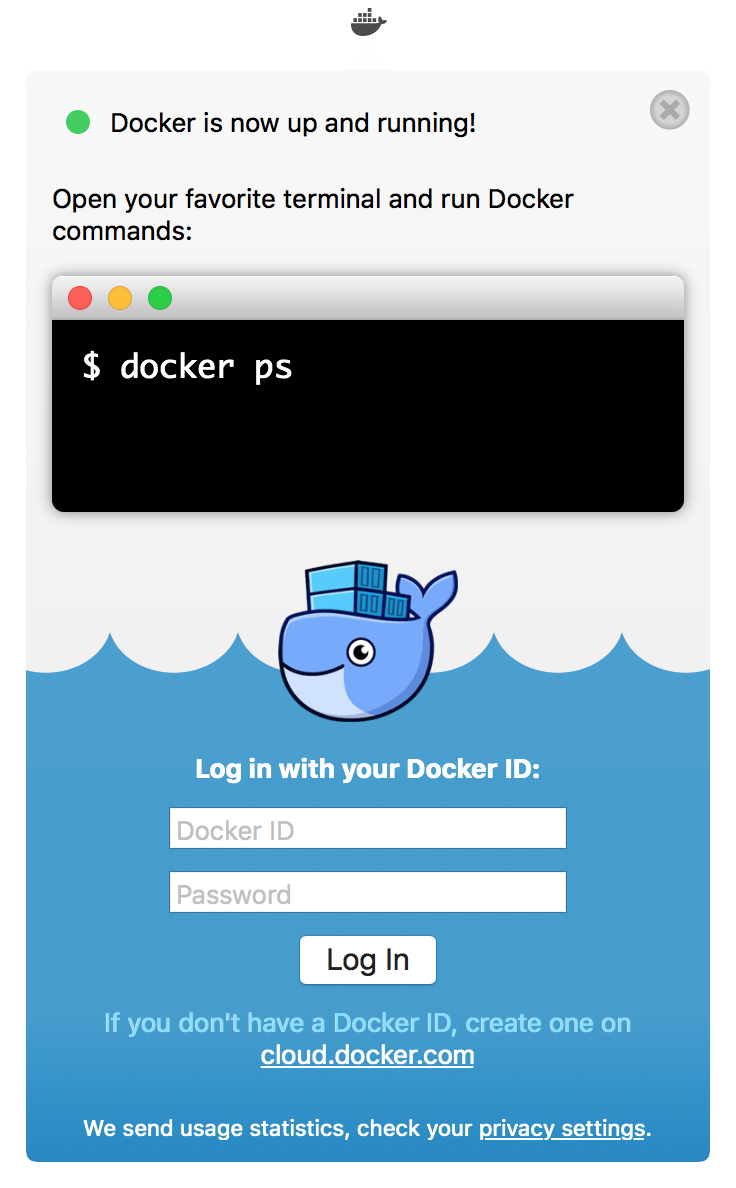
## 

## 

## 

## Whale in menu bar

## If you just installed the app, you also get a success message with suggested next steps and a link to this documentation. Click the whale (whale menu) in the status bar to dismiss this popup.



Click the whale (whale menu) to get Preferences and other options.

Select **About Docker** to verify that you have the latest version.

Congratulations! You are up and running with Docker Desktop for Mac.

**VIII. Demo:**

**Environment:**

# Docker: Docker version 18.09.2, build 6247962

# Docker compose: docker-compose version 1.23.2, build 1110ad01

# Docker machine: docker-machine.exe version 0.16.1, build cce350d7

# OS

# Windows 10: Windows Pro 64-bit (10.0, Build 17763)

# 

# Laravel

# Webserver: Nginx 1.17.1

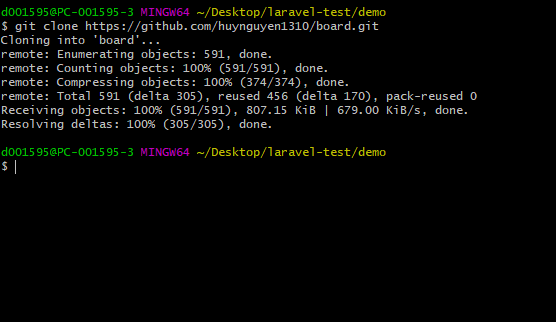
# PHP: 7.2-fpm

# MySql: 5.7.22

## 

## Step 1 — Downloading Laravel

|  |
| --- |
| $ git clone https://github.com/huynguyen1310/board.git |



*Get laravel project from Git*

Step 2 — Creating the Docker File:

We will create 4 files:

1. **docker-compose.yml** : defines our web server, database, and application services.
2. **Dockerfile** : enables you to create custom images that you can use to install the software required by your application and configure settings based on your requirements.
3. **nginx/config.d/app.conf** : you can modify the Nginx service to use PHP-FPM as the FastCGI server to serve dynamic content.
4. **mysql/my.cnf** : this is the file that you bind-mounted to /etc/mysql/my.cnf inside the container.

