

# Practical: Getting Started with Docker

## Table of Contents


- [Prerequisites and References](#)
- [Additional resources](#)
- [Installing and using Docker](#)
  - [Step 1: Install docker.](#)
  - [Step 2: Test run hello-world example.](#)
  - [Step 3: Remove the container](#)
  - [Step 4: Download Ubuntu image](#)
  - [Step 5: Environment variables at the CLI](#)
  - [Step 6: Environment variables with a file](#)

This activity will get you started with running Docker

Docker is an open 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.

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your server 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.

Source: [Docker Documentation](https://docs.docker.com/get-started/overview/)  (<https://docs.docker.com/get-started/overview/>)

Please continue reading the Docker overview page in the official documentation, [here](https://docs.docker.com/get-started/overview/)  (<https://docs.docker.com/get-started/overview/>).

Docker is available in two guises:

1. For enterprise customers (paid)
2. For customers in the community (free; we will use this one)

In this practical, we will work with Docker under Linux. Specifically, in an EC2 instance as created earlier.

If you need to launch a new instance, follow the earlier practicals, starting at [this one](https://canvas.qut.edu.au/courses/20367/pages/practical-creating-an-ec2-instance/) (<https://canvas.qut.edu.au/courses/20367/pages/practical-creating-an-ec2-instance/>).



You may also run Docker on Windows or Mac OS. Please note, if you are using an older version Windows, Docker may not be compatible. Please see the instructions and links in the appendix for details of these alternatives. Unfortunately, we cannot directly support every operating system. This practical assumes that your virtual machine is running a recent version of Ubuntu.

Note: Some screenshots may have minor inconsistencies due to command updates (Python 3) and recent EC2 updates from AWS. If you have any problems, please do not hesitate to contact the teaching team or consult your peers.



The basics of using Docker are covered very nicely in Docker's own tutorials. The full tutorial will take about 30 minutes if things go according to plan.

## Prerequisites and References

In this unit, we will work mainly with the Linux version of Docker, and the supported version will be **Ubuntu LTS 24.04**. It is a requirement of this prac exercise, and assignment 1, that you are able to deploy your Docker container to the public cloud. At some point, you will need to launch a VM on AWS. Preparation of the Docker image can take place anywhere you can access a tame Linux workstation.

- How to access to a Linux system:
  - **This approach is used in this practical:**
    - Use one Linux VM to prepare the image
    - Use another VM for deployment of the container after we have pushed it to AWS Elastic Container Registry.
  - Using personal Linux machines (i.e., Linux installed as the host operating system)
  - Docker under Windows Subsystem for Linux
    - [Visual Studio documentation](#)  <https://code.visualstudio.com/blogs/2020/03/02/docker-in-wsl2>
    - Note: this does not work well on the QUT lab machines
  - Using Virtual Box <https://www.virtualbox.org/>  <https://www.virtualbox.org/>
    - Install Linux as a guest operating system
    - or an alternative VM manager.
    - As discussed, running a virtual machine is expensive, in terms of hardware resources.
    - This approach has the lowest priority.

## Additional resources

- [Docker Reference](#)  <https://docs.docker.com/>
- [Docker Getting Started](#)  <https://docs.docker.com/get-started/>.
  - This is particularly good on the terminology and concepts.
    - For example, you should be familiar with the distinction between the container and the image.

# Installing and using Docker


Docker is fundamentally a system built for use under Linux, and you should experience few problems installing under a native Linux environment.

We will begin by installing Docker, and running a basic sanity check deployment of the "hello-world" image.

To install new software, we would normally use the `apt-get` application, but here there are more convenient alternatives.

If you haven't already, connect to an AWS EC2 instance.

## Step 1: Install docker.

Docker provide a user guide for installing Docker Engine on Ubuntu, [here](https://docs.docker.com/engine/install/ubuntu/)  (<https://docs.docker.com/engine/install/ubuntu/>). We will draw inspiration from that guide in our instructions. Please consider reading both the user guide and our instructions. We will follow "Install using the convenience script".

If you are using SSM (rather than ssh) to connect, you will need to change to the `ubuntu` user, like so

```
sudo -iu ubuntu
```

Make sure your current working directory is the home directory of your logged-in user.

```
cd ~
```

We will install Docker using a shell script that can obtained from docker.com

By default, your machine should have curl installed.

