Installation

Since installation requirements and instructions vary, I will only provide the links to the official documentation.

- **Installing on Mac**
- **Installing on Windows**
- **Installing on Linux**

Testing

Verify that you can run `docker` commands.

```
$ sudo docker run hello-world
Unable to find image' hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6eb3586dd03075f3034f10adcd
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.
```

Post-installation Steps

For Linux

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket instead of a TCP port. By default, the user `root owns Unix socket,` and other users can only access it using `sudo.` The Docker daemon always runs as the `root` user. If you do not want to preface the `docker` command with `sudo` , create a Unix group called `docker` and add users. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group.

> Warning
>
> The `docker` group grants privileges equivalent to the `root` user. For details on how this impacts security in your system.

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

1. Create the `docker` group.

```
$ sudo groupadd docker
```

Add your user to the `docker` group.

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

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

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

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

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

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
```

Configure Docker to start on boot

Most current Linux distributions (RHEL, CentOS, Fedora, Debian, Ubuntu) use `systemd` to manage which services start when the system boots. On Debian and Ubuntu, the Docker service is configured to start on boot by default. To automatically start Docker and Containerd on boot for other distros, use the commands below:

```
$ sudo systemctl enable docker.service
$ sudo systemctl enable containerd.service
```

To disable this behavior, use `disable` instead.

```
$ sudo systemctl disable docker.service
$ sudo systemctl disable containerd.service
```

Managing the Docker daemon

The docker daemon runs root-privileged processes and can be controlled with the `dockerd` binary. This daemon should be started by default and listens in socket `/var/run/docker.sock` for incoming docker requests. Any user that belongs to the group `docker` can execute it without sudo, but this group is root equivalent and should be managed with care.

The Docker daemon can be managed in Linux with any of the following options `[start, stop, restart, status]`. Example:

```
$ systemctl start docker
```

Troubleshooting

Kernel compatibility

Docker cannot run correctly if your kernel is older than version 3.10 or missing some modules. To check kernel compatibility, you can download and run the `check-config.sh` script.

```
$ curl https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh > check-config.sh
$ bash ./check-config.sh
```

The script only works on Linux, not macOS.

Cannot connect to the Docker daemon

If you see an error such as the following, your Docker client may be configured to connect to a Docker daemon on a different host, and that host may not be reachable.

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

To see which host your client is configured to connect, check the value of the `DOCKER_HOST` variable in your environment.

```
$ env | grep DOCKER_HOST
```

If this command returns a value, the Docker client is connected to a Docker daemon running on that host. If it is unset, the Docker client is set to connect to the Docker daemon running on the localhost. If it is set in error, use the following command to unset it:

```
$ unset DOCKER_HOST
```

You may need to edit your environment in files such as `~/.bashrc` or `~/.profile` to prevent the `DOCKER_HOST` variable from being set erroneously.

If `DOCKER_HOST` is set as intended, verify that the Docker daemon is running on the remote host and that a firewall or network outage is not preventing you from connecting.

Getting Started

First Container

Run the following command:

```
$ docker run -ti ubuntu /bin/bash
```

Your output should be similar to this.

