Container Basics - Lab Guide Hands-On

December 15, 2020

Table of Contents

Lab 1: Install Docker

In this Lab you will install Docker and familiarize yourself with some of the basic commands.
We will follow the installation process as described in the Docker documentation - „Install using the convenience script"

https://docs.docker.com/engine/install/ubuntu/

Installation steps

- Download and execute the installation script

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

- Add your username to the docker group:

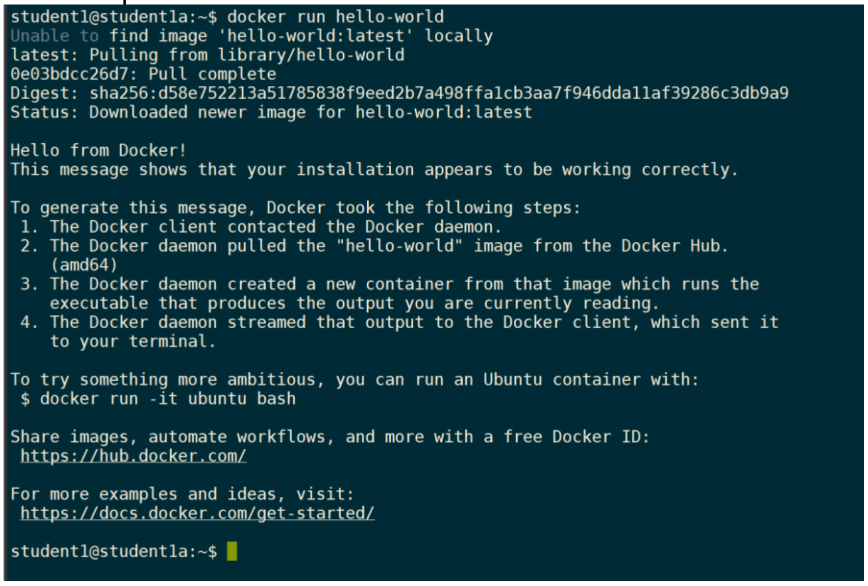
```
sudo usermod -aG docker username
```

where "*username*" is your username.

- Close the session and Login again with your username (this is needed so your environment recognizes the changes you made in the last step)
- Run your first container

```
docker run hello-world
```

Your Output should look like this:



```
student1@student1a:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9
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.
    (amd64)
 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://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

student1@student1a:~$
```

Lab 2: Docker - Basics

Explore basic commands

In this section we will run a few commands to get a basic understanding of the Docker commands. Feel free to try additional “show / display” commands, do not make any changes to the configuration yet 😉
Run the following commands and have a look at the output:

```
docker
```

```
docker -v
```

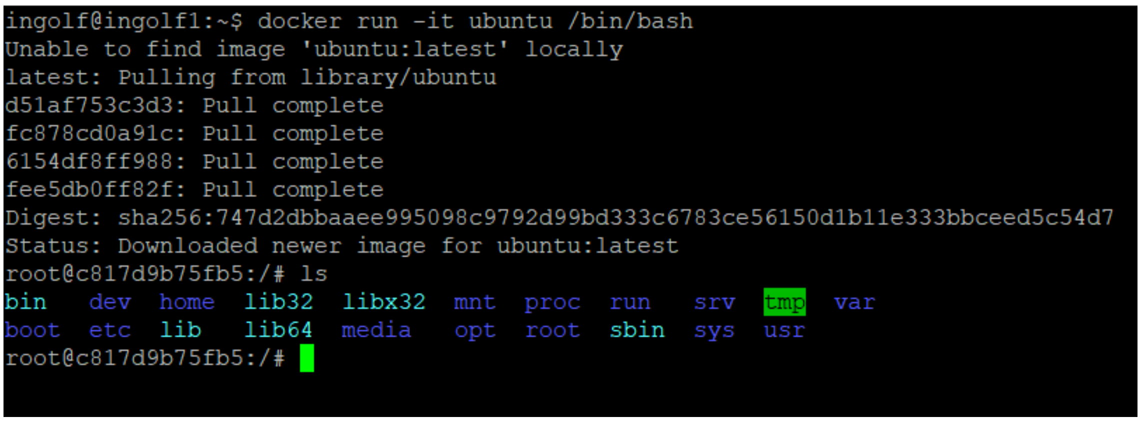
```
docker ps --help
```

Run Ubuntu container in interactive mode

Most of the times containers are started and run in detached mode. In this part we will run a container in interactive mode so that we are connected to the shell of the container.

```
docker run -it ubuntu /bin/bash
```

Your output should look like this:



```
ingolf@ingolf1:~$ docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
root@c817d9b75fb5:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@c817d9b75fb5:/#
```

Have a close look at the terminal and see the difference in the prompt. You are inside the container now.

Logout of the container shell with

```
exit
```

This also stops the bash process and this way also the container.

Explore the difference between these two commands:

```
docker ps
```

```
docker ps -a
```

Working with Images

Images are fundamental to containers, every container you start is built from an image. You can list the images on your system with the following command:

```
docker images
```

Images can either be created by yourself on your own local system or you can pull images from a registry. Registries can be internal in your company or you can use an external public registry. The most used external public registry is the Docker Hub.

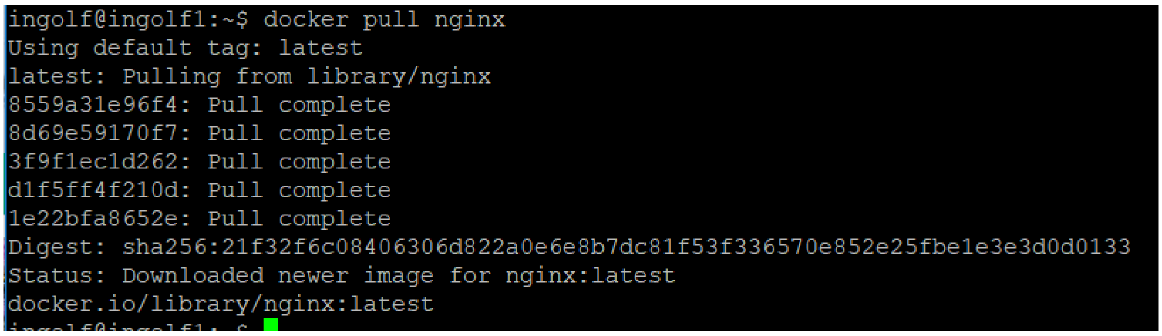
Open a browser window and have a short look at Docker hub.

> https://hub.docker.com/
>
> For example, find the latest image of Nginx on Docker Hub:
>
> https://hub.docker.com/_/nginx
> (NGINX is a server software that has lots of different functions, we will use it as a webserver only in our labs)

We will now pull the image for NGINX from the Docker Hub:

```
docker pull nginx
```

This should produce following output.



```
ingolf@ingolf1:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8559a31e96f4: Pull complete
8d69e59170f7: Pull complete
3f9f1ec1d262: Pull complete
d1f5ff4f210d: Pull complete
1e22bfa8652e: Pull complete
Digest: sha256:21f32f6c08406306d822a0e6e8b7dc81f53f336570e852e25fbe1e3e3d0d0133
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

> As you can see an image is constructed out of different layers. If a newer verison of an image has something changed only the affected layers are pulled again.
>
> We don’t cover the creation of images in detail, but there’s lots of materials public available if you want to create your own images.

Now run the image.

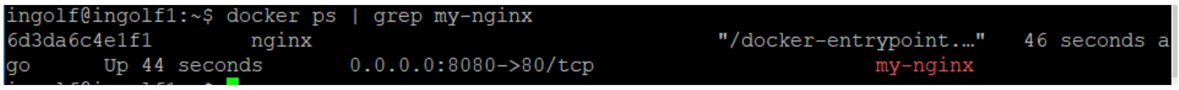
```
docker run --name my-nginx -d -p 8080:80 nginx
```

This also shows how to connect your container with the outside world.

> Explore the Docker documentation and try to figure out what the different options of the run command are doing:
> https://docs.docker.com/engine/reference/run/

Verify that the nginx container is started

```
docker ps | grep my-nginx
```

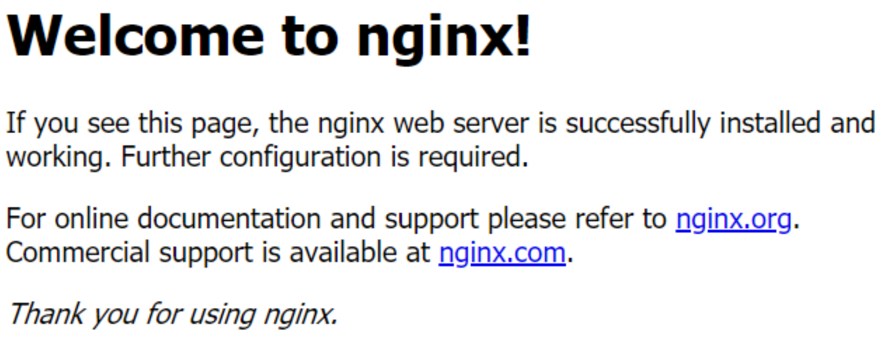


```
ingolf@ingolf1:~$ docker ps | grep my-nginx
6d3da6c4e1f1        nginx                              "/docker-entrypoint.…"   46 seconds a
go        Up 44 seconds        0.0.0.0:8080->80/tcp                    my-nginx
```

Next, we connect to the started webserver container with a browser, go with your browser to http://<Your-IP>:8080

> Where <Your-IP> refers to the IP address of your Lab VM.

Your browser should show the default nginx start page:



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Get details about your nginx container:

```
docker inspect my-nginx
```

https://docs.docker.com/engine/reference/commandline/inspect/