Use `sudo` to ensure that the installation scripts have elevated privileges.

```
curl -fsSL https://get.docker.com -o get-docker.sh  
sudo sh ./get-docker.sh
```

Next we need to add the `ubuntu` user to the `docker` group to give it the privileges necessary to run docker commands. We also need to re-login as `ubuntu` so that the `docker` group becomes effective. The following commands do these two tasks:

```
sudo adduser ubuntu docker  
sudo su -l ubuntu
```

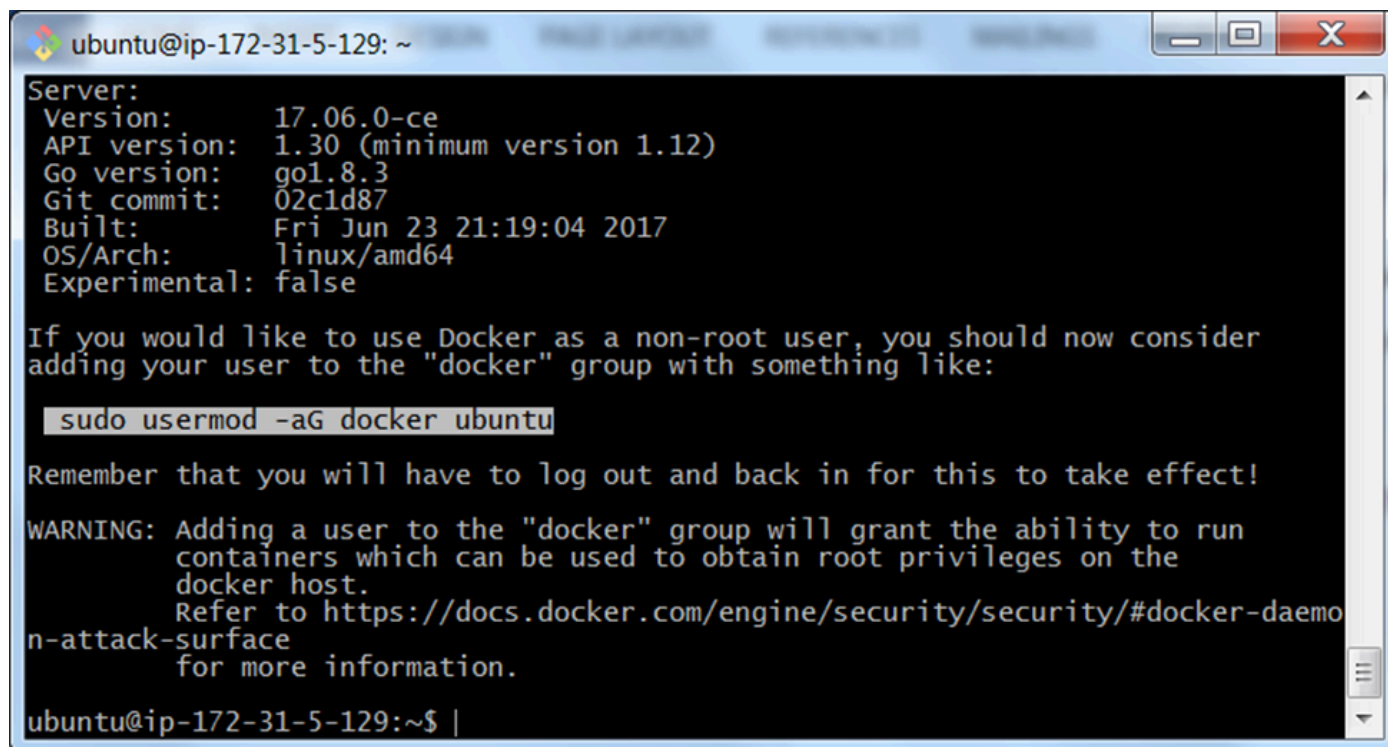
Once the Docker installation has completed, you should run a basic command to confirm that everything is running as expected.

