Install Docker

Prerequisites

1. **OS requirements**

To install Docker Engine, you need the `64-bit` version of one of these Ubuntu versions:

- Ubuntu Eoan 19.10
- Ubuntu Bionic 18.04 (LTS)
- Ubuntu Xenial 16.04 (LTS)

Docker Engine is supported on `x86_64` (or `amd64`), `armhf`, `arm64`, `s390x` (IBM Z), and `ppc64le` (IBM Power) architectures.

2. **Uninstall old versions**

Older versions of Docker were called `docker`, `docker.io`, or `docker-engine`. If these are installed, uninstall them:

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

It's OK if `apt-get` reports that none of these packages are installed.

The contents of `/var/lib/docker/`, including images, containers, volumes, and networks, are preserved. The Docker Engine package is now called `docker-ce`.

Install using the repository

SET UP THE REPOSITORY

1. Update the `apt` package index and install packages to allow `apt` to use a repository over HTTPS:

```
$ sudo apt-get update

$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

2. Add Docker's official GPG key:

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

Verify that you now have the key with the fingerprint `9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88`, by searching for the last 8 characters of the fingerprint.

```
$ sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

3. Set up the stable repository `x86_64/amd64`

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

INSTALL DOCKER ENGINE

1. Update the `apt` package index, and install the *latest version* of Docker Engine and containerd, or go to the next step to install a specific version:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

> Got multiple Docker repositories?
>
> If you have multiple Docker repositories enabled, installing or updating without specifying a version in the `apt-get install` or `apt-get update` command always installs the highest possible version, which may not be appropriate for your stability needs.

2. To install a *specific version* of Docker Engine, list the available versions in the repo, then select and install:

   a. List the versions available in your repo:

```
$ apt-cache madison docker-ce

  docker-ce | 5:18.09.1~3-0~ubuntu-xenial |
https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
  docker-ce | 5:18.09.0~3-0~ubuntu-xenial |
https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
  docker-ce | 18.06.1~ce~3-0~ubuntu       |
https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
  docker-ce | 18.06.0~ce~3-0~ubuntu       |
https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
  ...
```

b. Install a specific version using the version string from the second column, for example, `5:18.09.1~3-0~ubuntu-xenial`.

```
$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=
<VERSION_STRING> containerd.io
```

Uninstall Docker Engine

1. Uninstall the Docker Engine, CLI, and Containerd packages:

   ```
   $ sudo apt-get purge docker-ce docker-ce-cli containerd.io
   ```

2. Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes:

   ```
   $ sudo rm -rf /var/lib/docker
   ```

You must delete any edited configuration files manually.

Upgrade Docker Engine

To upgrade Docker Engine, first run `sudo apt-get update`, then follow the installation instructions, choosing the new version you want to install.

Config docker and set proxy

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user `root` and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

If you don't want to preface the `docker` command with `sudo`, create a Unix group called `docker` and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group.

To create the `docker` group and add your user:

1. Create the `docker` group.

   ```
   $ sudo groupadd docker
   ```

2. Add your user to the `docker` group.

   ```
   $ sudo usermod -aG docker $USER
   ```

3. Log out and log back in so that your group membership is re-evaluated.

   If testing on a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

   On a desktop Linux environment such as X Windows, log out of your session completely and then log back in.

   On Linux, you can also run the following command to activate the changes to groups:

   ```
   $ newgrp docker
   ```

4. Verify that you can run `docker` commands without `sudo`.

   ```
   $ docker run hello-world
   ```

   This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

   > If you initially ran Docker CLI commands using `sudo` before adding your user to the `docker` group, you may see the following error, which indicates that your `~/.docker/` directory was created with incorrect permissions due to the `sudo` commands.
   >
   > ```
   > WARNING: Error loading config file: /home/user/.docker/config.json -
   > stat /home/user/.docker/config.json: permission denied
   > ```
   >
   > To fix this problem, either remove the `~/.docker/` directory (it is recreated automatically, but any custom settings are lost), or change its ownership and permissions using the following commands:
   >
   > ```
   > $ sudo chown "$USER":"$USER" /home/"$USER"/.docker -R
   > $ sudo chmod g+rwx "$HOME/.docker" -R
   > ```

HTTP/HTTPS proxy

The following two kinds of proxy is necessary for those who working behind corporate proxy.

docker-container-proxy

1. On the Docker client, create or edit the file `~/.docker/config.json` in the home directory of the user which starts containers.

   > If there is a `config.json` , back up it.
   >
   > ```
   > cp ~/.docker/config.json ~/.docker/config.bk
   > ```

   Add JSON such as the following.