**docker-compose.yml**

* PHP service

|  |
| --- |
| version: '3'  services:  #PHP Service  app:  build:  context: .  dockerfile: Dockerfile  image: php  container\_name: app  restart: unless-stopped  tty: true  environment:  SERVICE\_NAME: app  SERVICE\_TAGS: dev  working\_dir: /var/www  volumes:  - ./:/var/www  - ./php/local.ini:/usr/local/etc/php/conf.d/local.ini  networks:  - app-network |

Some Explanation:

* version 3 : version of docker-compose
* context : dockerfile directory
* dockerfile : describes which image will be used be services, ...
* working\_dir : where the app code will be inside the container
* volumes : mount directory ./ (contain source code) on host into /var/www (in container)
* Web server service

|  |
| --- |
| #Nginx Service  webserver:  image: nginx:alpine  container\_name: webserver  restart: unless-stopped  tty: true  ports:  - "80:80"  - "443:443"  volumes:  - ./:/var/www  - ./nginx/conf.d/:/etc/nginx/conf.d/  networks:  - app-network |

Some Explanation:

* ports: - 80:80 : Map port 80 on the virtual machine to port 80 on the container.
* Database service:

|  |
| --- |
| #MySQL Service  db:  image: mysql:5.7.22  container\_name: db  restart: unless-stopped  tty: true  ports:  - "3306:3306"  environment:  MYSQL\_USERNAME: root  MYSQL\_DATABASE: laravel  MYSQL\_ROOT\_PASSWORD: secret  SERVICE\_TAGS: dev  SERVICE\_NAME: mysql  volumes:  - dbdata:/var/lib/mysql  - ./mysql/my.cnf:/etc/mysql/my.cnf  networks:  - app-network |

Some Explanation:

* image: mysql:5.7.22 : mySql service version
* dbdata:/var/lib/mysql : Create a named volume
* environment : Environment variables. With the above definition, mysql will give us a database and such a user. (If not specified, there will only be 1 user root and the default database of mysql)
* We used **laravel** as the database, **root** as user name & **secret** as the password as these values match what can be found in the default .env that ships with Laravel, meaning we won’t have to change it there.

Final **docker-compose.yml**:

|  |
| --- |
| version: '3'  services:  #PHP Service  app:  build:  context: .  dockerfile: Dockerfile  image: digitalocean.com/php  container\_name: app  restart: unless-stopped  tty: true  environment:  SERVICE\_NAME: app  SERVICE\_TAGS: dev  working\_dir: /var/www  volumes:  - ./:/var/www  - ./php/local.ini:/usr/local/etc/php/conf.d/local.ini  networks:  - app-network  #Nginx Service  webserver:  image: nginx:alpine  container\_name: webserver  restart: unless-stopped  tty: true  ports:  - "80:80"  - "443:443"  volumes:  - ./:/var/www  - ./nginx/conf.d/:/etc/nginx/conf.d/  networks:  - app-network  #MySQL Service  db:  image: mysql:5.7.22  container\_name: db  restart: unless-stopped  tty: true  ports:  - "3306:3306"  environment:  MYSQL\_USERNAME: root  MYSQL\_DATABASE: laravel  MYSQL\_ROOT\_PASSWORD: secret  SERVICE\_TAGS: dev  SERVICE\_NAME: mysql  volumes:  - dbdata:/var/lib/mysql  - ./mysql/my.cnf:/etc/mysql/my.cnf  networks:  - app-network  #Docker Networks  networks:  app-network:  driver: bridge  #Volumes  volumes:  dbdata:  driver: local |

**Dockerfile:**

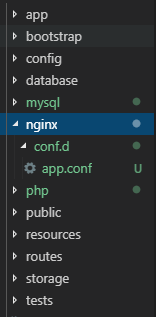
This **Dockerfile** will set the base image and specify the necessary commands and instructions to build the Laravel application image. Add the following code to the file:

|  |
| --- |
| FROM php:7.2-fpm  # Copy composer.lock and composer.json  COPY composer.lock composer.json /var/www/  # Set working directory  WORKDIR /var/www  # Install dependencies  RUN apt-get update && apt-get install -y \  build-essential \  mysql-client \  libpng-dev \  libjpeg62-turbo-dev \  libfreetype6-dev \  locales \  zip \  jpegoptim optipng pngquant gifsicle \  vim \  unzip \  git \  curl  # Clear cache  RUN apt-get clean && rm -rf /var/lib/apt/lists/\*  # Install extensions  RUN docker-php-ext-install pdo\_mysql mbstring zip exif pcntl  RUN docker-php-ext-configure gd --with-gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ --with-png-dir=/usr/include/  RUN docker-php-ext-install gd  # Install composer  RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer  # Add user for laravel application  RUN groupadd -g 1000 www  RUN useradd -u 1000 -ms /bin/bash -g www www  # Copy existing application directory contents  COPY . /var/www  # Copy existing application directory permissions  COPY --chown=www:www . /var/www  # Change current user to www  USER www  # Expose port 9000 and start php-fpm server  EXPOSE 9000  CMD ["php-fpm"] |