This command should print a small amount of information:

```
docker --version
```

This command will print more detailed information:

```
docker info
```



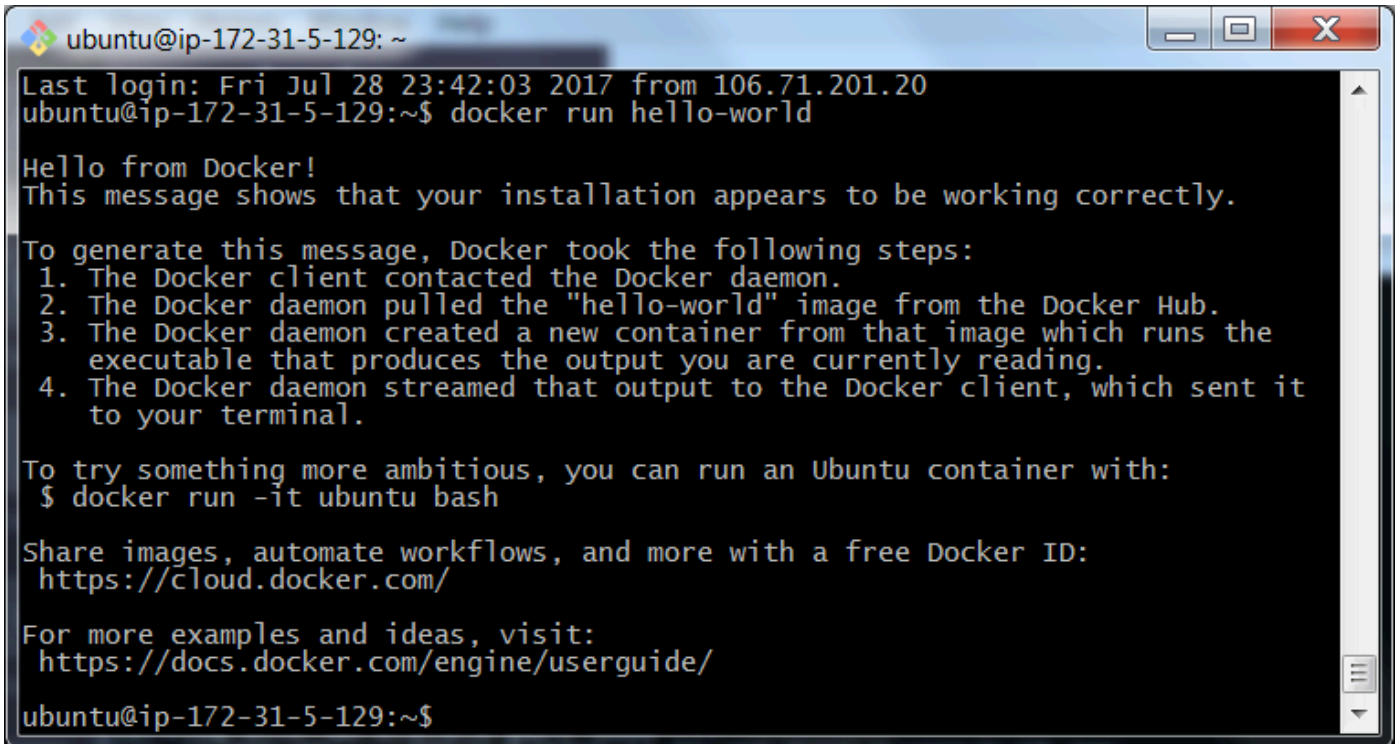
```
ubuntu@ip-172-31-5-129: ~  
Server:  
Version:      17.06.0-ce  
API version:  1.30 (minimum version 1.12)  
Go version:   go1.8.3  
Git commit:   02c1d87  
Built:        Fri Jun 23 21:19:04 2017  
OS/Arch:      linux/amd64  
Experimental: false  
  
If you would like to use Docker as a non-root user, you should now consider  
adding your user to the "docker" group with something like:  
  
  sudo usermod -aG docker ubuntu  
  
Remember that you will have to log out and back in for this to take effect!  
  
WARNING: Adding a user to the "docker" group will grant the ability to run  
containers which can be used to obtain root privileges on the  
docker host.  
Refer to https://docs.docker.com/engine/security/security/#docker-daemo  
n-attack-surface  
for more information.  
  
ubuntu@ip-172-31-5-129:~$ |
```

## Step 2: Test run `hello-world` example.

Let's run the code below. This should run without issue. Try running the `hello-world` example without using `sudo`:

```
docker run hello-world
```

You should see something like this, with an additional message if the image was not available locally and had to be pulled down from the web.