```
docker logs my-nginx
```

https://docs.docker.com/engine/reference/commandline/logs/

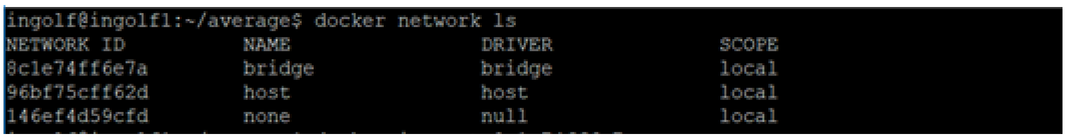
Docker networking

Docker relies on standard Linux networking technologies to create networks for containers. Technologies like bridges and virtual adapters are used to connect containers to different network so that containers can communicate with each other and can be connected to the outside world.

Docker create a default network during installation which we use for all the things we do in our labs, but there is of course lots of options to create a much more complex networking environment if needed.

Check which networks are available in your environment:

```
docker network ls
```



```
ingolf@ingolf1:~/average$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8c1e74ff6e7a        bridge              bridge              local
96bf75cff62d        host                host                local
146ef4d59cfd        none                null                local
```

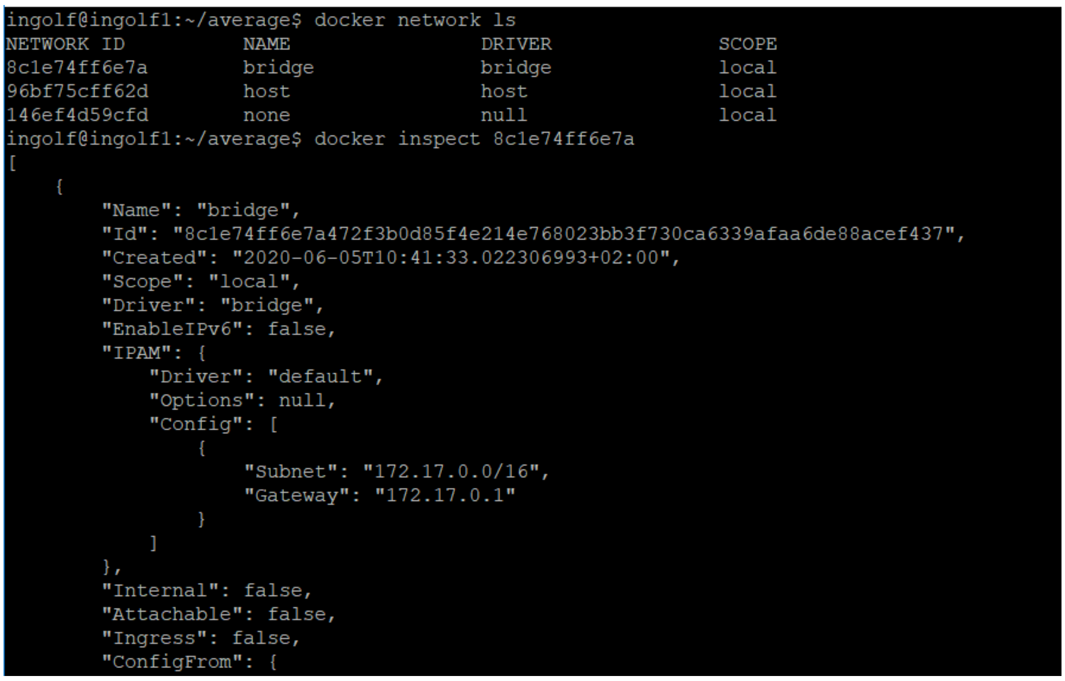
Default is "bridge" network

```
docker inspect <bridge-network-ID>
```

Everything in docker has an id – networks, containers, images…. If you want to be absolutely sure to interact with the right element using the ID is a good idea.

You do not have to type the full id, it's enough to type so much that it's clearly identified.
Try the inspect command also with containers or images.

This should produce the following output:



```
ingolf@ingolf1:~/average$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8c1e74ff6e7a        bridge              bridge              local
96bf75cff62d        host                host                local
146ef4d59cfd        none                null                local
ingolf@ingolf1:~/average$ docker inspect 8c1e74ff6e7a
[
    {
        "Name": "bridge",
        "Id": "8c1e74ff6e7a472f3b0d85f4e214e768023bb3f730ca6339afaa6de88acef437",
        "Created": "2020-06-05T10:41:33.022306993+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
```

Next we start a container in interactive mode (like we did with the ubuntu container earlier). This time we use an alpine Linux container:

```
docker run -it alpine /bin/sh
```

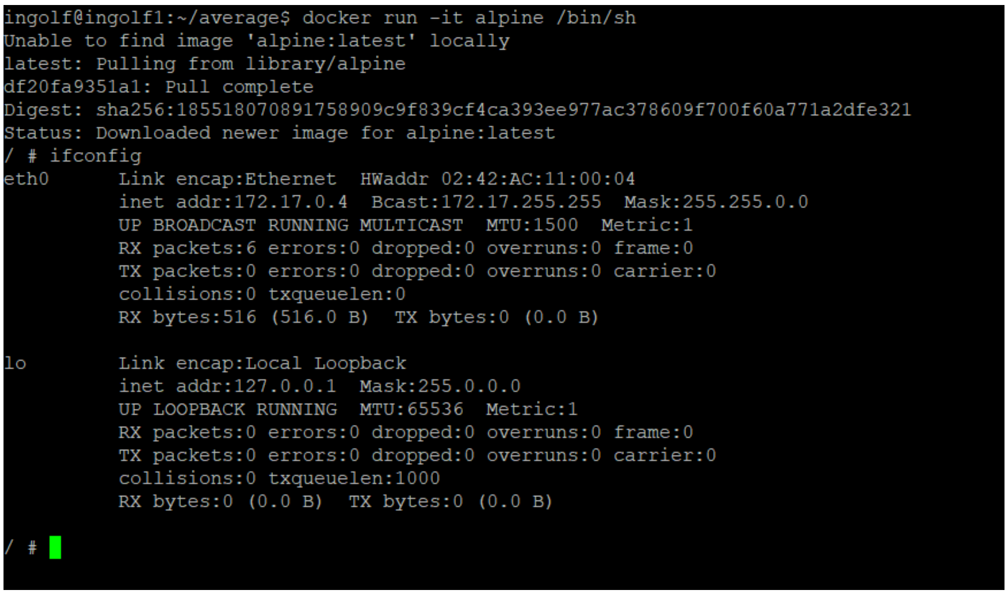
Have a look at the prompt again, we are inside the container again but the prompt looks different for alpine.

“-it” in the command ist for interactive, so your connected to the CLI prompt of the container.

Now we verify the network configuration inside the container:

```
ifconfig
```

This should give you an output similar to this one:



```
ingolf@ingolf1:~/average$ docker run -it alpine /bin/sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
df20fa9351a1: Pull complete
Digest: sha256:185518070891758909c9f839cf4ca393ee977ac378609f700f60a771a2dfe321
Status: Downloaded newer image for alpine:latest
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:04
          inet addr:172.17.0.4  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:516 (516.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ #
```

Logout of the container again

```
exit
```

You can also disconnect from the container with Ctrl-c which will also stop the container. If you press Ctrl-p and then Ctrl-q you disconnect from the container but the container keeps running. Try to find the command you could use to re-attach to the container 😉

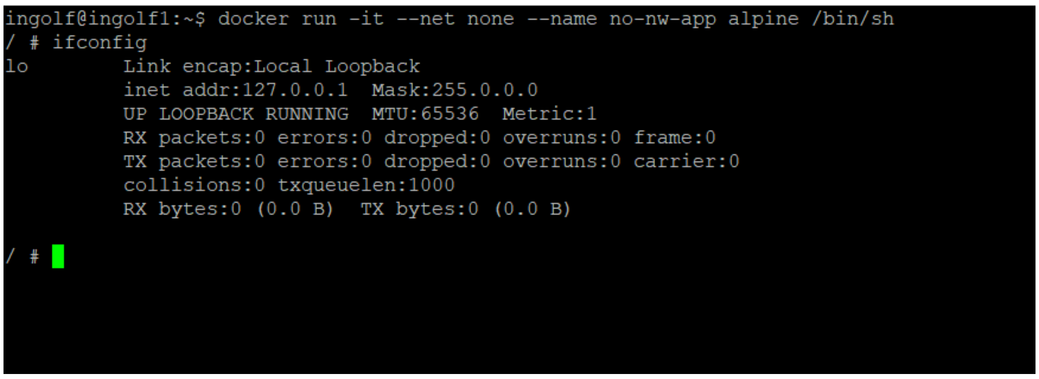
Now let's run a container with no network:

```
docker run -it --net none --name no-nw-app alpine /bin/sh
```

From within the container explore the network again:

```
ifconfig
```

Your output should be similar to this:



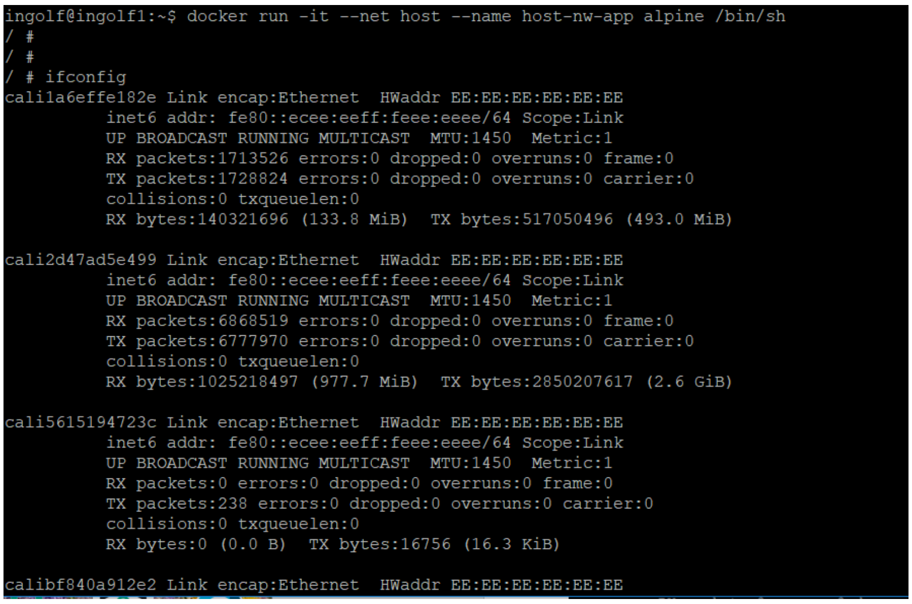
```
ingolf@ingolf1:~$ docker run -it --net none --name no-nw-app alpine /bin/sh
/ # ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0  (0.0 B)  TX bytes:0  (0.0 B)

