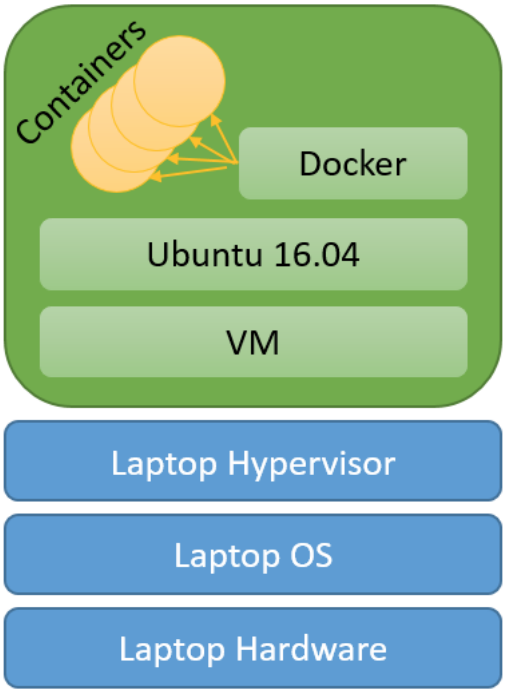
RX-M

Docker Containers

Lab 1 – Installing Docker

Docker is an open-source software system that automates the deployment of applications inside software containers. Containers abstract an application's operating environment from the underlying operating system. Containers also support the resource isolation features of the Linux kernel, such as cgroups and kernel namespaces, allowing many diverse containers to run within a single Linux instance without resource contention. Containers eliminate the overhead and startup latency of a full virtual machine while preserving most of the isolation benefits.

Docker can package an application and its dependencies into an image, which can be used to launch a container on any Linux system. This enables flexibility and portability, allowing applications to run reliably across a number of Linux distributions with various configurations in a range of cloud settings.