```
$ docker run -ti ubuntu /bin/bash
    Unable to find image 'ubuntu:latest' locally
    latest: Pulling from library/ubuntu
    35c102085707: Pull complete
    251f5509d51d: Pull complete
    8e829fe70a46: Pull complete
    6001e1789921: Pull complete
    Digest:
sha256:d1d454df0f579c6be4d8161d227462d69e163a8ff9d20a847533989cf0c94d90
    Status: Downloaded newer image for ubuntu:latest
```

Many things happened when executing the above command.

1. We ran the command indicating that we wanted to use the `Ubuntu` image. Docker checks for the image in the local repository. If not found, it will look for it in the default Docker registry, `DockerHub,` and it will `Pull` (Download) the image and store it locally.
2. The Image is then started as a container with its file system, network, IPs, and Bridge interface.
3. Finally, it will execute the instructions provided. Since we ran the command with the options `-ti` , it will run in `Terminal Interactive` mode the command: `/bin/bash`

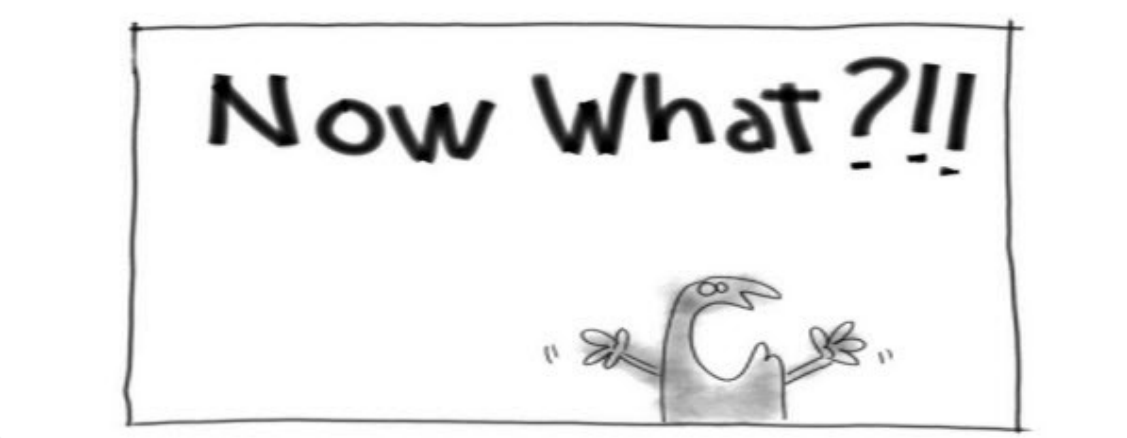

Fig. 1 - Now what?

Let us run some local commands to check our image environment.

```
root@4b5263ee03a5:/# hostname
4b5263ee03a5
root@4b5263ee03a5:/# hostname -I
172.17.0.2
root@4b5263ee03a5:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
root@4b5263ee03a5:/# ls -l home
total 0
root@4b5263ee03a5:/# man mount
bash: man: command not found
```

Notice how the container has its unique hostname, IP address, and file-system space. It looks like a regular Linux running, but **is not** many things not required have been stripped out like the manual pages. Remember that containers are not intended to be run `interactively,` so there is no need for man pages.

Let us run some more commands.

