Containerization.
Docker.
Lection 1.

Docker overview. Docker platform.

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 overview. Docker platform.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don’t need the extra load of a hypervisor, but run directly within the host machine’s kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

Docker provides tooling and a platform to manage the lifecycle of your containers:

Develop your application and its supporting components using containers.
The container becomes the unit for distributing and testing your application.
When you’re ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

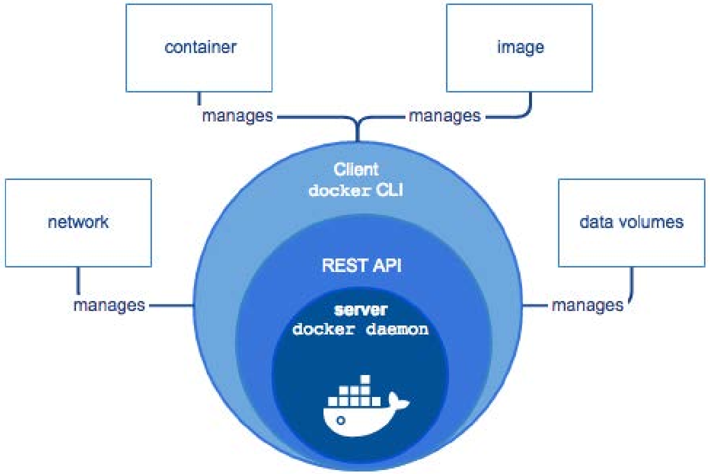
Docker Engine.

Docker Engine is a client-server application with these major components:

A server which is a type of long-running program called a daemon process (the ***dockerd*** command).

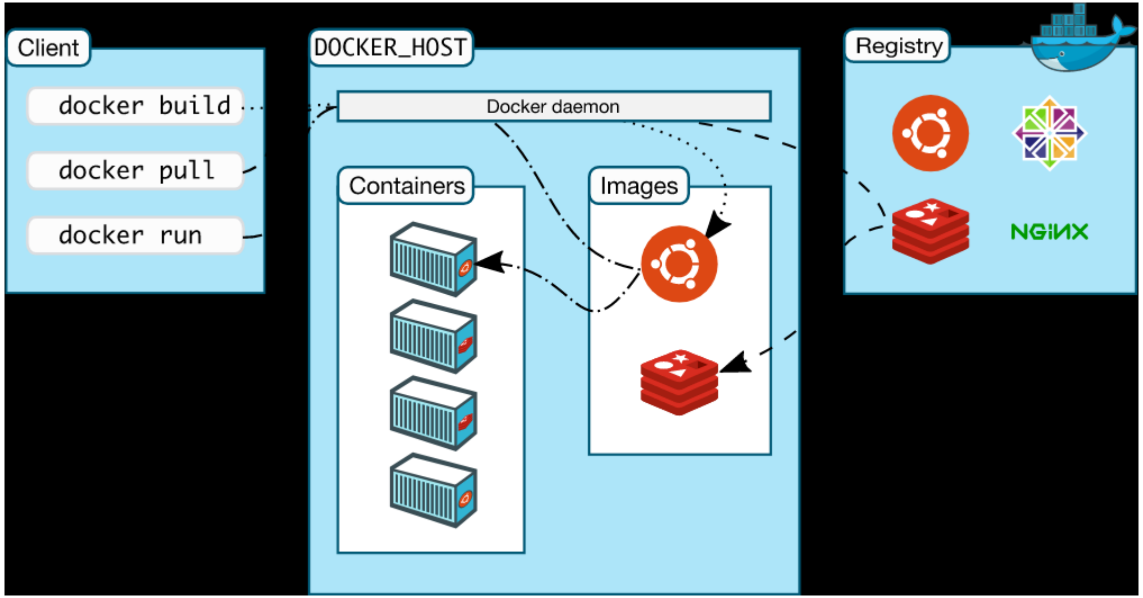
A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.

A command line interface (CLI) client
(the ***docker*** command).



Docker architecture.

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



https://docs.docker.com/get-started/overview/

Docker architecture.

The Docker daemon
The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as ***images***, ***containers***, ***networks***, and ***volumes***. A daemon can also communicate with other daemons to manage Docker services.

The Docker client
The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to ***dockerd***, which carries them out. The *docker* command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries
A Docker ***registry*** stores Docker ***images***. Docker Hub is a public *registry* that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private *registry*.

When you use the ***docker pull*** or ***docker run*** commands, the required images are pulled from your configured ***registry***. When you use the *docker push* command, your image is pushed to your configured registry.

https://docs.docker.com/get-started/overview/

Docker architecture.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

IMAGES

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Examples of *docker run* command

The following command runs an ***ubuntu*** container, attaches interactively to your local command-line session, and runs ***/bin/bash***.

