# Docker

Docker overview

Docker is an open-source platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don’t need the extra load of a hypervisor, but run directly within the host machine’s kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

Docker provides tooling and a platform to manage the lifecycle of your containers:

* Develop your application and its supporting components using containers.
* The container becomes the unit for distributing and testing your application.
* When you’re ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker Engine

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

* A server which is a type of long-running program called a daemon process (the dockerd command).
* A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client (the docker command).



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

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

What can I use Docker for?

**Fast, consistent delivery of your applications**

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

* Your developers write code locally and share their work with their colleagues using Docker containers.
* They use Docker to push their applications into a test environment and execute automated and manual tests.
* When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
* When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

**Responsive deployment and scaling**

Docker’s container-based platform allows for highly portable workloads. Docker containers can run on a developer’s local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

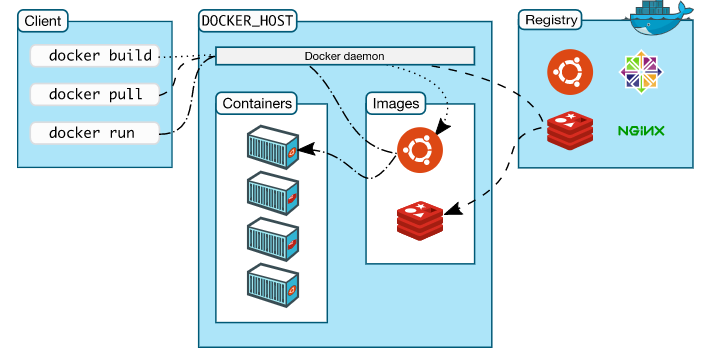
Docker’s portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

**Running more workloads on the same hardware**

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

Docker architecture

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



The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) 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 dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

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

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

IMAGES

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

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.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

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.

Example docker run command

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

$ docker run -i -t ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine’s network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

SERVICES

Services allow you to scale containers across multiple Docker daemons, which all work together as a *swarm* with multiple *managers* and *workers*. Each member of a swarm is a Docker daemon, and all the daemons communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.

When would we use Dockers:

There are lot of reasons to use docker. Although you will generally hear about docker used in conjunction with development and deployment of applications, there are a ton of examples for use:

\* Configuration simplification

\* Enhance Developer Productivity

\* Server Consolidation and management

\* Applicaiton isolation

\* Rapid Deployment

\* Build Management

# Get Started with Docker

### We have a complete container solution for you - no matter who you are and where you are on your containerization journey.



#### Docker Desktop

Developer productivity tools and a local Kubernetes environment.

[Download for  
Windows](https://download.docker.com/win/stable/Docker%20Desktop%20Installer.exe)



#### Docker Hub

Cloud-based application registry and development team collaboration services.

[Signup](https://hub.docker.com/signup)

Topic 1 :: Introduction & Installation

1. Installation & configuration of dockers

--> explain the default folder structure

--> docker hub & its usage

--> image & its defaults with tags

Commands:Installation on Linux

yum install docker -y

systemctl enable docker

systemctl restart docker

Topic 2:: Working with containers

1. pulling centos image: centos & centos:6.9

#docker pull centos

#docker pull centos:6.9

2. Starting the first container using centos & explain in detail about the magic happened

options : -i,-t & usecases

# docker run -it centos /bin/bash

3. Running container in backend using "-d" option && Working with Multiple Images

4. Inspecting the container.

#docker inspect <containerid>

5. Attaching and executing commands inside the container

#docker run -ti centos /bin/bash

note: ctrl+p to comeout of container with out killing it

#docker exec -it <containerid> /bin/bash -c "uptime;df -hT"

6. Create two containers with test.txt file & explain in detail about the container architecture

Topic 3: Building our own images or Image customization.

1. Explain docker image build using "commit" & via Dockerfile

#docker commit <containerid> <imagename>

#docker build -t "imagename" .

For explainig this concept you can install package telnet & create user called devops.

Dockerfile :

FROM centos

RUN yum install telnet -y && useradd devops

Topic 4: Dockerfile References

exec

# docker exec -it <containerid> /bin/bash -c "uptime"

RUN

ENV

ADD

COPY

USER

PORTFORWARD

EXPOSE

VOLUME(Storage management)

CMD

ENTRYPOINT

Example ::

FROM centos:7

RUN yum install telnet nginx -y && useradd test1

ENV myenv=100

ADD tmp.tar /tmp/

COPY tmp.tar /tmp/

EXPOSE 80

VOLUME ["/usr/share/nginx/html"]

CMD ["nginx", "-g", "daemon off;"]

Execution:

docker run -d --name mynginx -v /mydata:/usr/share/nginx/html/ -p 80:80 mynginx:v1

CMD vs ENTRYPOINT ::

CMD:

This helps you to overwrite the base command declared inside the image

ENTRYPOINT:

You cannot over write the entrypoint once you created the image

CMD Execution ::

[root@dockers ~]# cat script1.sh

#!/bin/bash

echo "Hi I am Script1"

[root@dockers ~]# cat script2.sh

#!/bin/bash

echo "Hi I am script2"

[root@dockers ~]#

[root@dockers ~]# cat Dockerfile

FROM centos:7

COPY script1.sh script2.sh /tmp/

CMD ["/bin/bash","/tmp/script1.sh"]

Execution :

docker build -t "cmd:v1" .