   ```
   {
    "proxies":
    {
      "default":
      {
        "httpProxy": "http://child-prc.intel.com:913/",
        "httpsProxy": "http://child-prc.intel.com:913/",

   "noProxy":"localhost,.intel.com,,127.0.0.0/8,172.16.0.0/20,192.168.0.0/16,10
   .0.0.0/8"

      }
    }
   }
   ```

2. restart docker

   ```
   systemctl restart docker
   ```

3. When you create or start new containers, the environment variables are set automatically within the container.

docker-daemon-proxy

If you are behind an HTTP or HTTPS proxy server, for example in corporate settings, you need to add this configuration in the Docker systemd service file.

1. Create a systemd drop-in directory for the docker service:

   ```
   $ sudo mkdir -p /etc/systemd/system/docker.service.d
   ```

2. Create a file called `/etc/systemd/system/docker.service.d/http-proxy.conf` that adds the `HTTP_PROXY` environment variable:

   ```
   [Service]
   Environment="HTTP_PROXY=http://child-prc.intel.com:913/"
   "NO_PROXY=localhost,.intel.com,,127.0.0.0/8,172.16.0.0/20,192.168.0.0/16,10.
   0.0.0/8"
   ```

3. **Or**, if you are behind an HTTPS proxy server, Create a file called `/etc/systemd/system/docker.service.d/https-proxy.conf` that adds the `HTTPS_PROXY` environment variable:

```
[Service]
Environment="HTTP_PROXY=http://child-prc.intel.com:913/"
"NO_PROXY=localhost,.intel.com,,127.0.0.0/8,172.16.0.0/20,192.168.0.0/16,10.
0.0.0/8"
```

4. Flush changes:

```
$ sudo systemctl daemon-reload
```

5. Restart Docker:

```
$ sudo systemctl restart docker
```

6. Verify that the configuration has been loaded:

```
$ systemctl show --property=Environment docker
```

Docker Base Command

You can get details here.

Service

- Start docker service

```
systemctl start docker
```

- Stop docker service

```
systemctl stop docker
```

- Restart docker service

```
systemctl restart docker
```

Container

- Run a command in a new container

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

The `docker run` command first `creates` a writeable layer over the specified image, and then `starts` it using the specified command. That is, `docker run` is equivalent to the API `/containers/create` then `/containers/(id)/start`. A stopped container can be restarted with all its previous changes intact using `docker start` .

Options

Name, shorthand	Description
--interactive, -i	Keep STDIN open even if not attached
--tty, -t	Allocates a `pseudo-TTy`
--name	Assign a name to the container
--volume, -v	Bind mount a volume
--net	Connect a container to a network
--gpus	GPU devices to add to the container(`all` to pass all GPUs)
--device	Add a host device to the container

- List containers

  ```
  docker ps [OPTIONS]
  ```

  Options

Name, shorthand	Description
--all, -i	Show all containers(default shows just running)
--quiet, -q	Only display numeric IDs

  Examples

  The `docker ps` command only shows running containers by default. To see all containers, use `-a` flag.

  ```
  docker ps -a
  ```

- Stop one or more **running** containers

  ```
  docker stop [OPTIONS] CONTAINER [CONTAINER...]
  ```

  CONTAINER can be `name` or `ID`

- Start one or more **stopped** containers

  ```
  docker start [OPTIONS] CONTAINER [CONTAINER...]
  ```

- Remove one or more **stopped** containers

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

Options

Name, shorthand	Description
--force, -f	Force the removal of a **running container**(uses SIGKILL)

Examples

- Remove **all stopped** containers

  ```
  docker rm $(docker ps -aq)
  ```

  The command `docker ps -aq` above returns all existing container IDs and passes them to the `rm` command which deletes them.

- Run a command in a running container.

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Examples

First, start a container.

```
$ docker run -it --name ubuntu_test ubuntu bash
```

This will create a command named `ubuntu_test` and start a Bash session.