```
root@4b5263ee03a5:/# df -h
Filesystem                                  Size  Used Avail Use% Mounted
on
overlay                                      69G   29G   37G  44% /
tmpfs                                        64M     0   64M   0% /dev
tmpfs                                       7.7G     0  7.7G   0%
/sys/fs/cgroup
shm                                          64M     0   64M   0% /dev/shm
/dev/mapper/fedora_localhost--live-root      69G   29G   37G  44%
/etc/hosts
tmpfs                                       7.7G     0  7.7G   0%
/proc/asound
tmpfs                                       7.7G     0  7.7G   0%
/proc/acpi
tmpfs                                       7.7G     0  7.7G   0%
/proc/scsi
tmpfs                                       7.7G     0  7.7G   0%
/sys/firmware
root@4b5263ee03a5:/# cd /
root@4b5263ee03a5:/# free -h
               total        used        free      shared  buff/cache
available
Mem:            15G        3.9G        5.1G        1.5G        6.3G
   9.6G
Swap:           11G          0B         11G
root@4b5263ee03a5:/# mount
overlay on / type overlay
(rw,relatime,seclabel,lowerdir=/var/lib/docker/overlay2/l/XTY3VPCN6NRDDL
H4JTCJOBE2SY:/var/lib/docker/overlay2/l/C7D33OTVZQJAYL3ZJEFXIK5NU4:/var/
lib/docker/overlay2/l/NRX7LSDR6HSMMHC3QFIGZPPSTT:/var/lib/docker/overlay
2/l/PSLG57UYEWLJF5LZUH4ILPNC4V:/var/lib/docker/overlay2/l/5RDN5J67YQ7GOL
FVCRYFOJOIWX,upperdir=/var/lib/docker/overlay2/ca5fe0ac1803bcba3cfc2d7e9
c6ed1206f99b7521c1b7797ddacc5e64b8c6fd6/diff,workdir=/var/lib/docker/ove
rlay2/ca5fe0ac1803bcba3cfc2d7e9c6ed1206f99b7521c1b7797ddacc5e64b8c6fd6/w
ork)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs
(rw,nosuid,seclabel,size=65536k,mode=755,inode64)
devpts on /dev/pts type devpts
(rw,nosuid,noexec,relatime,seclabel,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime,seclabel)
tmpfs on /sys/fs/cgroup type tmpfs
(rw,nosuid,nodev,noexec,relatime,seclabel,mode=755,inode64)
cgroup on /sys/fs/cgroup/systemd type cgroup
(ro,nosuid,nodev,noexec,relatime,seclabel,xattr,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(ro,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(ro,nosuid,nodev,noexec,relatime,seclabel,cpuset)
/dev/mapper/fedora_localhost--live-root on /etc/resolv.conf type ext4
(rw,relatime,seclabel)
/dev/mapper/fedora_localhost--live-root on /etc/hostname type ext4
(rw,relatime,seclabel)
/dev/mapper/fedora_localhost--live-root on /etc/hosts type ext4
(rw,relatime,seclabel)
```

Also, notice how some of the resources look the same as the ones in our host's because they are shared like the Memory and CPU. When you exit the container, it automatically deletes all traces of it. The container will only run for as long as the specified commands are running. The container still exists, but it is OFF.

Pet's vs. Cattle

You just saw a container be created and then destroyed! Why?.

Pets vs. cattle is a simple analogy that allows you to view the containers in the correct perspective.

> `Containers are cattle not pets` .

- **Pets**: Get unique names and special treatment. When they are "sick," we worry that they are carefully nursed back to health, often with a significant time and financial investment.
- **Cattle**: Are part of an identical group. We see them as numbers, not names; they receive no special treatment. When something goes wrong, they are replaced, and we try to minimize their investment as much as possible.

Fig. 2 - Pet’s V.S. Cattle

In terms of containers, this analogy suggests that we should not struggle when a piece of infrastructure breaks, nor should it take an entire team of people to nurse it back to health. Your infrastructure should be made up of components you can treat like cattle, self-sufficient, easily replaced, and manageable by the hundreds or thousands. Unlike virtual machines or physical servers that require special attention, containers can be spun up, replicated, destroyed, and managed with much greater flexibility.

Container creation workflow

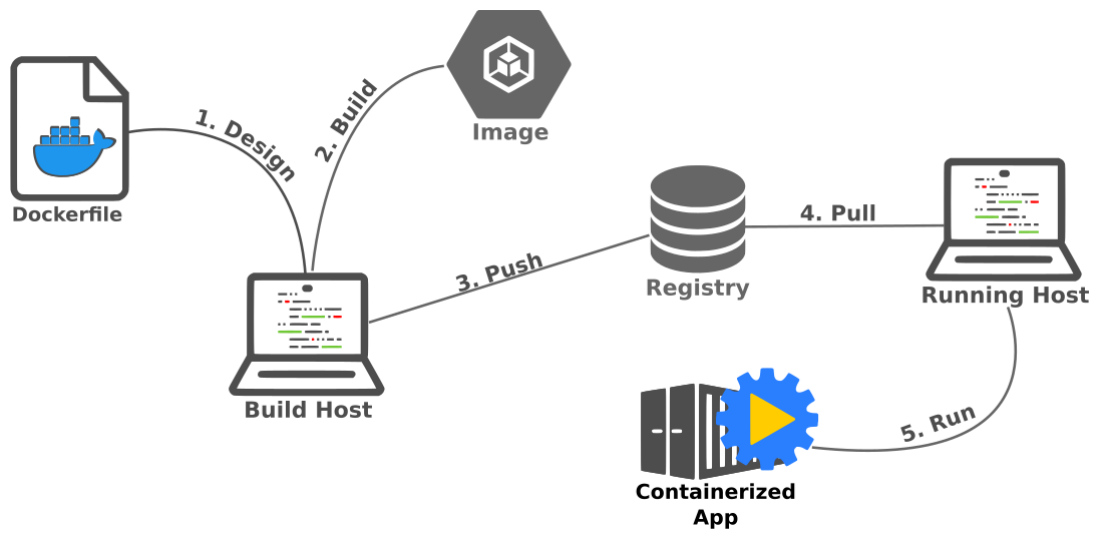

Fig. 3 - Container creation workflow

Every container must be designed, built, and tested locally to be pushed into the registry and made available to whoever will be running it.

Docker Client [CLI]

Listing

We can see the list of `all` the containers that have been executed with the command `docker ps -a'