A terminal window titled 'ubuntu@ip-172-31-5-129: ~' with standard window controls. The terminal shows the output of 'docker run hello-world'. It displays the last login time, the command executed, and a 'Hello from Docker!' message. It then lists four steps Docker took to generate the message: contacting the daemon, pulling the 'hello-world' image, creating a new container, and streaming the output. Finally, it provides instructions on how to run an Ubuntu container and links to Docker documentation.

```
ubuntu@ip-172-31-5-129: ~  
Last login: Fri Jul 28 23:42:03 2017 from 106.71.201.20  
ubuntu@ip-172-31-5-129:~$ docker run hello-world  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://cloud.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/engine/userguide/  
ubuntu@ip-172-31-5-129:~$
```

## Step 3: Remove the container

You just ran a container that we don't need again in the future. You should remove the container.

First, find the container by ID:

```
docker container ls -a
```

The `-a` flag instructs `ls` (list) to include all containers, rather than just those running.

Find the ID of the container running hello-world.

Run `rm` to remove the container. Be sure to specify the container ID. Do not include `<` and `>`.

```
docker container rm <id>
```

## Step 4: Download Ubuntu image

In preparation for the next steps, will obtain the `ubuntu` Docker image then run a bash shell within.

```
docker run -it --rm ubuntu bash
```

Here `-it` says that we want an interactive session with the container, `--rm` says to delete the container when it exits, `ubuntu` is the name of the image to run and `bash` (the usual Linux command shell) is the command that we want to run on the container.

```

root@22e06dea457d: /
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://cloud.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

ubuntu@ip-172-31-5-129:~$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
e0a742c2abfd: Pull complete
486cb8339a27: Pull complete
dc6f0d824617: Pull complete
4f7a5649a30e: Pull complete
672363445ad2: Pull complete
Digest: sha256:84c334414e2bfdcae99509a6add166bbb4fa4041dc3fa6af08046a66fed3005f
Status: Downloaded newer image for ubuntu:latest
root@22e06dea457d:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@22e06dea457d:/#

```

Note: the username and hostname of the prompt have changed from `ubuntu@ip` to the `root@22e...`, as shown at the bottom of the screen. We are now in the command interface of Ubuntu within the Docker container, which is sitting on top of Ubuntu on the AWS VM.

However, this is not immediately useful, given we already have a bash shell available to us in our Ubuntu-based EC2 instance. It is just a demonstration of running bash within a Docker container, which is running on top of Ubuntu. That is: Bash within Bash (to some extent).

You can type `exit` to exit bash within the container. Given the application in the container has exited, the container will exit too.

## Step 5: Environment variables at the CLI

Let's run the container with some environment variables passed in. First, we will do that at the command-line. Later, we will write a file that manages for us. Using a file is a better approach because it can be version controlled. Keeping track of commands issue in a terminal is difficult.

We will use the `-e` flag.

```
docker run -it -e unit=CAB432 --rm ubuntu bash -c "echo test"
```

That did nothing with the environment variable, but it is a stepping stone to that.

Try this:

```
docker run -it -e unit=CAB432 --rm ubuntu bash -c "printenv"
```

you should see `unit=CAB432`, indicating that the environment variable `unit` is set to `CAB432`. Programs in the container can read the environment variables, giving a way to pass parameters to the program.

## Step 6: Environment variables with a file

Let's try that again but rather than using the CLI `-e` flag, we will use a file of variables.

Create a file in your current working directory (on EC2), with the name `.env`.

```
unit=CAB432
name=Cloud Computing
uni=QUT
```

Run:

```
docker run -it -e unit=CAB432 --rm --env-file .env ubuntu bash -c "pr
intenv"
```

You should get something like this:

```
HOSTNAME=ea2155beb33c
uni=QUT
PWD=/
HOME=/root
unit=CAB432
TERM=xterm
SHLVL=0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
name=Cloud Computing
_=/usr/bin/printenv
```

This mechanism can be used to pass more complex parameters to a program running in a container without having to specify them all on the command line every time.