/ #
```

And finally, a container with "host" network:

```
docker run -it --net host --name host-nw-app alpine /bin/sh
```

Check the networking inside the container again:

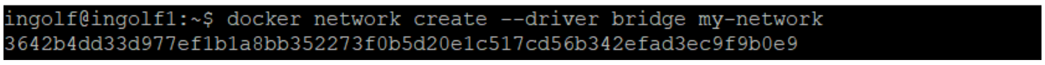
```
ifconfig
```



```
ingolf@ingolf1:~$ docker run -it --net host --name host-nw-app alpine /bin/sh
/ #
/ #
/ # ifconfig
cali1a6effe182e Link encap:Ethernet  HWaddr EE:EE:EE:EE:EE:EE
          inet6 addr: fe80::ecee:eeff:feee:eeee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:1713526 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1728824 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:140321696 (133.8 MiB)  TX bytes:517050496 (493.0 MiB)

cali2d47ad5e499 Link encap:Ethernet  HWaddr EE:EE:EE:EE:EE:EE
          inet6 addr: fe80::ecee:eeff:feee:eeee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:6868519 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6777970 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1025218497 (977.7 MiB)  TX bytes:2850207617 (2.6 GiB)

cali5615194723c Link encap:Ethernet  HWaddr EE:EE:EE:EE:EE:EE
          inet6 addr: fe80::ecee:eeff:feee:eeee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:238 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0  (0.0 B)  TX bytes:16756 (16.3 KiB)

calibf840a912e2 Link encap:Ethernet  HWaddr EE:EE:EE:EE:EE:EE
```

You can also create your own bridge network:

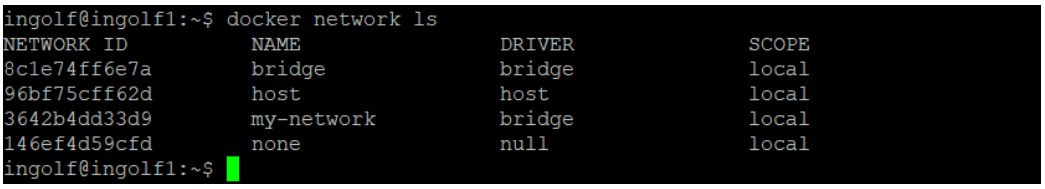
```
docker network create --driver bridge my-network
```



```
ingolf@ingolf1:~$ docker network create --driver bridge my-network
3642b4dd33d977ef1b1a8bb352273f0b5d20e1c517cd56b342efad3ec9f9b0e9
```

Use the LS command to verify your networks:

```
docker network ls
```



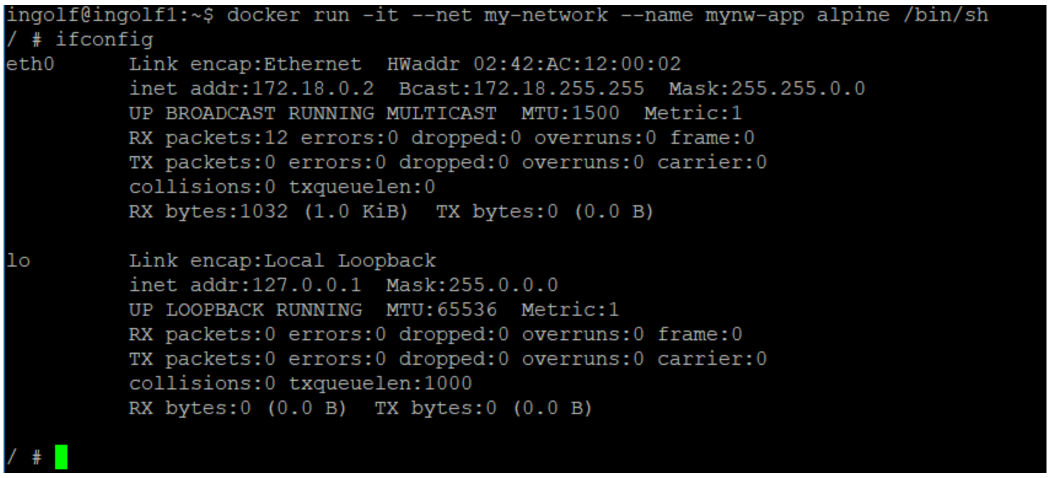
```
ingolf@ingolf1:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8c1e74ff6e7a        bridge              bridge              local
96bf75cff62d        host                host                local
3642b4dd33d9        my-network          bridge              local
146ef4d59cfd        none                null                local
ingolf@ingolf1:~$
```

Start another container with your new network and explore the differences in the container networking:

```
docker run -it --net my-network --name mynw-app alpine /bin/sh
```

from inside the container:

```
ifconfig
```



```
ingolf@ingolf1:~$ docker run -it --net my-network --name mynw-app alpine /bin/sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1032 (1.0 KiB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ #
```

For more information about Docker networking, please refer to https://docs.docker.com/network/

This concludes the docker basics part of the labs.

Lab 3: Docker – Advanced

Docker Compose

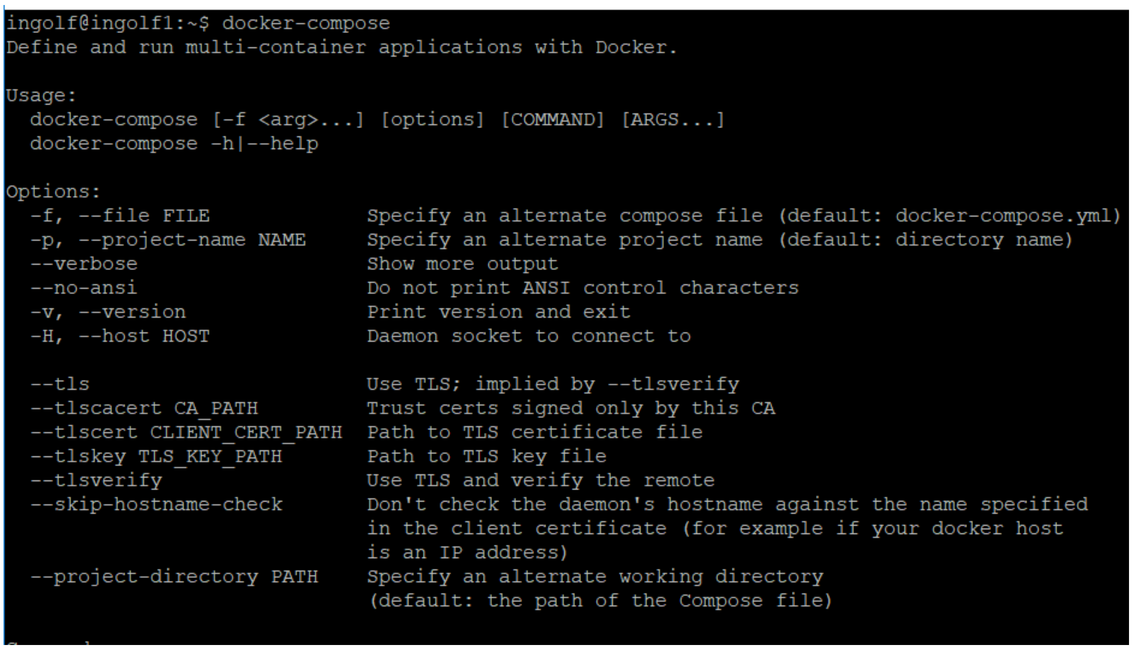
Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services.

https://docs.docker.com/compose/

The tool needs to be installed separately.

```
sudo apt-get install docker-compose
```

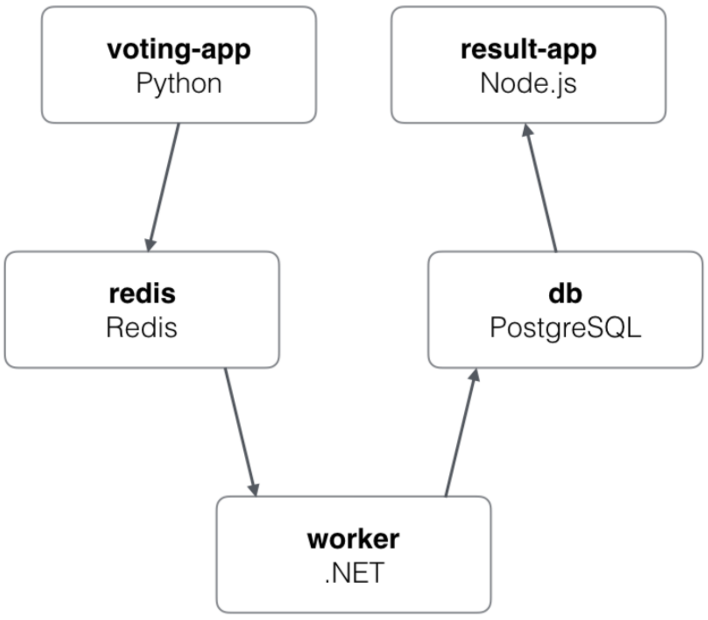
```
docker-compose
```



```
ingolf@ingolf1:~$ docker-compose
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name (default: directory name)
  --verbose                   Show more output
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the name specified
                              in the client certificate (for example if your docker host
                              is an IP address)
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
```

Files for this example application are accessible on GitHub:
https://github.com/dockersamples/example-voting-app

Application architecture:



Download the **master.zip** file using wget

```
wget insertURLhere
```

Unzip the file

```
unzip insertURLhere
```

Jump to the folder of the voting app on your host

```
cd example-voting-app
```

Check what is in there

```
ls
```

Explore the docker-compose.yml file.