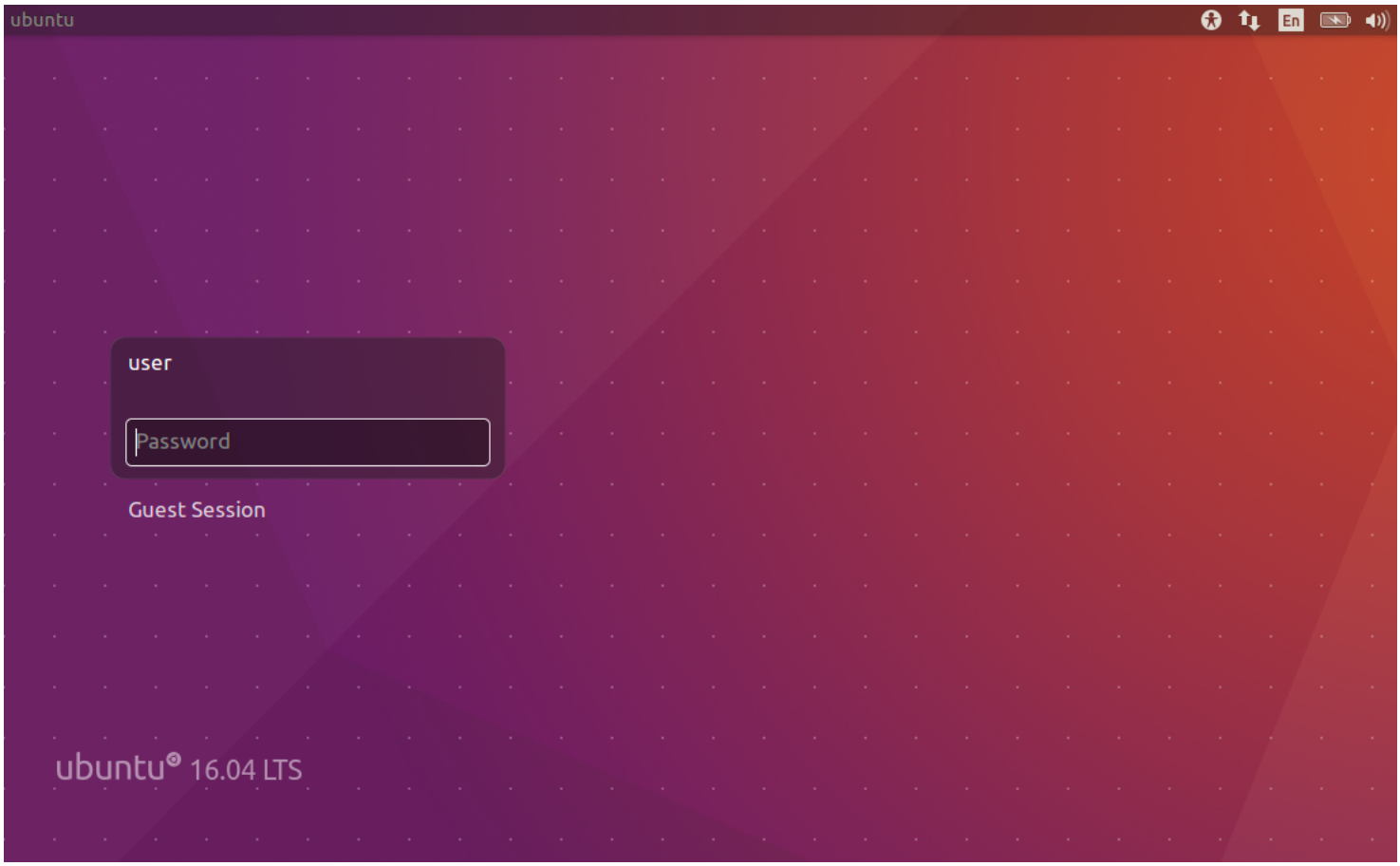
In this lab we will install and test Docker on the classroom supplied virtual lab machine [see model above]. The class room VM is an Ubuntu server with a basic desktop installed. Versions of the lab system are available for Virtual Box and VMware desktop hypervisors. If you do not have a copy of the lab VM, instructions for downloading it can be found here:

- https://s3-us-west-1.amazonaws.com/rx-m-class-setup/rx-m-lab-sys-setup.html
- N.B. If you cannot access the lab VM any base Ubuntu system with the same version will work

The Lab system user account is “user” and the password is “user”. This user has full sudo permissions.

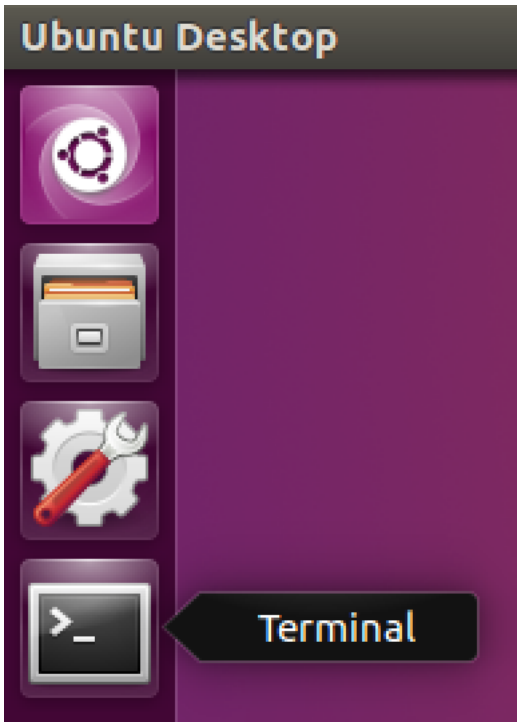
1. Explore Ubuntu built-in Docker support

Start your classroom virtual machine and log in as “user” with the password “user”.



There is no reason to update this lab system and doing so may require large downloads. If the VM prompts you to perform a system update, choose the “Remind me later” option to avoid tying up the class room Internet connection.