$ docker run -i -t ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

If you do not have the ***ubuntu*** image locally, Docker pulls it from your configured registry, as though you had run ***docker pull ubuntu*** manually.

Docker creates a new container, as though you had run a ***docker container create*** command manually.

Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.

Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.

Docker starts the container and executes ***/bin/bash***. Because the container is running interactively and attached to your terminal (due to the ***-i*** and ***-t*** flags), you can provide input using your keyboard while the output is logged to your terminal.

When you type ***exit*** to terminate the ***/bin/bash*** command, the container stops but is not removed. You can start it again or remove it.

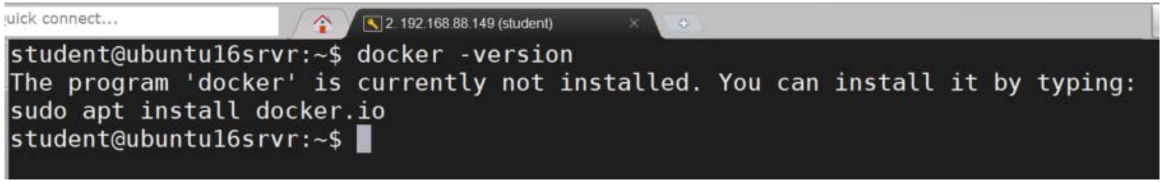
Docker practice basics

- How to install Docker
- How to use Docker Image
- How to run Docker Container
- What is Dockerfile
- What is DockerHub
- How to build Docker Image from Dockerfile
- How to create an updated Docker Image from Docker Container
- All basic Docker commands

Docker practice basics

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

How to install Docker



```
student@ubuntu16srvr:~$ docker -version
The program 'docker' is currently not installed. You can install it by typing:
sudo apt install docker.io
student@ubuntu16srvr:~$
```

Prerequisites

OS requirements

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

Ubuntu Focal 20.04 (LTS)

Ubuntu Bionic 18.04 (LTS)

Ubuntu Xenial 16.04 (LTS)

Docker Engine is supported on x86_64 (or amd64), armhf, and arm64 architectures.

Uninstall old versions

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

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

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

The contents of ***/var/lib/docker/***, including images, containers, volumes, and networks, are preserved. If you do not need to save your existing data, and want to start with a clean installation, refer to the uninstall Docker Engine section at the bottom of this page.

Docker practice basics

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

Installation methods

You can install Docker Engine in different ways, depending on your needs:

Most users set up Docker's repositories and install from them, for ease of installation and upgrade tasks. This is the recommended approach.

Some users download the DEB package and install it manually and manage upgrades completely manually. This is useful in situations such as installing Docker on air-gapped systems with no access to the internet.

In testing and development environments, some users choose to use automated convenience scripts to install Docker.

Docker practice basics

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

Install using the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

SET UP THE REPOSITORY

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

```
$ sudo apt-get update

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

Docker practice basics

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

2. Add Docker’s official GPG key:

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

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