```
view docker-compose.yml
```

And now deploy your Example Voting App using Docker Compose:

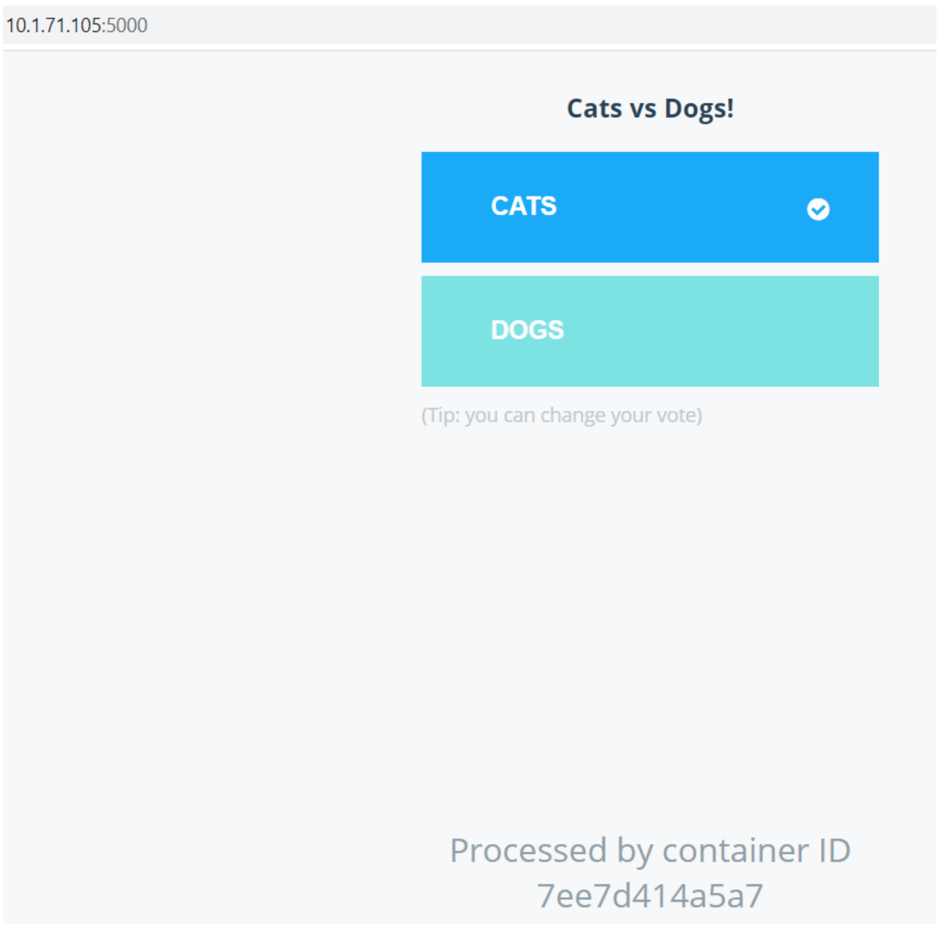
```
docker-compose up
```

Watch the deployment of your application finish and test the application with your browser:

http://<your-IP>:5000

http://<your-IP>:5001

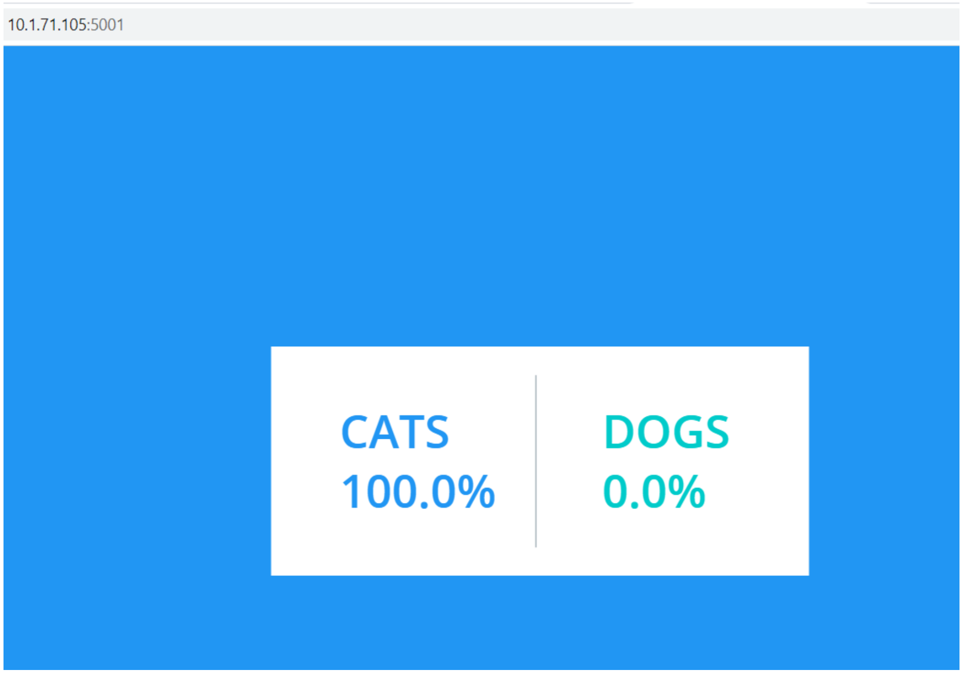
Where <your-IP> refers to the IP address of your Lab VM.


10.1.71.105:5000

Cats vs Dogs!

(Tip: you can change your vote)

Processed by container ID 7ee7d414a5a7



Stop the application:

CTRL-C

```
docker-compose down
```

Working with own Docker Registry

Introduction

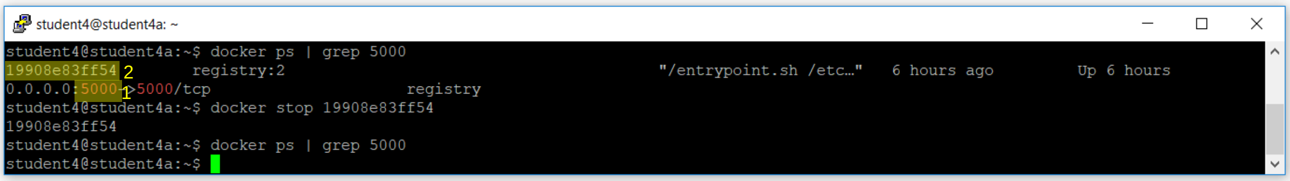
Up until this point, we always used the default Docker registry.

> You can access this registry at https://hub.docker.com.
>
> A self-hosted registry can especially be useful in the context of a CI/CD pipeline. Here, images might contain sensitive data which should not be stored off-site. The Docker registry server is open-source and licensed under the Apache License.

In this Lab exercise, we are going to create another custom image and we are going to transfer it to a second machine using a registry.
First, we are going to create a docker registry on your first VM. Second, we are going to create a new custom image on your VM. Afterwards we will upload this image to our new created registry. As a fourth step, we are going to pull this image from a second VM and run it there.

Step 1: Create your own docker registry
Docker communicates with Registries using HTTPS. In this example we are going to stick to the insecure HTTP protocol for simplicity. In all standard examples, port 5000 is used to communicate with the registries. However, it is possible that port 5000 is already occupied by a different application.
So, we first need to check whether any container is already occupying port 5000.
We are going to get a list of all container using docker ps. Then we are going to pipe the result of this command to the input of the grep command using |. The grep command the searches for the number 5000 in the list of containers.

```
docker ps | grep 5000
```



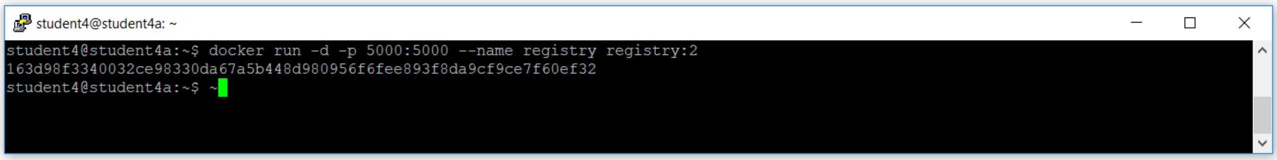
```
student4@student4a:~$ docker ps | grep 5000
19908e83ff54        registry:2                                          "/entrypoint.sh /etc…"   6 hours ago         Up 6 hours
0.0.0.0:5000->5000/tcp                registry
student4@student4a:~$ docker stop 19908e83ff54
19908e83ff54
student4@student4a:~$ docker ps | grep 5000
student4@student4a:~$
```

If there is a server running on port 5000 (1 in the picture), copy its ID (2 in the picture). Then stop it using the command

```
docker stop <container-id>
```

Docker provides an image for hosting your own registry. You can run a container from this image using the following command:

```
docker run -d -p 5000:5000 --name registry registry:2
```



```
student4@student4a:~$ docker run -d -p 5000:5000 --name registry registry:2
163d98f3340032ce98330da67a5b448d980956f6fee893f8da9cf9ce7f60ef32
student4@student4a:~$ ~
```

Step 2: Create a new image

You might have already done this in Lab 2, but we are going to create a fresh image to avoid errors.

Start with a base image

```
docker run -it ubuntu /bin/bash
```

Inside the container execute the following commands:

```
apt-get update
```

```
apt-get install nodejs
```

While installing NodeJS, you might have to select your geographic region, so that NodeJS can select the correct geographic region.

If Nodejs is installed, you have to install the text editor “vim” to your host.