* First, the Dockerfile creates an image on top of the [php:7.2-fpm Docker image](https://hub.docker.com/_/php/). This is a Debian-based image that has the PHP FastCGI implementation [PHP-FPM](https://php-fpm.org/) installed. The file also installs the prerequisite packages for Laravel: mcrypt, pdo\_mysql, mbstring, and imagick with composer.
* The RUN directive specifies the commands to update, install, and configure settings inside the container, including creating a dedicated user and group called www. The WORKDIR instruction specifies the /var/www directory as the working directory for the application.
* Creating a dedicated user and group with restricted permissions mitigates the inherent vulnerability when running Docker containers, which run by default as root. Instead of running this container as root, we've created the www user, who has read/write access to the /var/www folder thanks to the COPY instruction that we are using with the --chown flag to copy the application folder's permissions.
* Finally, the EXPOSE command exposes a port in the container, 9000, for the php-fpm server. CMD specifies the command that should run once the container is created. Here, CMD specifies "php-fpm", which will start the server..

Configuring **Nginx:**

To configure Nginx, you will create an***app.conf*** file with the service configuration in the ***~/nginx/conf.d/***folder. Next, create the***app.conf*** configuration file.



Add the following code to the file to specify your Nginx configuration:

|  |
| --- |
| **server** {  **listen** 80;  **index** index.php index.html;  **error\_log** /var/log/nginx/error.log;  **access\_log** /var/log/nginx/access.log;  **root** /var/www/public;  **location** ~ \.php$ {  **try\_files** $uri =404;  **fastcgi\_split\_path\_info** ^(.+\.php)(/.+)$;  **fastcgi\_pass** app:9000;  **fastcgi\_index** index.php;  **include** fastcgi\_params;  **fastcgi\_param** SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  **fastcgi\_param** PATH\_INFO $fastcgi\_path\_info;  }  **location** / {  **try\_files** $uri $uri/ /index.php?$query\_string;  **gzip\_static** **on**;  }  } |

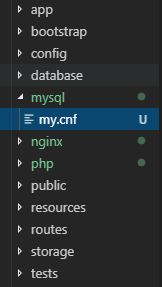
The [server block](https://www.digitalocean.com/community/tutorials/understanding-nginx-server-and-location-block-selection-algorithms) defines the configuration for the Nginx web server with the following directives:

* **listen**: This directive defines the port on which the server will listen to incoming requests.
* **error\_log** and **access\_log**: These directives define the files for writing logs.
* **root**: This directive sets the root folder path, forming the complete path to any requested file on the local file system.

Configuring **MySQL:**

To configure MySQL, you will create the **my.cnf** file in the **mysql** folder. This is the file that you bind-mounted to **/etc/mysql/my.cnf** inside the container. This bind mount allows you to override the **my.cnf** settings as and when required.

First, create the mysql directory. Next, make the **my.cnf** file:



In the file, add the following code to enable the query log and set the log file location:

|  |
| --- |
| **[mysqld]**  general\_log = 1  general\_log\_file = /var/lib/mysql/general.log |

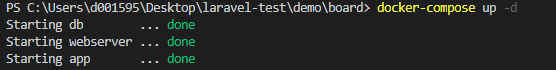
Step 3 — Running the Containers and Modifying Environment Settings:

We will make a copy of the .env.example file that Laravel includes by default and name the copy .env, which is the file Laravel expects to define its environment:

|  |
| --- |
| *$* cp .env.example .env |

With all of your services defined in your docker-compose file, you need start all of the containers, create the volumes, and set up and connect the networks:

|  |
| --- |
| *$* docker-compose up -d |



When you run **docker-compose** up for the first time, it will download all of the necessary Docker images, which might take a while. Once the images are downloaded and stored in your local machine, Compose will create your containers. The **-d** flag helps running your containers in the background.