[root@dockers ~]# docker run cmd:v1

Hi I am Script1

[root@dockers ~]#

[root@dockers ~]# docker run cmd:v1 /bin/bash /tmp/script2.sh

Hi I am script2

[root@dockers ~]#

ENTRYPOINT Execution ::

[root@dockers ~]# docker build -t "entry:v1" .

Sending build context to Docker daemon 73.22kB

Step 1/3 : FROM centos:7

---> 5e35e350aded

Step 2/3 : COPY script1.sh script2.sh /tmp/

---> Using cache

---> 5f974cbdbe98

Step 3/3 : ENTRYPOINT ["/bin/bash","/tmp/script1.sh"]

---> Running in 07caa0e525ed

Removing intermediate container 07caa0e525ed

---> cf9341cc8594

Successfully built cf9341cc8594

Successfully tagged entry:v1

[root@dockers ~]# docker run entry:v1

Hi I am Script1

[root@dockers ~]# docker run entry:v1 /bin/bash /tmp/script2.sh

Hi I am Script1

Topic 5: Storage Management

Storage Overview:

Data is classified based up on our need, it means temporary data & perminent data.

Ephemeral represents short time data. Basically containers will get the temporary volumes binded with them when we create them.

If we are storing any data inside the container that data will stay with it till its next restart. which means if its getting restarted

then the entire data is lost. Inorder to overcome this situation we are having storage concept which in deed helps us to attach a volume to the

container and it act like perminent storage in simple we call it like persistant volume.

Categories of data storage:

Non-persistent

Local storage

Data that is ephemeral (short term data)

Every container has it

Tied to the lifecycle of the contain

Persistent

Volumes

Volumes are decoupled from containers

Non-persistent Data ::

By default all the containers uses local storage. Default location for the containers are as follows.

Storage locations:

Linux: /var/lib/docker/[STORAGE-DRIVER]/

Windows: C:\ProgramData\Docker\windowsfilter\

In each and every OS there will be default storage drivers which helps contaniers for their consumption.

Storage Drivers:

RHEL uses overlay2.

Ubuntu uses overlay2 or aufs.

SUSE uses btrfs.

Windows uses its own.

Persistent Data Using Volumes ::

When ever we are using volumes with containers it will do the below.

-> creates the volume first and then creates the container.

-> mounts it to a directory inside the container

-> if container have data inside that directory it will be overwritten

-> deleting a container does not delete the volume

-> We have different drivers available in the market some of them are

Block storage --> EBS

File storage -> EFS,NFS

Object storage -> Amazon Simple Storage Service (S3),Google Cloud Storage,Azure Blob Storage

Volume Commands:

docker volume ls

docker volume create myvol1

docker volume inspect myvol1

Create the volume myvol1 and attach it to the nginx container.

hint: /usr/share/nginx/html is the default directory for nginx.

Explain diff between bind mounts & volume mounts.

Topic 6: Network Management

1. Explain in detail about the ifconfig output & network command of docker

2. Explain about the man page of docker

man docker-network-create

3. Create one adapter with the below configuration.

gateway 10.1.0.1

ip-range=10.1.4.0/24

subnet 10.1.0.0/16

4. Now lets create one container with this adapter

docker run -idt --name mynet1 --net devops --ip 10.1.4.5 centos /bin/bash

5. Creating one container with ipaddress.

docker run -idt --name mynet1 --net devops --ip 10.1.4.5 centos /bin/bash

1. Delete the pods, & network adapter created.

# Install Docker Desktop on Windows

Docker Desktop for Windows is the [Community](https://www.docker.com/community-edition) version of Docker for Microsoft Windows. You can download Docker Desktop for Windows from Docker Hub.

[Download from Docker Hub](https://hub.docker.com/editions/community/docker-ce-desktop-windows/)

By downloading Docker Desktop, you agree to the terms of the [Docker Software End User License Agreement](https://www.docker.com/legal/docker-software-end-user-license-agreement" \t "_blank) and the [Docker Data Processing Agreement](https://www.docker.com/legal/data-processing-agreement" \t "_blank).

## What to know before you install

### System Requirements

* Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later).
* Hyper-V and Containers Windows features must be enabled.
* The following hardware prerequisites are required to successfully run Client Hyper-V on Windows 10:
  + 64 bit processor with [Second Level Address Translation (SLAT)](http://en.wikipedia.org/wiki/Second_Level_Address_Translation)
  + 4GB system RAM
  + BIOS-level hardware virtualization support must be enabled in the BIOS settings. For more information, see [Virtualization](https://docs.docker.com/docker-for-windows/troubleshoot/#virtualization-must-be-enabled).