```
apt-get install vim
```

```
mkdir /root/random
```

```
cd /root/random
```

Create a Node.js program with vi:

```
vi random.js
```

Enter the following code to the file random.js:

If you are not familiar with vi, find help in the appendix

```
#!/usr/bin/env nodejs
console.log(Math.floor(Math.random() * 100));
```

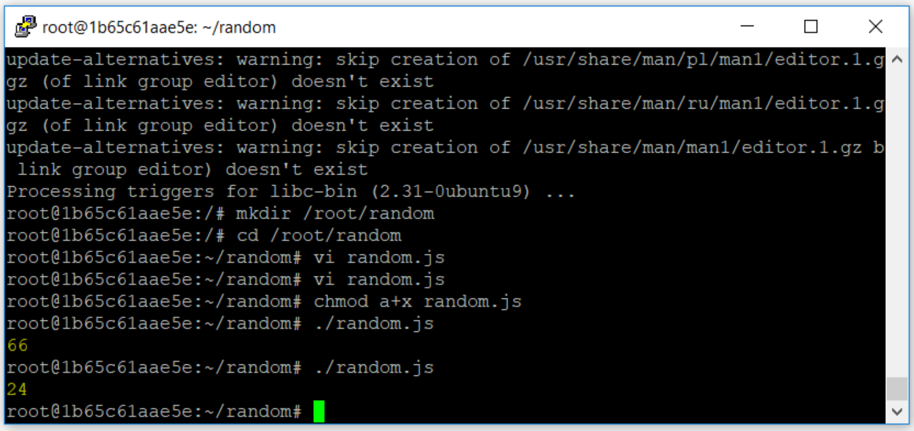
close vi

Give all users the right to execute the file:

```
chmod a+x random.js
```

Test the program

```
./random.js
```

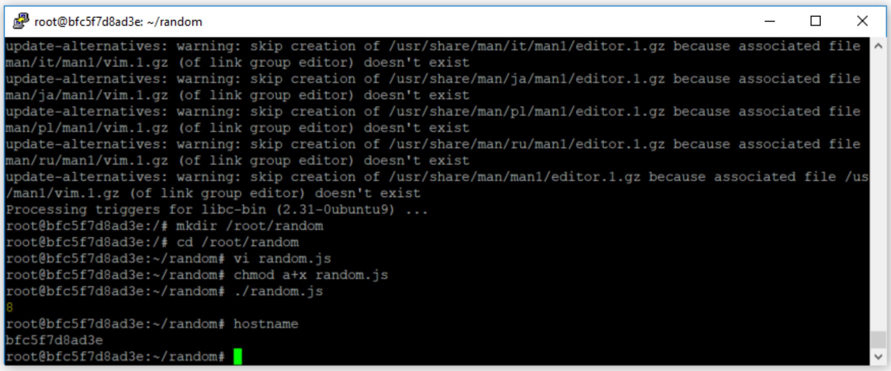


```
update-alternatives: warning: skip creation of /usr/share/man/pl/man1/editor.1.g
gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ru/man1/editor.1.g
gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz b
 link group editor) doesn't exist
Processing triggers for libc-bin (2.31-0ubuntu9) ...
root@1b65c61aae5e:/# mkdir /root/random
root@1b65c61aae5e:/# cd /root/random
root@1b65c61aae5e:~/random# vi random.js
root@1b65c61aae5e:~/random# vi random.js
root@1b65c61aae5e:~/random# chmod a+x random.js
root@1b65c61aae5e:~/random# ./random.js
66
root@1b65c61aae5e:~/random# ./random.js
24
root@1b65c61aae5e:~/random#
```

Get your container ID and copy it to the clipboard. The container id is the same as the hostname of the container:

```
hostname
```

Example



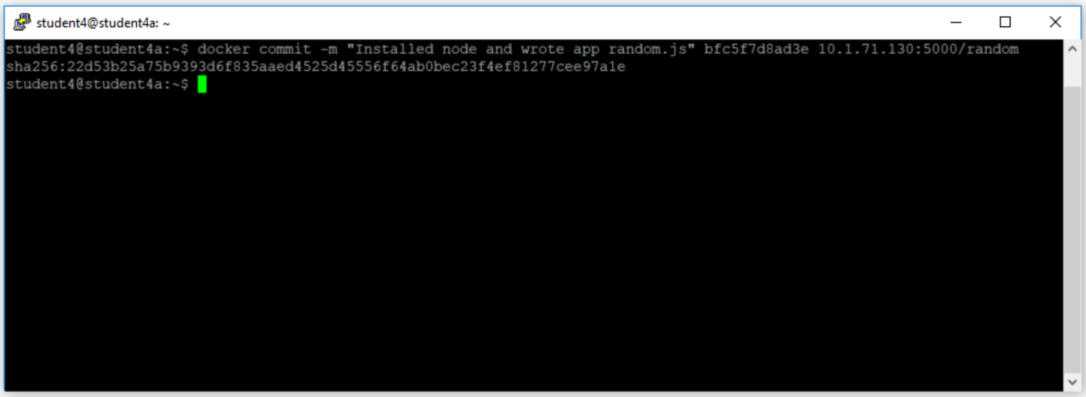
```
update-alternatives: warning: skip creation of /usr/share/man/it/man1/editor.1.gz because associated file
man/it/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ja/man1/editor.1.gz because associated file
man/ja/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/pl/man1/editor.1.gz because associated file
man/pl/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ru/man1/editor.1.gz because associated file
man/ru/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz because associated file /us
/man1/vim.1.gz (of link group editor) doesn't exist
Processing triggers for libc-bin (2.31-0ubuntu9) ...
root@bfc5f7d8ad3e:/# mkdir /root/random
root@bfc5f7d8ad3e:/# cd /root/random
root@bfc5f7d8ad3e:~/random# vi random.js
root@bfc5f7d8ad3e:~/random# chmod a+x random.js
root@bfc5f7d8ad3e:~/random# ./random.js
8
root@bfc5f7d8ad3e:~/random# hostname
bfc5f7d8ad3e
root@bfc5f7d8ad3e:~/random#
```

If you press Ctrl-p and then Ctrl-q you disconnect from the container but the container keeps running.

Now exit the bash shell, commit your changes and tag the newly created image:

```
docker commit -m "Installed node and wrote app random.js" <container-id> <ip-address-of-first-vm>:5000/random
```

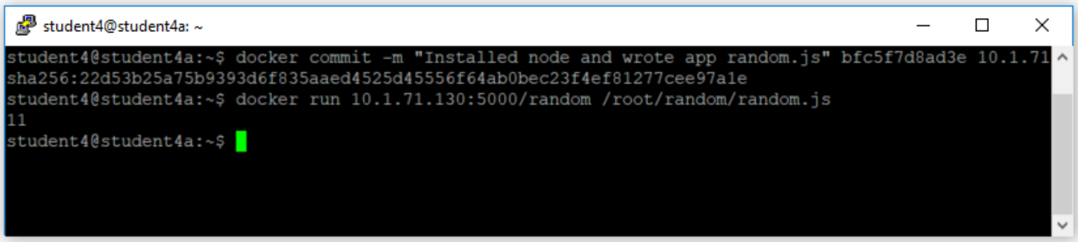
The argument <container-id> refers to the hostname from the previous step.



```
student4@student4a:~$ docker commit -m "Installed node and wrote app random.js" bfc5f7d8ad3e 10.1.71.130:5000/random
sha256:22d53b25a75b9393d6f835aaed4525d45556f64ab0bec23f4ef81277cee97a1e
student4@student4a:~$
```

Test the application in a container using the new Image ID:

```
docker run <ip-address-of-first-vm>:5000/random /root/random/random.js
```



```
student4@student4a:~$ docker commit -m "Installed node and wrote app random.js" bfc5f7d8ad3e 10.1.71
sha256:22d53b25a75b9393d6f835aaed4525d45556f64ab0bec23f4ef81277cee97a1e
student4@student4a:~$ docker run 10.1.71.130:5000/random /root/random/random.js
11
student4@student4a:~$
```

Step 3: Push your image to your registry

Before we can access our registry, we need to instruct docker to use http instead of https to access the registry.

Edit the docker daemon configuration file:

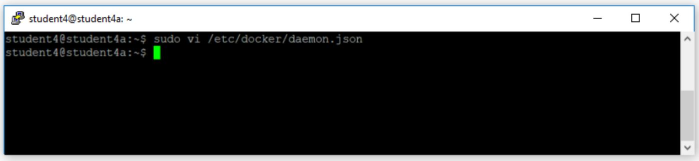
```
sudo vi /etc/docker/daemon.json
```

Paste the following configuration snippet into the config file

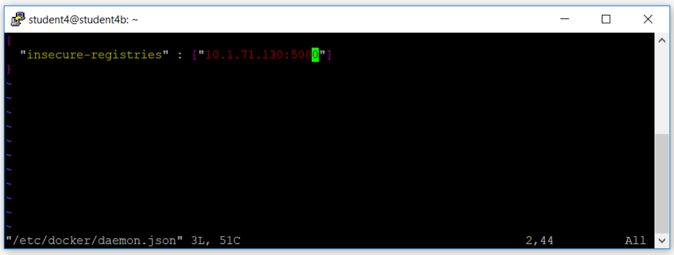
don't forget to switch to insert mode

Replace <first-vm-ip-address> with the IP-Address of your first VM.

```
{
  "insecure-registries" : ["<first-vm-ip-address>:5000"]
}
```



```
student4@student4a:~$ sudo vi /etc/docker/daemon.json
student4@student4a:~$
```

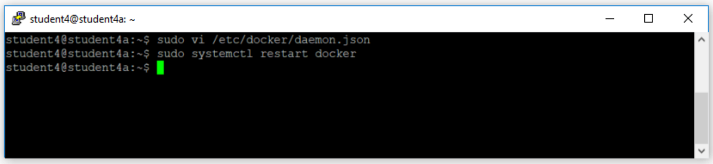


```
{
  "insecure-registries" : ["10.1.71.130:5000"]
}
~
~
~
~
~
~
~
~
~
~
~
"/etc/docker/daemon.json" 3L, 51C                          2,44          All
```

Save and exit.

Restart docker

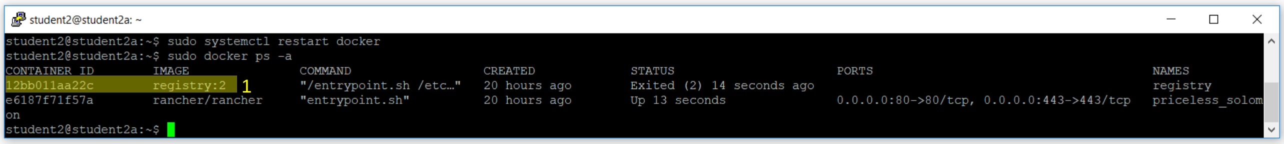
```
sudo systemctl restart docker
```



```
student4@student4a:~$ sudo vi /etc/docker/daemon.json
student4@student4a:~$ sudo systemctl restart docker
student4@student4a:~$
```

While restarting Docker, we also stopped our registry container. Let's check the running containers

```
sudo docker ps -a
```



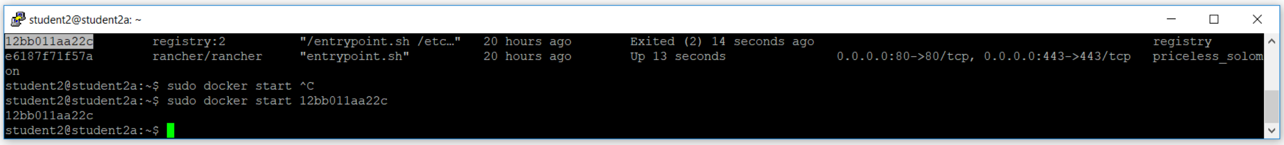
```
student2@student2a:~$ sudo systemctl restart docker
student2@student2a:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS                                      NAMES
12bb011aa22c        registry:2  1       "/entrypoint.sh /etc…"   20 hours ago        Exited (2) 14 seconds ago                                              registry
e6187f71f57a        rancher/rancher     "entrypoint.sh"          20 hours ago        Up 13 seconds               0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp   priceless_solom
on
student2@student2a:~$
```

As you can see, our registry container was stopped during the restart.

We need to start it using the following command.