```
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS
               PORTS     NAMES
4b5263ee03a5   ubuntu         "/bin/bash"   2 hours ago     Exited (127)
19 seconds ago             quizzical_engelbart
ce35c4e6c746   fedora         "/bin/bash"   14 months ago   Exited (0) 14
months ago                optimistic_kalam
fbbe7d946f4e   ubuntu         "sh"          15 months ago   Exited (0) 15
months ago                eager_pike
153d402cb0c2   fedora         "sh"          15 months ago   Exited (130)
15 months ago              wizardly_allen
b1da1e6e3ea8   hello-world    "/hello"      16 months ago   Exited (0) 16
months ago                eager_gauss
13149fb95bc8   hello-world    "/hello"      16 months ago   Created
                          condescending_visvesvaraya
```

Or just the ones that are currently running

```
$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS
               PORTS     NAMES
13149fb95bc8   hello-world    "/hello"      16 months ago   Created
                          condescending_visvesvaraya
```

You can also run `ps -l` to show the `last` container executed.

```
$ docker ps -l
CONTAINER ID   IMAGE     COMMAND       CREATED       STATUS
        PORTS     NAMES
4b5263ee03a5   ubuntu    "/bin/bash"   2 hours ago   Exited (127) 2
minutes ago             quizzical_engelbart
```

Or the size of the containers with the `-s' option

```
 docker ps -ls
CONTAINER ID   IMAGE     COMMAND       CREATED       STATUS
      PORTS  NAMES               SIZE
4b5263ee03a5   ubuntu    "/bin/bash"   2 hours ago   Exited (127) 3
minutes ago        quizzical_engelbart 105B (virt 64.2MB)
```

There are three ways a container can be identified: `Short UUID` , `Long UUID,` and `Name.`

Inspect

The `docker inspect` command will interrogate our container and return its configuration information, name, commands, network configuration, and more valuable data.

```
docker inspect 4b5263ee03a5
[
    {
        "Id":
"4b5263ee03a5e7d83a9916eb194c59759de97852d9ec231cb53b9e0988ae2535",
        "Created": "2021-06-28T20:35:50.587156443Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "exited",
            "Running": false,
...
-Output_truncated
```

you can use it with the `--format` flag to limit the amount of data returned as in the following examples:

```
docker inspect --format='{{.State.Running}}' daemon_alice
docker inspect --format='{{.NetworkSettings.IPAddress}}' daemon_alice
docker inspect --format='{{.Name}} {{.State.Running}}' daemon_alice
$ docker inspect --format='{{.State.Running}}' 4b5263ee03a5
false
```

We can list multiple containers by listing them.

```
docker inspect --format='{{.Name}} {{.State.Running}}' daemon_alice
daemon_dan daemon_juan
```

Processes Management

We can monitor the processes running inside of a container with `top`

```
$ docker top a7cc3837caf4
UID      PID      PPID      C      STIME      TTY      TIME          CMD
root     22397    22376     0      18:35      ?        00:00:00
/bin/sleep 5000
```

You can also use `stats` to see `CPU MEM IO` and more.