The Launch Pad on the left side of the desktop has short cuts to commonly used programs. Click the Terminal icon to launch a new Bash command line shell.



In the terminal window use the apt-cache command to display the Docker packages available:

```
user@ubuntu:~$ sudo apt-cache search docker

libcib-dev - cluster resource manager CIB library development
libcib4 - cluster resource manager CIB library
libcrmcluster-dev - cluster resource manager cluster library development
libcrmcluster4 - cluster resource manager cluster library
libcrmcommon-dev - cluster resource manager common library development
libcrmcommon3 - cluster resource manager common library
```

```
libcrmservice-dev - cluster resource manager service library development
libcrmservice3 - cluster resource manager service library
liblrmd-dev - cluster resource manager LRMD library development
liblrmd1 - cluster resource manager LRMD library
libpe-rules2 - cluster resource manager Policy Engine rules library
libpe-status10 - cluster resource manager Policy Engine status library
libpengine-dev - cluster resource manager Policy Engine library development
libpengine10 - cluster resource manager Policy Engine library
libstonithd-dev - cluster resource manager STONITH daemon library development
libstonithd2 - cluster resource manager STONITH daemon library
libtransitioner2 - cluster resource manager transitioner library
pacemaker - cluster resource manager
pacemaker-cli-utils - cluster resource manager command line utilities
pacemaker-common - cluster resource manager common files
pacemaker-doc - cluster resource manager HTML documentation
pacemaker-resource-agents - cluster resource manager general resource agents
python-magnumclient - client library for Magnum API - Python 2.x
python-magnumclient-doc - client library for Magnum API - doc
cadvisor - analyze resource usage and performance characteristics of running containers
ctop - Command line / text based Linux Containers monitoring tool
debocker - docker-powered package builder for Debian
debootstick - Turn a chroot environment into a bootable image
docker - System tray for KDE3/GNOME2 docklet applications
docker-compose - Punctual, lightweight development environments using Docker
docker-registry - Docker toolset to pack, ship, store, and deliver content
docker.io - Linux container runtime
golang-docker-dev - Transitional package for golang-github-docker-docker-dev
golang-github-docker-distribution-dev - Docker toolset to pack, ship, store, and deliver content (source)
golang-github-docker-docker-dev - Externally reusable Go packages included with Docker
golang-github-docker-go-units-dev - parse and print size and time units in human-readable format
golang-github-docker-libkv-dev - Key/Value store abstraction library
golang-github-docker-libtrust-dev - Primitives for identity and authorization
golang-github-docker-notary-dev - tool for running and interacting with trusted collections
golang-github-docker-spdystream-dev - multiplexed stream library using spdy
golang-github-endophage-gotuf-dev - Go implementation of the TUF specification
golang-github-fsouza-go-dockerclient-dev - Docker client library in Go
golang-github-google-cadvisor-dev - analyze resource usage and performance of running containers
golang-github-opencontainers-runc-dev - Open Container Project - develpoment files
golang-github-samalba-dockerclient-dev - Docker client library in Go
golang-github-vishvananda-netlink-dev - netlink library for go
gosu - Simple Go-based setuid+setgid+setgroups+exec
karbon - vector graphics application for the Calligra Suite
kdocker - lets you dock any application into the system tray
libnss-docker - nss module for finding Docker containers
magnum-api - OpenStack containers as a service
magnum-common - OpenStack containers as a service - API server
magnum-conductor - OpenStack containers as a service - conductor
needrestart - check which daemons need to be restarted after library upgrades
ovn-docker - OVN Docker drivers
pacemaker-remote - cluster resource manager proxy daemon for remote nodes
pidgin - graphical multi-protocol instant messaging client for X
python-docker - Python wrapper to access docker.io's control socket
python-dockerpty - Pseudo-tty handler for docker Python client (Python 2.x)
python-magnum - OpenStack containers as a service - Python library
python3-docker - Python 3 wrapper to access docker.io's control socket
python3-dockerpty - Pseudo-tty handler for docker Python client (Python 3.x)
python3-magnumclient - client library for Magnum API - Python 3.x
rawdns - raw DNS interface to the Docker API
ruby-docker-api - Ruby gem to interact with docker.io remote API
sen - Terminal user interface for docker engine
systemd-docker - wrapper for "docker run" to handle systemd quirks
vim-syntax-docker - Docker container engine - Vim highlighting syntax files
```