```
docker start <registry-container-id>
```

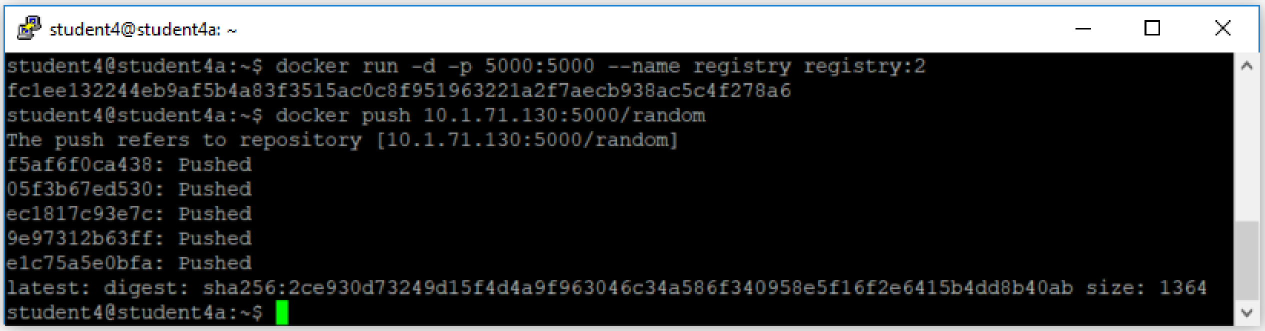
<registry-container-id> is the ID of the container with the image registry:2 (1 in the image)



```
student2@student2a: ~
12bb011aa22c        registry:2          "/entrypoint.sh /etc…"   20 hours ago        Exited (2) 14 seconds ago                                                registry
e6187f71f57a        rancher/rancher     "entrypoint.sh"          20 hours ago        Up 13 seconds               0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp   priceless_solom
on
student2@student2a:~$ sudo docker start ^C
student2@student2a:~$ sudo docker start 12bb011aa22c
12bb011aa22c
student2@student2a:~$
```

Now you can push the image to your registry

```
docker push <ip-address-of-first-vm>:5000/random
```



```
student4@student4a: ~
student4@student4a:~$ docker run -d -p 5000:5000 --name registry registry:2
fc1ee132244eb9af5b4a83f3515ac0c8f951963221a2f7aecb938ac5c4f278a6
student4@student4a:~$ docker push 10.1.71.130:5000/random
The push refers to repository [10.1.71.130:5000/random]
f5af6f0ca438: Pushed
05f3b67ed530: Pushed
ec1817c93e7c: Pushed
9e97312b63ff: Pushed
e1c75a5e0bfa: Pushed
latest: digest: sha256:2ce930d73249d15f4d4a9f963046c34a586f340958e5f16f2e6415b4dd8b40ab size: 1364
student4@student4a:~$
```

You can verify that you pushed it successfully by pulling it again

```
docker pull <ip-address-of-first-vm>:5000/random
```

Step 4: Pull the image onto your second VM

For the fourth and final step, we are going to connect to the registry on a second VM and our image. In a production environment, it is highly recommended to secure a registry using SSL and an authentication mechanism. This would exceed the scope of this lab guide, so we are going to ignore security concerns in this guide. You can find further reading on this topic below

First, log into your second VM.

We also need to instruct docker on the second VM to use http.

Edit the docker daemon configuration file:

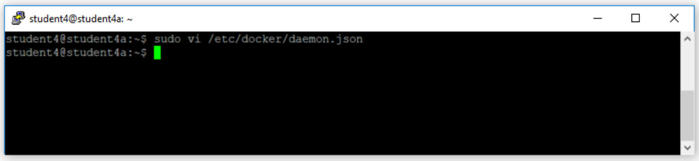
```
sudo vi /etc/docker/daemon.json
```

Paste the following configuration snippet into the config file

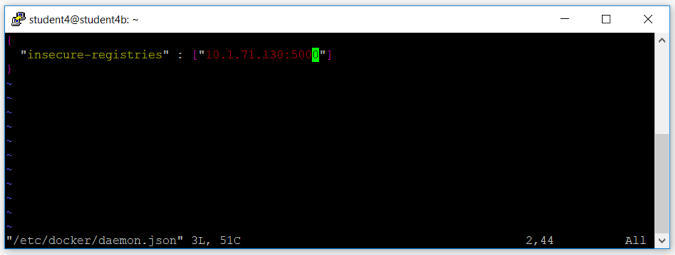
Don’t forget to switch to insert mode

Replace <first-vm-ip-address> with the IP-Address of your first VM.

```
{
  "insecure-registries": ["<first-vm-ip-address>:5000"]
}
```



```
student4@student4a:~$ sudo vi /etc/docker/daemon.json
student4@student4a:~$
```



```
{
  "insecure-registries" : ["10.1.71.130:5000"]
}
~
~
~
~
~
~
~
~
~
~
~
"/etc/docker/daemon.json" 3L, 51C                              2,44          All
```

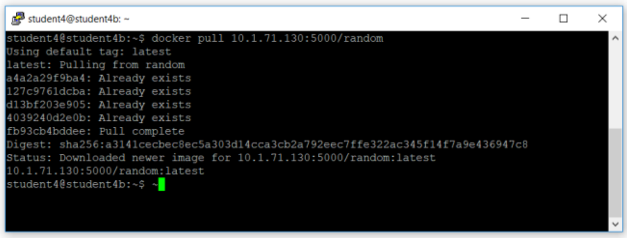
Save and exit.

Restart docker

```
sudo systemctl restart docker
```

Now you can pull the image.

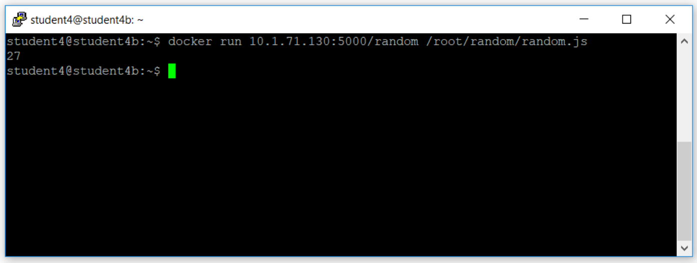
```
docker pull <first-vm-ip-address>:5000/random
```



```
student4@student4b:~$ docker pull 10.1.71.130:5000/random
Using default tag: latest
latest: Pulling from random
a4a2a29f9ba4: Already exists
127c9761dcba: Already exists
d13bf203e905: Already exists
4039240d2e0b: Already exists
fb93cb4bddee: Pull complete
Digest: sha256:a3141cecbec8ec5a303d14cca3cb2a792eec7ffe322ac345f14f7a9e436947c8
Status: Downloaded newer image for 10.1.71.130:5000/random:latest
10.1.71.130:5000/random:latest
student4@student4b:~$ ~
```

And run it

```
docker run <first-vm-ip-address>:5000/random /root/random/random.js
```



```
student4@student4b:~$ docker run 10.1.71.130:5000/random /root/random/random.js
27
student4@student4b:~$
```

Congratulations! You have successfully created and used your own registry

A guide on setting up a private registry while considering security measures:

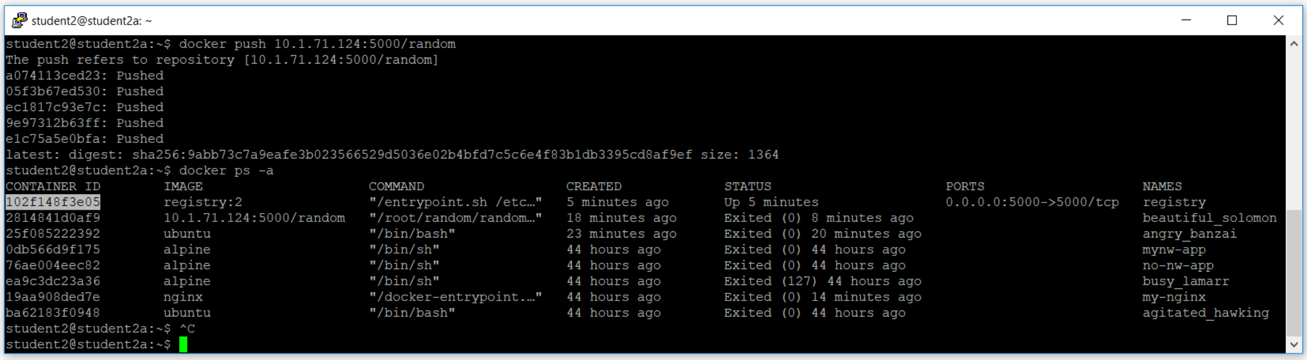
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04

Clean up Containers

Now we need to revert everything back to its original state in order for the next Lab to work. Let's stop the registry container to free up port 5000.

Check which container id the registry has.