```
$ docker stats
CONTAINER ID   NAME             CPU %     MEM USAGE / LIMIT   MEM %
NET I/O       BLOCK I/O     PIDS
a7cc3837caf4   hardcore_moore   0.00%     388KiB / 15.34GiB   0.00%
4.43kB / 0B   73.7kB / 0B   1
```

Monitoring

We can see what is going on inside a container with `logs`

```
$ docker logs daemon_juan
```

We can also simulate a `tail -f` with the `-f` flag. Stop with Ctrl+C

```
$ docker logs -f daemon_juan
```

or with `--tail`

```
$ docker logs --tail 10 daemon_juan
```

We can also add timestamps to our output `-t`

```
$ docker logs -ft daemon_juan
```

You can control the logging driver with the `--log-driver` flag in both the daemon and client and can be passed along with the `run` command. We can change from the default format of `JSON` to `syslog` that disable docker logs and send them to the syslog files.

```
$ docker run --log-driver="syslog" --name daemon_dan -d ubuntu /bin/sh -c
"while true; do echo hello dan; sleep 1; done"
```

The above will redirect the logs to syslog; therefore, `docker logs` will not show any output.

Another redirection option is `none` that disables logging.

```
$ docker run --log-driver=none --name daemon_dan -d ubuntu /bin/sh -c
"while true; do echo hello dan; sleep 1; done"
```

Naming

Docker will automatically generate a name for each container. But if we want to specify a name, we use: `--name`

```
$ docker run --name contained_bob -ti ubuntu /bin/bash
```

- Only characters and numbers are allowed.
- Names must be unique

Starting

There is not much to say about it. If the container exist, it will start!

```
$ docker start mycontainer
```

Creating

Nevertheless, if the container does not exist, it needs to be created first. `create` Creates the container and all its dependencies but does not run it.

```
$ docker create --name mycontainer ubuntu
50d16cb60fdc2ba4a1a0c08919b6a17b7b1488cd78390922c9ad14cb8bf998f1
```

Attaching

Containers will run with all the specified options. We can reattach to our session as follows:

```
$ docker attach contained_bob
```

Daemonizing

If we need a long-run container, we can use a daemonized `-d' one that does not have an interactive session which is ideal for applications and services.

```
$ docker run --name daemon_juan -d ubuntu /bin/sh -c "while true; do echo
echo world; sleep 1; done"
```

Exec

We can run new processes on existing containers with `exec.` There are two types of commands we can run:

- Background and Interactive
- Background & Foreground processes!

```
# Non-Interactive command [Simple]
$ docker exec daemon_dan ls -l /etc/new_config_file
# Background command [Daemonized]
$ docker exec -d daemon_dan touch /etc/new_config_file
# Interactive command [Interactive]
$ docker exec -ti daemon_dan /bin/bash
```

Stopping

This sends a SIGTERM signal to the container running process.

```
$ docker stop daemon_juan
```

You can also send a SIGKILL with

```
$ docker kill daemon_juan
```

We can check `N number` of recently stopped containers with

```
$ docker ps -n 10
```

Restarting

You can use the `--restart` flag to restart a container automatically.

It does it by checking the exit code and making a decision. The default behavior is not to restart!.

```
$ docker run --restart=always --name daemon_alice -d ubuntu /bin/sh -c "while true; do echo hello alice; sleep 1; done"
```

Options:

- `on-failure` This also accepts an optional count max: `on-failure:5` attempting to restart to a max of 5 times
- `always`

Deleting

You can use the `rm` command to remove a container definitively. If the container is running to force deleting it, you can add the `-f` flag!

```
$ docker rm daemon_alice
# Error response from daemon: You cannot remove a running container 70a3dcabd2b022deaaa804c68dc7b2e9185fe9388f221cb62e28848ed06ea165. Stop the container before attempting removal or force remove
```

If needed, it can be forced.

Fig. 4 - Finally Over!