Several of the items listed are associated with the Docker container management system. Among others, the docker.io package provides the Docker container runtime and client executable, docker-compose installs the docker tool used to launch multiple containers at once and the vim-syntax-docker package allows the Vim editor to perform syntax highlighting on Dockerfiles. To find out what version of Docker the docker.io package offers, use the apt-cache policy sub command:

```
user@ubuntu:~$ apt-cache policy docker.io

docker.io:
  Installed: (none)
  Candidate: 1.11.2-0ubuntu5~16.04
  Version table:
     1.11.2-0ubuntu5~16.04 500
        500 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages
     1.10.3-0ubuntu6 500
        500 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 Packages
```

Note the version of the docker.io package. Docker is a very fast moving platform. Near the release of Ubuntu 16.04 Docker version 1.10 was current, however a few months later several significant Docker releases had been made available. If we want the latest version of Docker, installing from default packages is not the right approach. Fortunately it is easy to install the latest Docker on most Linux systems directly from get.docker.com packages.

2. Install Docker from packages

As discussed in class, Docker supplies installation instructions for many platforms. The Ubuntu 16.04 installation guide can be found online here (**do not** follow these instructions, however, if you have time, you may want to scan the content):

- https://docs.docker.com/installation/ubuntulinux

As described in the installation guide, Docker requires a 64 bit system with a Linux kernel having version 3.10 or newer. Use the `uname` command to check the version of your kernel:

```
user@ubuntu:~$ uname -r

4.4.0-31-generic
```

To get the latest version of Docker we will use the Docker supplied packages. While the installation guide provides useful and detailed instructions, Docker supplies an easy to use installer script we can simply download and execute to install Docker (sudo may prompt you for your password):

```
user@ubuntu:~$  wget -qO- https://get.docker.com/ | sh

apparmor is enabled in the kernel and apparmor utils were already installed
+ sudo -E sh -c apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
Executing: /tmp/tmp.fUEOWnhaT0/gpg.1.sh --keyserver
hkp://ha.pool.sks-keyservers.net:80
--recv-keys
58118E89F3A912897C070ADBF76221572C52609D
gpg: requesting key 2C52609D from hkp server ha.pool.sks-keyservers.net
gpg: key 2C52609D: public key "Docker Release Tool (releasedocker) <docker@docker.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
+ break
+ sudo -E sh -c apt-key adv -k 58118E89F3A912897C070ADBF76221572C52609D >/dev/null
+ sudo -E sh -c mkdir -p /etc/apt/sources.list.d
+ dpkg --print-architecture
+ sudo -E sh -c echo deb \[arch=amd64\] https://apt.dockerproject.org/repo ubuntu-xenial main > /etc/apt/sources.list.d/docker.list
+ sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-engine
Hit:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [95.7 kB]
Get:4 https://apt.dockerproject.org/repo ubuntu-xenial InRelease [30.2 kB]
Get:5 https://apt.dockerproject.org/repo ubuntu-xenial/main amd64 Packages [2,313 B]
Hit:6 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Get:7 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [319 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe i386 Packages [316 kB]
Fetched 763 kB in 1s (409 kB/s)
Reading package lists... Done
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  aufs-tools cgroupfs-mount
The following NEW packages will be installed:
  aufs-tools cgroupfs-mount docker-engine
0 upgraded, 3 newly installed, 0 to remove and 34 not upgraded.
Need to get 19.6 MB of archives.
After this operation, 102 MB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 aufs-tools amd64 1:3.2+20130722-1.1ubuntu1 [92.9 kB]
Get:2 https://apt.dockerproject.org/repo ubuntu-xenial/main amd64 docker-engine amd64 1.12.1-0~xenial [19.5 MB]
Get:3 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 cgroupfs-mount all 1.2 [4,970 B]
Fetched 19.6 MB in 19s (1,005 kB/s)
Selecting previously unselected package aufs-tools.
(Reading database ... 122337 files and directories currently installed.)
Preparing to unpack .../aufs-tools_1%3a3.2+20130722-1.1ubuntu1_amd64.deb ...
Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
Selecting previously unselected package cgroupfs-mount.
Preparing to unpack .../cgroupfs-mount_1.2_all.deb ...
Unpacking cgroupfs-mount (1.2) ...
Selecting previously unselected package docker-engine.
Preparing to unpack .../docker-engine_1.12.1-0~xenial_amd64.deb ...
Unpacking docker-engine (1.12.1-0~xenial) ...
Processing triggers for libc-bin (2.23-0ubuntu3) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for systemd (229-4ubuntu7) ...
Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
Setting up cgroupfs-mount (1.2) ...
Setting up docker-engine (1.12.1-0~xenial) ...
Processing triggers for libc-bin (2.23-0ubuntu3) ...
Processing triggers for systemd (229-4ubuntu7) ...
Processing triggers for ureadahead (0.100.0-19) ...
+ sudo -E sh -c docker version
Client:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64

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 user

Remember that you will have to log out and back in for this to take effect!

user@ubuntu:~$
```