```
docker ps -a
```



```
student2@student2a:~$ docker push 10.1.71.124:5000/random
The push refers to repository [10.1.71.124:5000/random]
a074113ced23: Pushed
05f3b67ed530: Pushed
ec1817c93e7c: Pushed
9e97312b63ff: Pushed
e1c75a5e0bfa: Pushed
latest: digest: sha256:9abb73c7a9eafe3b023566529d5036e02b4bfd7c5c6e4f83b1db3395cd8af9ef size: 1364
student2@student2a:~$ docker ps -a
CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS                       PORTS                    NAMES
102f148f3e05        registry:2                "/entrypoint.sh /etc…"   5 minutes ago       Up 5 minutes                 0.0.0.0:5000->5000/tcp   registry
2814841d0af9        10.1.71.124:5000/random   "/root/random/random…"   18 minutes ago      Exited (0) 8 minutes ago                              beautiful_solomon
25f085222392        ubuntu                    "/bin/bash"              23 minutes ago      Exited (0) 20 minutes ago                             angry_banzai
0db566d9f175        alpine                    "/bin/sh"                44 hours ago        Exited (0) 44 hours ago                               mynw-app
76ae004eec82        alpine                    "/bin/sh"                44 hours ago        Exited (0) 44 hours ago                               no-nw-app
ea9c3dc23a36        alpine                    "/bin/sh"                44 hours ago        Exited (127) 44 hours ago                             busy_lamarr
19aa908ded7e        nginx                     "/docker-entrypoint.…"   44 hours ago        Exited (0) 14 minutes ago                             my-nginx
ba62183f0948        ubuntu                    "/bin/bash"              44 hours ago        Exited (0) 44 hours ago                               agitated_hawking
student2@student2a:~$ ^C
student2@student2a:~$
```

Then let's stop the registry container:

```
docker stop <CONTAINER ID>
```

Bonus: Working with Dockerfiles

Now we will create a container for the same Node.js application, but this time "programmatically", using a **Dockerfile**.

Start with the Node.js program, login into your Lab VM.

```
cd
```

```
cd average
```

Both files should be already there, just have a look at these files.

```
cat average.js
```

```
var sum = 0;
var count = 0;
process.argv.forEach(function(val, index, array){

        if (index > 1) {
            sum += parseInt(val);
            count++;
        }
});
console.log(sum / count);
```

Then create a Dockerfile

```
cat Dockerfile
```

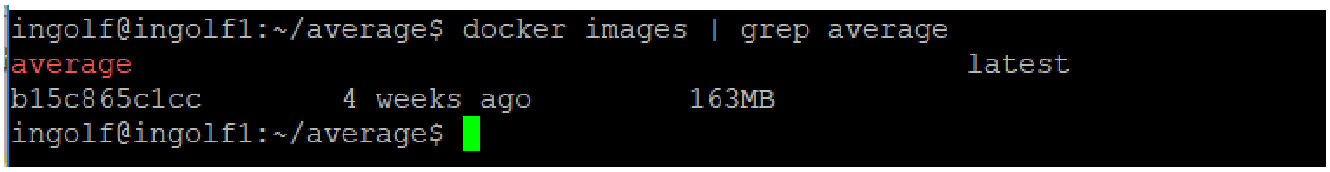
```
FROM ubuntu
Maintainer yourName
RUN apt-get update && apt-get install -y nodejs
RUN mkdir average
ADD average.js average/
WORKDIR average
ENTRYPOINT ["nodejs","average.js"]
```

Now create the image from the Dockerfile

```
docker build -t average .
```

Check the newly created image

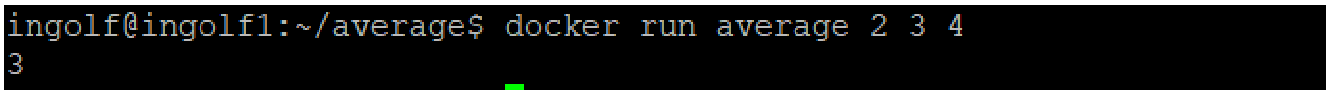
```
docker images | grep average
```



```
ingolf@ingolf1:~/average$ docker images | grep average
average                                                 latest
b15c865c1cc         4 weeks ago         163MB
ingolf@ingolf1:~/average$
```

And finally run the newly created image

```
docker run average 2 3 4
```



```
ingolf@ingolf1:~/average$ docker run average 2 3 4
3
```

Cool!! We created our first customized image using a Dockerfile.

Appendix: Useful Linux Commands

Navigate through directories using the bash shell

While the windows explorer provides a great and easy to use interface for file manipulation, navigating through directories using a bash shell can also be useful

Introduction

Files paths in Linux are relative to one common root. Unlike in windows, there are no explicit drive letters. Instead, any removable media is mounted to a folder somewhere inside the root folder. You can then access its content just like the contents of a normal folder.
One guiding principle of Linux is that as much as possible of the system configuration should be stored in easily accessible plain text files. Therefore, you won't find things like the

Using sudo

Sudo is an abbreviation for Super User DO. It allows you to execute any command with the rights of the superuser, or more commonly called root user. The root user has permission to do anything on the system.

Common Linux file structure

Linux is based on UNIX and therefore has a similar file structure as other UNIX-based systems. Some of the most important folders are explained in the following:

- /
  This is the root folder. Everything is located inside this folder.
  - /home – user directories
    Inside the home folder, there is a folder for each user with the same name as the username of the user. When you connect to a Linux server, you will probably start in your home directory (e.g. /home/student4)
  - /etc – configuration files
    Docker, Nginx and many other applications store their global
  - /root – The home directory of the root user
    The admin user in Linux is called root. Its home directory is not located in /home/root but in /root. The home directory of the root user (/root) should not be confused with the root directory (/)

Paths in Linux

Paths determine the location of a file or folder. In Linux, paths consist of a series of steps, each separated by a forward slash / . They can be either relative or absolute. All absolute paths begin with a / (e.g. /home/user/test.txt). They provide the instruction on how to get from to root directory to the specified file or directory.

Relative paths, on the other hand, do not begin with a /. They describe the series of steps taken from the current directory to the specified file or directory. The current directory is the directory that the user or the script is currently in.

The step “/../” means to go back up the hierarchy by one step.

The absolute path /home/user/test.txt can be specified in multiple ways: “user/test.txt”, if the user is currently in “/home”. “../user/test.txt”, if the user is currently in “/home/user2”.

Navigating through folders

You can navigate to a folder by using **cd <path>** where path can be either relative or absolute.

CD is an abbreviation for change directory. If you use cd without providing a path, you will go to the user’s directory

Inspecting folders

You can view the contents of a folder by using the command **ls <path>**.

LS is an abbreviation for list. Using ls without a path will list the contents of the current directory

The command ls -a lists all files even including system files beginning with a dot.

Creating folders

You can create a folder using **mkdir <folder-name>**

Removing folders

rmdir <folder-name>
To remove a directory, it must be empty

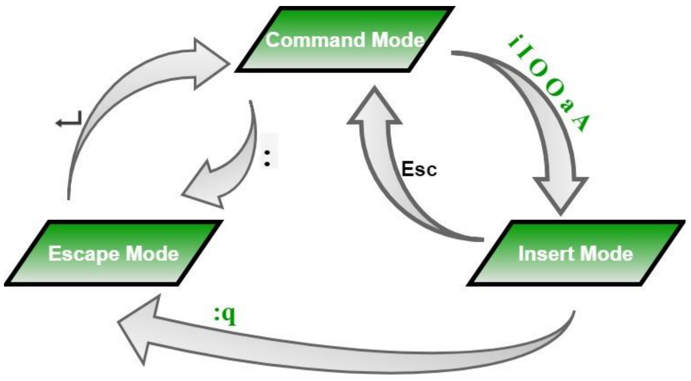
Removing files and folders
You can remove a file using **rm <file-name>**
rm -r <file-name> deletes a folder and its contents.

Appendix: How to edit files using Vi

Vi is a text editor commonly used on Linux. It does not provide formatting options but as only the following three functionalities: Edit, navigate in and save files.

The three modes in Vi

Vi can be used in three modes:



Command mode:
When you open a file in Vi, the editor is first in command mode. This mode allows you to issue commands to navigate inside the file or change its contents:

Command	Explanation
Arrow key or h, j , k, l	Navigate inside the file (e.g. jump one character to the left by pressing the left arrow key)
w, b	Jump back and forth by one word
x, dw, dd	Remove the current character, word or line
u	Undo the last action

Insert mode

The insert mode allows you to type characters into the file just like in any other editor. You can exit the insert mode by pressing the escape key.

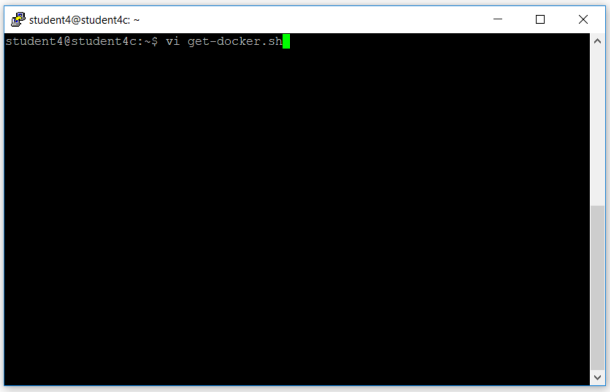
Escape Mode

The escape/exit mode is used to quit the editor.

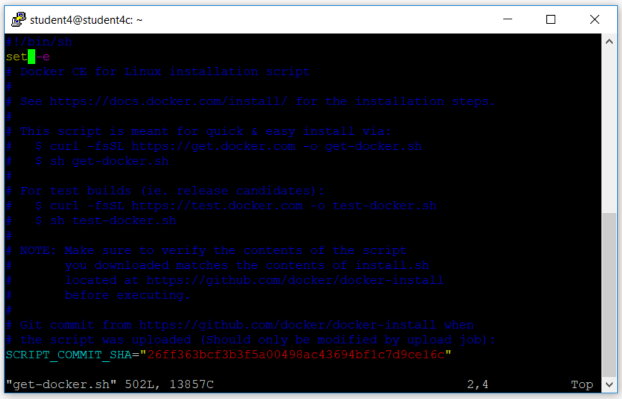
Command	Explanation
:w	Save the file
:q	Quit vi
:q!	Quit without saving
:wq	Save and then quit
:w <file-name>	Save the file with a new filename

Open files in Vi

Type **vi <file-name>** to open a file in vi. Replace <file-name> with the name of the file you want to open.



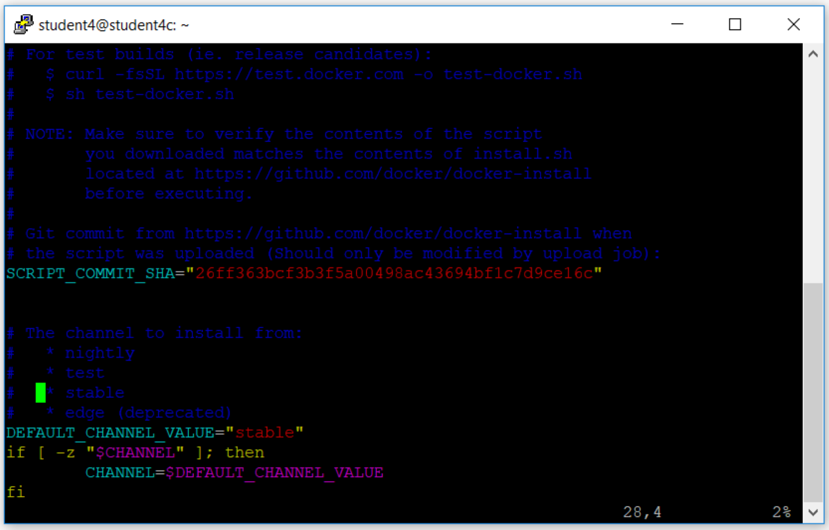
```
student4@student4c:~$ vi get-docker.sh
```