```
$ sudo apt-key fingerprint 0EBFCD88
```

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

Docker practice basics

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

Use the following command to set up the **stable repository**.
To add the nightly or test repository, add the word nightly or test (or both) after the word stable in the commands below. Learn about nightly and test channels.

!!! The ***lsb_release -cs*** sub-command below returns the name of your Ubuntu distribution, such as xenial. Sometimes, in a distribution like Linux Mint, you might need to change $(lsb_release -cs) to your parent Ubuntu distribution. For example, if you are using Linux Mint Tessa, you could use bionic. Docker does not offer any guarantees on untested and unsupported Ubuntu distributions.

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

Docker practice basics

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

INSTALL DOCKER ENGINE

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

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

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

Docker practice basics

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

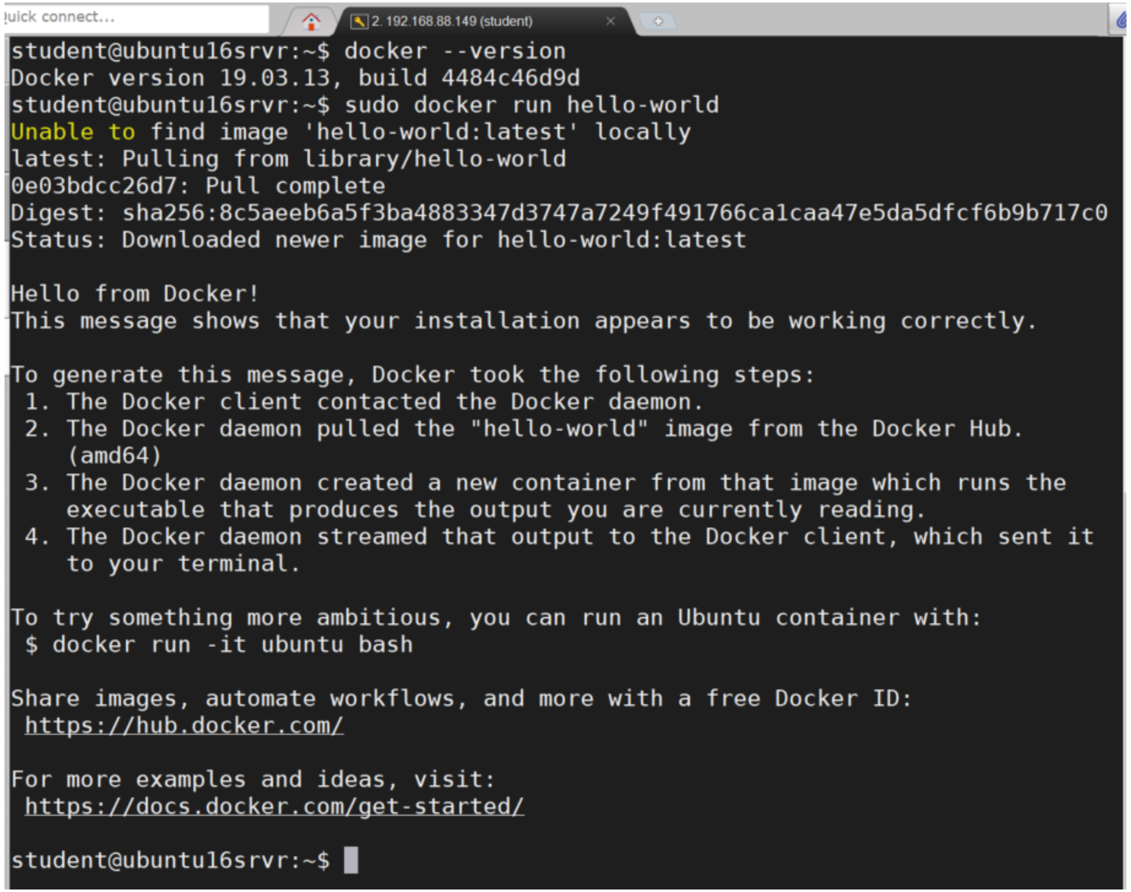
INSTALL DOCKER ENGINE

Verify that Docker Engine
is installed correctly
by running the
hello-world image.

$ sudo docker run hello-world

This command downloads a ***test image and runs it in a container***.

When the container runs, it prints
an informational message and exits.



```
student@ubuntu16srvr:~$ docker --version
Docker version 19.03.13, build 4484c46d9d
student@ubuntu16srvr:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:8c5aeeb6a5f3ba4883347d3747a7249f491766ca1caa47e5da5dfcf6b9b717c0
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/

student@ubuntu16srvr:~$
```

Docker practice basics

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

Docker Engine is installed and running.
The ***docker*** group is created but ***no users*** are added to it.
You need to use ***sudo*** to run Docker commands.
Post-installation
There are some steps for Linux to ***allow non-privileged users to run Docker commands*** and for other optional configuration steps.
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 student
```

Remember to log out and back in for this to take effect!

Warning:
Adding a user to the "docker" group grants them the ability to run containers which can be used to obtain root privileges on the Docker host. Refer to ***Docker Daemon Attack Surface*** *for more information.*

Docker practice basics (Ubuntu 16 image)

Create directory for Dockerfile(-s) and and dive into it.

```
$ mkdir dockerfiles
$ cd dockerfiles
```

Create your file in that directory:

```
#$ touch Dockerfile
```

Edit it and add the commands with nano:

```
$ nano Dockerfile
```

Finally build it:

```
$ docker build -t tag .
```

```
FROM ubuntu:16.04
RUN apt-get -y update
RUN apt-get -y install apache2
RUN echo 'Hi there, what is love?' > /var/www/html/index.html
RUN echo 'It is just a song ...' > /var/www/html/index.html
CMD ["/usr/sbin/apache2ctl", "-DFOREGROUND"]
EXPOSE 80
```

Docker practice basics

Create directory for Dockerfile(-s) and and dive into it.