> *If your receive the error: "E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?", your system is probably performing apt initialization in the background. If you wait a minute or two for all of the processes running /usr/bin/dpkg to exit, you should then be able to perform the installation.*

Normal user accounts must use the sudo command to run command line tools as root. For our in-class purposes, eliminating the need for sudo execution of the Docker command will simplify our practice sessions. To make it possible to connect to the local Docker daemon domain socket without sudo we need to add our user id to the “docker” group. To add the “user” user to the “docker” group execute the following command:

```
user@ubuntu:~$ sudo usermod -a -G docker user
```

Now display the ID and Group IDs for "user":

```
id user
```

```
user@ubuntu:~$ id user

uid=1000(user) gid=1000(user) groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare),999(docker)
```

The account “user” is now a member of the “docker” group. Try running “id” without an account name:

```
user@ubuntu:~$ id
uid=1000(user) gid=1000(user) groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare)
```

While the “docker” group was added to your group list, your login shell maintains the old groups. After updating your user groups you will need to restart your login shell to ensure the changes take effect. Reboot your system to complete the installation.

```
user@ubuntu:~$ sudo reboot
```

While a reboot is not completely necessary it verifies that your system is properly configured to boot up with the Docker daemon running.

When the system has restarted log back in as “user” with the password “user”.

3. Verify the installation

Check your Docker client version with the Docker client --version switch:

```
user@ubuntu:~$ docker --version

Docker version 1.12.1, build 23cf638
```