```
#!/bin/sh
set -e
# Docker CE for Linux installation script
#
# See https://docs.docker.com/install/ for the installation steps.
#
# This script is meant for quick & easy install via:
#   $ curl -fsSL https://get.docker.com -o get-docker.sh
#   $ sh get-docker.sh
#
# For test builds (ie. release candidates):
#   $ curl -fsSL https://test.docker.com -o test-docker.sh
#   $ sh test-docker.sh
#
# NOTE: Make sure to verify the contents of the script
#       you downloaded matches the contents of install.sh
#       located at https://github.com/docker/docker-install
#       before executing.
#
# Git commit from https://github.com/docker/docker-install when
# the script was uploaded (Should only be modified by upload job):
SCRIPT_COMMIT_SHA="26ff363bcf3b3f5a00498ac43694bf1c7d9ce16c"

"get-docker.sh" 502L, 13857C                                2,4           Top
```

Navigate inside files

You can navigate in the file by using the arrow keys.

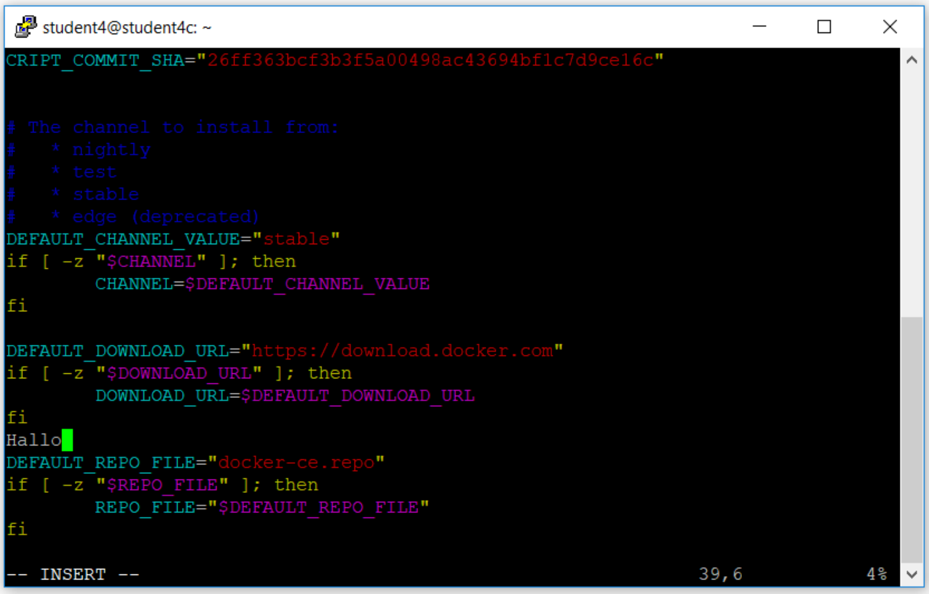


```
# For test builds (ie. release candidates):
#   $ curl -fsSL https://test.docker.com -o test-docker.sh
#   $ sh test-docker.sh
#
# NOTE: Make sure to verify the contents of the script
#       you downloaded matches the contents of install.sh
#       located at https://github.com/docker/docker-install
#       before executing.
#
# Git commit from https://github.com/docker/docker-install when
# the script was uploaded (Should only be modified by upload job):
SCRIPT_COMMIT_SHA="26ff363bcf3b3f5a00498ac43694bf1c7d9ce16c"


# The channel to install from:
#   * nightly
#   * test
#   * stable
#   * edge (deprecated)
DEFAULT_CHANNEL_VALUE="stable"
if [ -z "$CHANNEL" ]; then
        CHANNEL=$DEFAULT_CHANNEL_VALUE
fi
                                                              28,4          2%
```

Edit files

When you open vim, you are first in normal mode. This mode is used to navigate in the file and issue commands. To edit files, you first have to switch to insert mode by pressing the **I** key.

You can insert text from your clipboard in Putty by pressing the right mouse key.


student4@student4c: ~

```
CRIPT_COMMIT_SHA="26ff363bcf3b3f5a00498ac43694bf1c7d9ce16c"


# The channel to install from:
#   * nightly
#   * test
#   * stable
#   * edge (deprecated)
DEFAULT_CHANNEL_VALUE="stable"
if [ -z "$CHANNEL" ]; then
        CHANNEL=$DEFAULT_CHANNEL_VALUE
fi

DEFAULT_DOWNLOAD_URL="https://download.docker.com"
if [ -z "$DOWNLOAD_URL" ]; then
        DOWNLOAD_URL=$DEFAULT_DOWNLOAD_URL
fi
Hallo
DEFAULT_REPO_FILE="docker-ce.repo"
if [ -z "$REPO_FILE" ]; then
        REPO_FILE="$DEFAULT_REPO_FILE"
fi

-- INSERT --                                             39,6          4%
```

Save files

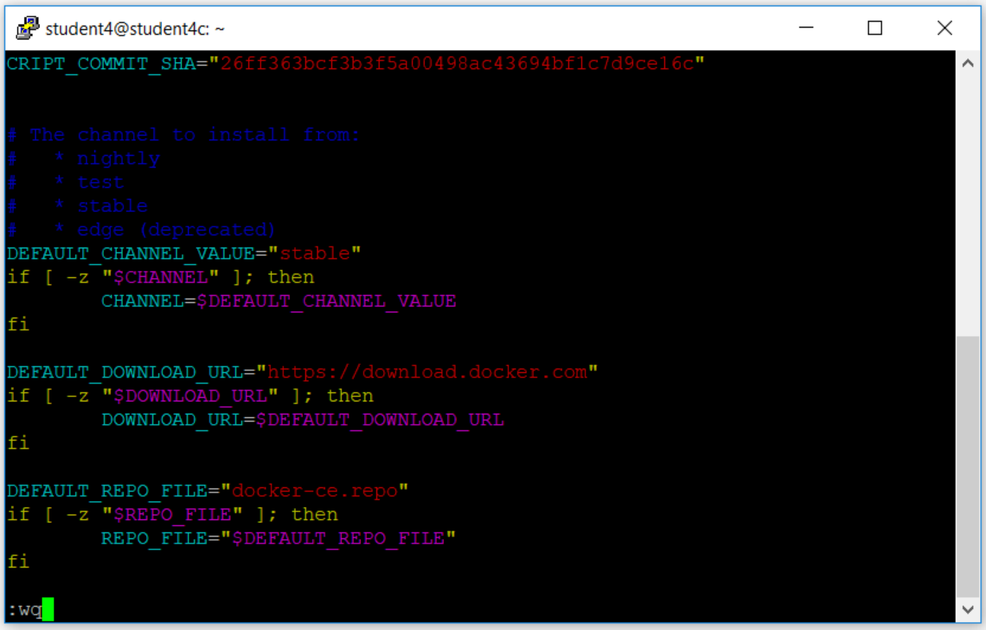
To save files you first have to switch back to normal mode. You can do that by pressing the **Escape key**.

Then you can issue commands by typing **:**

To save type **w** and press the enter key.

To quit type **q**.

You can combine commands by typing multiple letters. E.g. type **:wq** to save and quit.


student4@student4c: ~

```
CRIPT_COMMIT_SHA="26ff363bcf3b3f5a00498ac43694bf1c7d9ce16c"


# The channel to install from:
#   * nightly
#   * test
#   * stable
#   * edge (deprecated)
DEFAULT_CHANNEL_VALUE="stable"
if [ -z "$CHANNEL" ]; then
        CHANNEL=$DEFAULT_CHANNEL_VALUE
fi

DEFAULT_DOWNLOAD_URL="https://download.docker.com"
if [ -z "$DOWNLOAD_URL" ]; then
        DOWNLOAD_URL=$DEFAULT_DOWNLOAD_URL
fi

DEFAULT_REPO_FILE="docker-ce.repo"
if [ -z "$REPO_FILE" ]; then
        REPO_FILE="$DEFAULT_REPO_FILE"
fi

:wq
```

Further Reading

https://www.dummies.com/web-design-development/web-hosting/how-to-edit-files-with-vi/