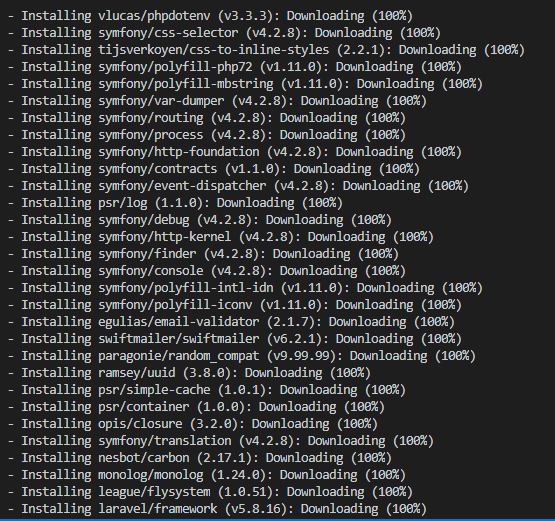
Remember to update the .env file, find the block that specifies DB\_CONNECTION and update it to reflect the specifics of your setup:

|  |
| --- |
| DB\_CONNECTION=mysql  DB\_HOST=db  DB\_PORT=3306  DB\_DATABASE=laravel  DB\_USERNAME=root  DB\_PASSWORD=secret |

* DB\_HOST will be your db database container.
* DB\_DATABASE will be the laravel database.
* DB\_USERNAME will be the username you will use for your database. In this case, we will use root.
* DB\_PASSWORD will be the secure password you would like to use for this user account.

Next, use Docker's [**composer** image](https://hub.docker.com/r/library/composer/) to mount the directories that you will need for your Laravel project and avoid the overhead of installing Composer globally:

|  |
| --- |
| *$* docker run --rm -v /$(pwd):/app composer install |



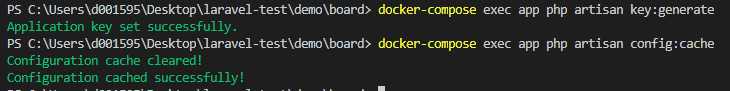
Next, set the application key for the Laravel application with the php artisan key:generate command. This command will generate a key and copy it to your .env file, ensuring that your user sessions and encrypted data remain secure:

|  |
| --- |
| *$* docker-compose exec app php artisan key:generate |

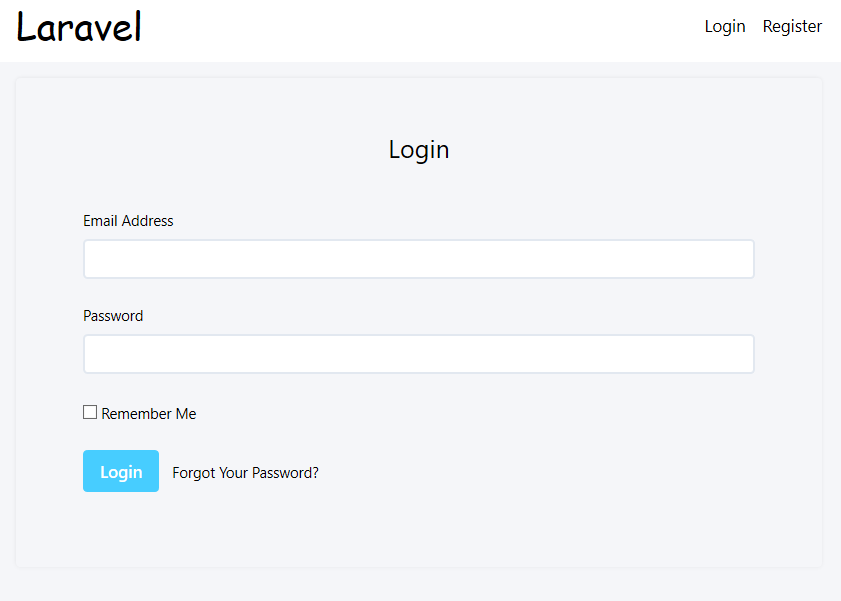
You now have the environment settings required to run your application. To cache these settings into a file, which will boost your application's load speed, run:

|  |
| --- |
| *$* docker-compose exec app php artisan config:cache |

Your configuration settings will be loaded into **/var/www/bootstrap/cache/config.php** on the container.



As a final step, visit http://localhost:80 in the browser. You will see the following home page for your Laravel application:



Conclusion

Key to the simplicity of this installation is Docker Compose, which allows you to create a group of Docker containers, defined in a single file, with a single command. If you would like to learn more about how to do CI with Docker Compose, take a look at [How To Configure a Continuous Integration Testing Environment with Docker and Docker Compose on Ubuntu 16.04](https://www.digitalocean.com/community/tutorials/how-to-configure-a-continuous-integration-testing-environment-with-docker-and-docker-compose-on-ubuntu-16-04). If you want to streamline your Laravel application deployment process then [How to Automatically Deploy Laravel Applications with Deployer on Ubuntu 16.04](https://www.digitalocean.com/community/tutorials/automatically-deploy-laravel-applications-deployer-ubuntu) will be a relevant resource.