Next, execute a command on the container.

```
$ docker exec -d ubuntu_test touch /tmp/execWorks
```

This will create a new file `/tmp/execWorks` inside the running container `ubuntu_test`, in the background(`-d` stands for `detach` which means `run command in the background`)

You can select working directory(default is `/`) for the command to execute into

```
$ docker exec -it -w /root ubuntu_test pwd
/root
```

- Exit a container

  1. When you want to exit a container, you can type `exit`, and the status of this container will be `Exited`.You can enter an `Exited` container by the following commands:

First, start an `Exited` container

```
$ docker start CONTAINER  # start an Exited container
```

Second, enter the `running` container

  - ```
    $ docker attach CONTAINER
    ```

  - Or, you can enter it by `exec`
    ```

```
$ docker exec -it CONTAINER /bin/bash
```

If you enter a container by `exec`, when you type `exit` to exit it, the container's status won't be `Exited`.

2. You can type `Ctrl + p q` to exit the container, and in this way, the status of container won't be `Eixted`

Image

- List images

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Options

Name, shorthand	Description
--all, -a	Show all images(default hides intermediate images)
--quiet, -q	Only show numeric IDs

- Remove one or more images

```
docker rmi [OPTIONS] IAMGE [IMAGE...]
```

- Search the Docker Hub for images

```
docker search [OPTIONS] TERM
```

- Pull an image or a repository from a registry

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Dockerfile

- Build an image from a Dockerfile(The name of Dockerfile can be anything, such as 'hello')

```
docker build -f ./dockerfile -t demo:01 .
```

The above command will use the current directory as the build `context`.

Options

Name, shorthand	Description
--file, -f	Name of the Dockerfile(Default is 'PATH/Dockerfile')
--tag, -t	Name and optionally a tag in the 'name:tag' forma

Enable X server in container(to use GUI)

The magic word in the X window system is **DISPLAY**. A DISPLAY consists(simplified) of:

- a keyboard
- a mouse
- and a screen

A DISPLAY is managed by a server program, known as an `X server` . The server serves displaying capabilities to other programs that connect to it.

The remote server knows where it have to redirect the X network traffic via the definition of the DISPLAY environment variable which generally points to an X Display server located on your local computer.

The form of DISPLAY is:

```
hostname:displaynumber.screennumber
```

The phrase `display` is usually used to refer to collection of monitors that share a common keyboard and pointer(mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, **multi-user systems**, hower, frequently hvave several displays so that more than one person can be doing graphics work at once. To avoid confusion, **each display on a machine is assigned a display number**(beginning at 0) when the X server for that display is started.

In your host

1. `echo $DISPLAY` and remeber displaynumber.
2. `xauth list` will show **keys**, copy the line which has the **displaynumber**

for example：

```
$ echo $DISPLAY
localhost:10.0 # displaynumber is 10
$ xauth list
libo7x-1080/unix:16  MIT-MAGIC-COOKIE-1  5e3cdb10571f9788a1fad8e2867528d8
libo7x-1080/unix:15  MIT-MAGIC-COOKIE-1  30c42520d12c217bce1af72fa98b7103
libo7x-1080/unix:17  MIT-MAGIC-COOKIE-1  e15a8826c215d38674cba298a939c21a
libo7x-1080/unix:13  MIT-MAGIC-COOKIE-1  4bb2729857abeb7d53bf05f0ca8f9979
libo7x-1080/unix:12  MIT-MAGIC-COOKIE-1  2845213a622e554ef1ffb878dca19a8b
libo7x-1080/unix:14  MIT-MAGIC-COOKIE-1  6222f0a07a79266065ed854e6594b2bf
libo7x-1080/unix:11  MIT-MAGIC-COOKIE-1  857e702481b85e7ccb07d46d8c832886
libo7x-1080/unix:10  MIT-MAGIC-COOKIE-1  ec4a34baf9002c1b04edaecbf41c37ad
# copy this line (key):
# libo7x-1080/unix:10  MIT-MAGIC-COOKIE-1  ec4a34baf9002c1b04edaecbf41c37ad
```

In your container.

If you add `-e DISPLAY` in `docker run` command, the container will have the same DISPLAY as host.

Add the authorization information to your display:

```
xauth add key # Paste the content you copied earlier in "key"
```

From now on, we can use `X server` in this container. You can change your display by modify $DISPLAY, and `xauth add` the corresponding key.

Reference

Install docker engine on Ubuntu

Manage docker as a non-root user

Docker-container-proxy

Docker-daemon-proxy

Docker-dns

Docker CLI

Glossary

Term	Definition
base image	A **base image** has no parent image specified in its Dockerfile. It is created using a Dockerfile with the `FROM scratch` directive.
build	build is the process of building Docker images using a Dockerfile. The build uses a Dockerfile and a "context". The context is the set of files in the directory in which the image is built.
Compose	Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.*Also known as : docker-compose, fig*
copy-on-write	Docker uses a copy-on-write technique and a union file system for both images and containers to optimize resources and speed performance. Multiple copies of an entity share the same instance and each one makes only specific changes to its unique layer.Multiple containers can share access to the same image, and make container-specific changes on a writable layer which is deleted when the container is removed. This speeds up container start times and performance.Images are essentially layers of filesystems typically predicated on a base image under a writable layer, and built up with layers of differences from the base image. This minimizes the footprint of the image and enables shared development.For more about copy-on-write in the context of Docker, see Understand images, containers, and storage drivers.
container	A container is a runtime instance of a docker image.A Docker container consists ofA Docker imageAn execution environmentA standard set of instructionsThe concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.
Docker	The term Docker can refer toThe Docker project as a whole, which is a platform for developers and sysadmins to develop, ship, and run applicationsThe docker daemon process running on the host which manages images and containers (also called Docker Engine)
Docker Hub	The Docker Hub is a centralized resource for working with Docker and its components. It provides the following services:Docker image hostingUser authenticationAutomated image builds and work-flow tools such as build triggers and web hooksIntegration with GitHub and Bitbucket
Dockerfile	A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile.

Term	Definition
ENTRYPOINT	In a Dockerfile, an `ENTRYPOINT` is an optional definition for the first part of the command to be run. If you want your Dockerfile to be runnable without specifying additional arguments to the `docker run` command, you must specify either `ENTRYPOINT`, `CMD`, or both.If `ENTRYPOINT` is specified, it is set to a single command. Most official Docker images have an `ENTRYPOINT` of `/bin/sh` or `/bin/bash`. Even if you do not specify `ENTRYPOINT`, you may inherit it from the base image that you specify using the `FROM` keyword in your Dockerfile. To override the `ENTRYPOINT` at runtime, you can use `--entrypoint`. The following example overrides the entrypoint to be `/bin/ls` and sets the `CMD` to `-l /tmp`.`$ docker run --entrypoint=/bin/ls ubuntu -l /tmp ``CMD` is appended to the `ENTRYPOINT`. The `CMD` can be any arbitrary string that is valid in terms of the `ENTRYPOINT`, which allows you to pass multiple commands or flags at once. To override the `CMD` at runtime, just add it after the container name or ID. In the following example, the `CMD` is overridden to be `/bin/ls -l /tmp`.`$ docker run ubuntu /bin/ls -l /tmp`In practice, `ENTRYPOINT` is not often overridden. However, specifying the `ENTRYPOINT` can make your images more flexible and easier to reuse.
image	Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.
layer	In an image, a layer is modification to the image, represented by an instruction in the Dockerfile. Layers are applied in sequence to the base image to create the final image. When an image is updated or rebuilt, only layers that change need to be updated, and unchanged layers are cached locally. This is part of why Docker images are so fast and lightweight. The sizes of each layer add up to equal the size of the final image.
Machine	Machine is a Docker tool which makes it really easy to create Docker hosts on your computer, on cloud providers and inside your own data center. It creates servers, installs Docker on them, then configures the Docker client to talk to them.*Also known as : docker-machine*
parent image	An image's **parent image** is the image designated in the `FROM` directive in the image's Dockerfile. All subsequent commands are based on this parent image. A Dockerfile with the `FROM scratch` directive uses no parent image, and creates a **base image**.
persistent storage	Persistent storage or volume storage provides a way for a user to add a persistent layer to the running container's file system. This persistent layer could live on the container host or an external device. The lifecycle of this persistent layer is not connected to the lifecycle of the container, allowing a user to retain state.
registry	A Registry is a hosted service containing repositories of images which responds to the Registry API.The default registry can be accessed using a browser at Docker Hub or using the `docker search` command.

Term	Definition
repository	A repository is a set of Docker images. A repository can be shared by pushing it to a registry server. The different images in the repository can be labeled using tags.Here is an example of the shared nginx repository and its tags.
tag	A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other.*Note : This label is not related to the key=value labels set for docker daemon.*
virtual machine	A virtual machine is a program that emulates a complete computer and imitates dedicated hardware. It shares physical hardware resources with other users but isolates the operating system. The end user has the same experience on a Virtual Machine as they would have on dedicated hardware.Compared to containers, a virtual machine is heavier to run, provides more isolation, gets its own set of resources and does minimal sharing.*Also known as : VM*
volume	A volume is a specially-designated directory within one or more containers that bypasses the Union File System. Volumes are designed to persist data, independent of the container's life cycle. Docker therefore never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container. *Also known as: data volume*There are three types of volumes: *host, anonymous, and named*:A **host volume** lives on the Docker host's filesystem and can be accessed from within the container.A **named volume** is a volume which Docker manages where on disk the volume is created, but it is given a name.An **anonymous volume** is similar to a named volume, however, it can be difficult, to refer to the same volume over time when it is an anonymous volumes. Docker handle where the files are stored.