Verify that the Docker server (also called Docker daemon and Docker engine) is running using the system service interface. We are using Ubuntu 16.04 with the SystemD service manager so we can use the service status command:

```
user@ubuntu:~$ service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2016-08-21 13:39:47 PDT; 2min 36s ago
     Docs: https://docs.docker.com
 Main PID: 4548 (dockerd)
    Tasks: 18
   Memory: 55.2M
      CPU: 667ms
   CGroup: /system.slice/docker.service
           ├─4548 /usr/bin/dockerd -H fd://
           └─4671 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --shim docker-containerd-shim --metrics-interval

Aug 21 13:39:45 ubuntu dockerd[4548]: time="2016-08-21T13:39:45.940446908-07:00" level=info msg="Graph migration to content-addressability took 0.0
Aug 21 13:39:45 ubuntu dockerd[4548]: time="2016-08-21T13:39:45.940824529-07:00" level=warning msg="Your kernel does not support swap memory limit.
Aug 21 13:39:45 ubuntu dockerd[4548]: time="2016-08-21T13:39:45.941652564-07:00" level=info msg="Loading containers: start."
Aug 21 13:39:46 ubuntu dockerd[4548]: time="2016-08-21T13:39:46.472299356-07:00" level=info msg="Firewalld running: false"
Aug 21 13:39:47 ubuntu dockerd[4548]: time="2016-08-21T13:39:47.129096766-07:00" level=info msg="Default bridge (docker0) is assigned with an IP ad
Aug 21 13:39:47 ubuntu dockerd[4548]: time="2016-08-21T13:39:47.156565990-07:00" level=info msg="Loading containers: done."
Aug 21 13:39:47 ubuntu dockerd[4548]: time="2016-08-21T13:39:47.157720993-07:00" level=info msg="Daemon has completed initialization"
Aug 21 13:39:47 ubuntu dockerd[4548]: time="2016-08-21T13:39:47.157762595-07:00" level=info msg="Docker daemon" commit=23cf638 graphdriver=aufs ver
Aug 21 13:39:47 ubuntu systemd[1]: Started Docker Application Container Engine.
Aug 21 13:39:47 ubuntu dockerd[4548]: time="2016-08-21T13:39:47.172636556-07:00" level=info msg="API listen on /var/run/docker.sock"
lines 1-22/22 (END)
```

- What is the path of the service configuration file loaded by SystemD?

- Where could we find more documentation?
- What is the process ID of the Docker daemon?
- What is the name of the Docker daemon executable?
- Under the CGroup key for the docker.service two processes are listed, what are their names and IDs?
- Does your kernel support limiting swap memory?
- What Unix domain socket is the Docker daemon listening on?
- Did Docker start successfully?

Type **q** to quit the output view.

By default, the Docker daemon listens on a Unix socket for commands. This Unix socket is only accessible locally by root and the "docker" group by default. In our lab system the Docker command line client will communicate with the docker daemon using this domain socket, requiring the user to be root or to be a member of the *docker* group. Examine the permissions on the docker.sock socket file and the groups associated with your login ID.

```
user@ubuntu:~$ ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Jul 29 18:19 /var/run/docker.sock

user@ubuntu:~$ id
uid=1000(user) gid=1000(user) groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare),999(docker)
```

Now check the version of all parts of the Docker platform with the `docker version` command:

```
user@ubuntu:~$ docker version
Client:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64
user@ubuntu:~$
```

The client version information is listed first followed by the server version information. You can also use the Docker client to retrieve basic platform information from the Docker daemon:

```
user@ubuntu:~$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.12.1
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: null host bridge overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Security Options: apparmor seccomp
Kernel Version: 4.4.0-34-generic
Operating System: Ubuntu 16.04.1 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.937 GiB
Name: ubuntu
ID: 222O:USAP:JG2M:YRNA:G26Q:XQKM:XQB3:MORI:RHXJ:X4TO:H44T:QSXL
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
Insecure Registries:
 127.0.0.0/8
```

Food for thought:

- What version of Docker is your “docker” command line client?
- How many containers are running on the server?
- What version of Docker is your Docker engine?
- What is the Logging Driver in use by your Docker engine?

- What is the Storage Driver in use by your Docker engine?
- Does the word “Driver” make you think that these component can be substituted?
- What is the Root Directory used by your Storage Driver?
- Is the server in debug mode?
- What runtime is in use by your system?
- What is the Docker engine root directory?

4. Run a Container

As a final lab step we will run our first container. Use the `docker run` command to run the hello-world image:

```
user@ubuntu:~$ docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

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 Hub account:
 https://hub.docker.com

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

You now have a working Docker installation! Food for thought:

- Where might you get more examples and ideas for using Docker?
- Did the Docker engine run the container from a hello-world image it found locally?
- In the Docker output the hello-world image is listed with a suffix, what is the suffix?
- What do you think this suffix represents?

Congratulation you have successfully installed Docker!