**Note:** Docker supports Docker Desktop on Windows based on Microsoft’s support lifecycle for Windows 10 operating system. For more information, see the [Windows lifecycle fact sheet](https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet).

.

### About Windows containers

Looking for information on using Windows containers?

* [Switch between Windows and Linux containers](https://docs.docker.com/docker-for-windows/#switch-between-windows-and-linux-containers) describes how you can toggle between Linux and Windows containers in Docker Desktop and points you to the tutorial mentioned above.
* [Getting Started with Windows Containers (Lab)](https://github.com/docker/labs/blob/master/windows/windows-containers/README.md) provides a tutorial on how to set up and run Windows containers on Windows 10, Windows Server 2016 and Windows Server 2019. It shows you how to use a MusicStore application with Windows containers.
* Docker Container Platform for Windows [articles and blog posts](https://www.docker.com/microsoft/) on the Docker website.

## Install Docker Desktop on Windows

1. Double-click **Docker Desktop Installer.exe** to run the installer.

If you haven’t already downloaded the installer (Docker Desktop Installer.exe), you can get it from **[Docker Hub](https://hub.docker.com/editions/community/docker-ce-desktop-windows/)**. It typically downloads to your Downloads folder, or you can run it from the recent downloads bar at the bottom of your web browser.

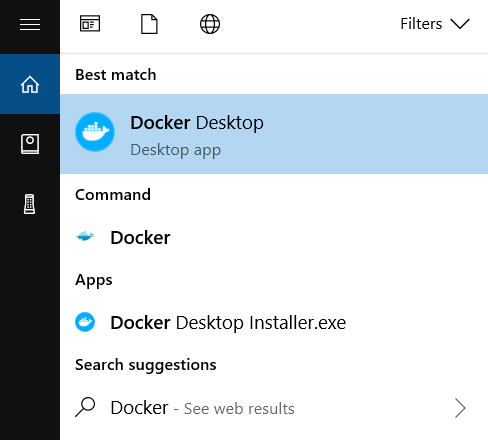
1. Follow the instructions on the installation wizard to accept the license, authorize the installer, and proceed with the install.

When prompted, authorize the Docker Desktop Installer with your system password during the install process. Privileged access is needed to install networking components, links to the Docker apps, and manage the Hyper-V VMs.

1. Click **Finish** on the setup complete dialog and launch the Docker Desktop application.

## Start Docker Desktop

Docker Desktop does not start automatically after installation. To start Docker Desktop, search for Docker, and select **Docker Desktop** in the search results.

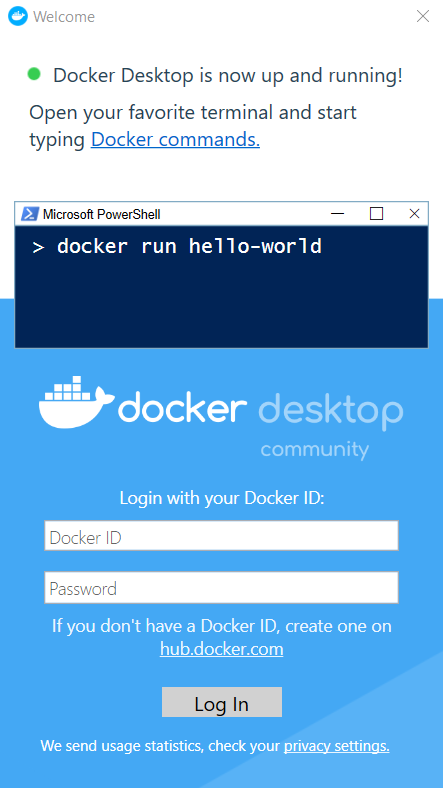


When the whale icon in the status bar stays steady, Docker Desktop is up-and-running, and is accessible from any terminal window.

whale on taskbar

If the whale icon is hidden in the Notifications area, click the up arrow on the taskbar to show it. To learn more, see [Docker Settings](https://docs.docker.com/docker-for-windows/" \l "docker-settings-dialog).

After installing the Docker Desktop app, you also get a pop-up success message with suggested next steps, and a link to this documentation.



When initialization is complete, click the whale icon in the Notifications area and select **About Docker Desktop** to verify that you have the latest version.

Congratulations! You are successfully running Docker Desktop on Windows.

## Uninstall Docker Desktop

To uninstall Docker Desktop from your Windows machine:

1. From the Windows **Start** menu, select **Settings** > **Apps** > **Apps & features**.
2. Select **Docker Desktop** from the **Apps & features** list and then select **Uninstall**.
3. Click **Uninstall** to confirm your selection.

**Note:** Uninstalling Docker Desktop will destroy Docker containers and images local to the machine and remove the files generated by the application.