$ mkdir dockerfiles

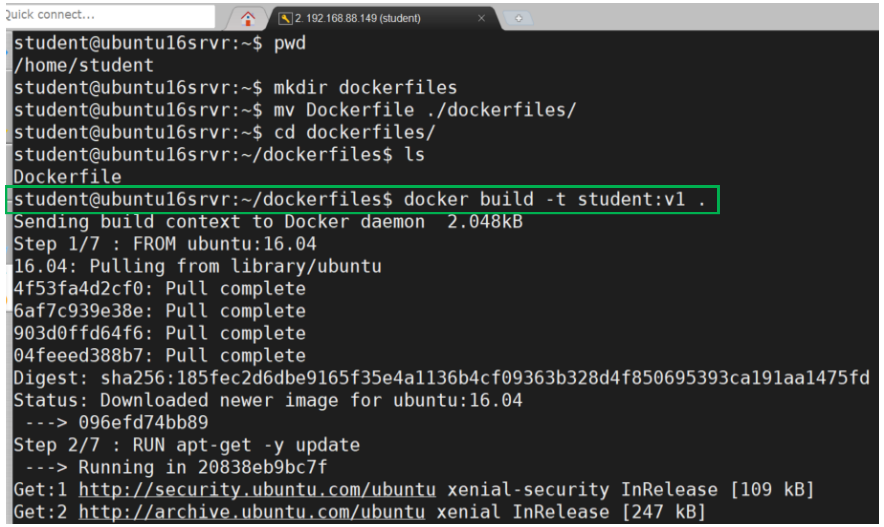
$ cd dockerfiles

Edit it and add the commands with nano:

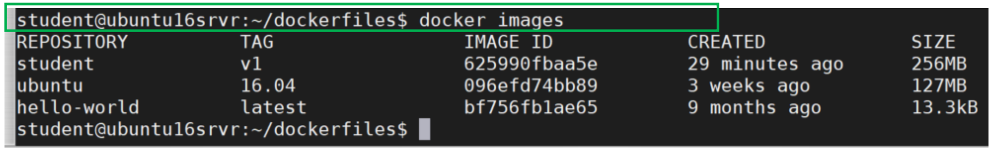
$ nano Dockerfile

Finally build it:

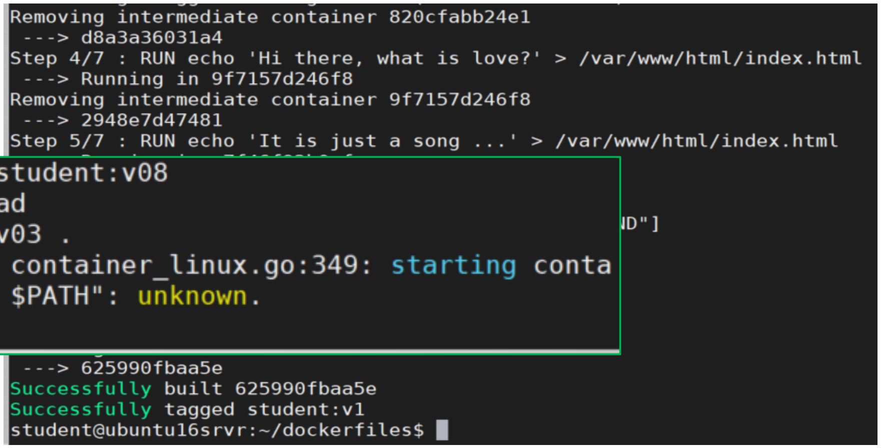
$ docker build -t tag .



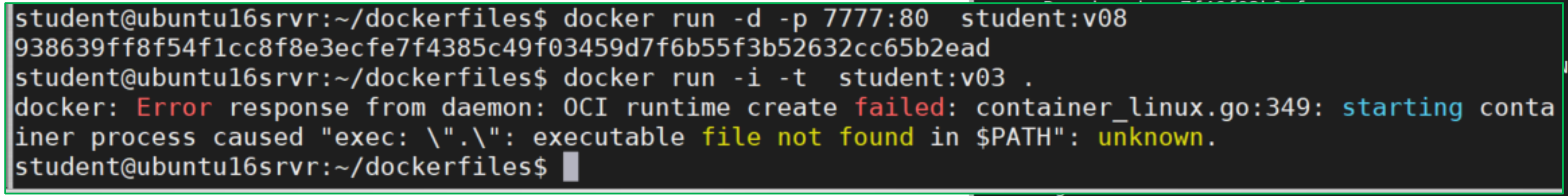
```
student@ubuntu16srvr:~$ pwd
/home/student
student@ubuntu16srvr:~$ mkdir dockerfiles
student@ubuntu16srvr:~$ mv Dockerfile ./dockerfiles/
student@ubuntu16srvr:~$ cd dockerfiles/
student@ubuntu16srvr:~/dockerfiles$ ls
Dockerfile
student@ubuntu16srvr:~/dockerfiles$ docker build -t student:v1 .
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
4f53fa4d2cf0: Pull complete
6af7c939e38e: Pull complete
903d0ffd64f6: Pull complete
04feeed388b7: Pull complete
Digest: sha256:185fec2d6dbe9165f35e4a1136b4cf09363b328d4f850695393ca191aa1475fd
Status: Downloaded newer image for ubuntu:16.04
 ---> 096efd74bb89
Step 2/7 : RUN apt-get -y update
 ---> Running in 20838eb9bc7f
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
```



```
student@ubuntu16srvr:~/dockerfiles$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
student             v1                  625990fbaa5e        29 minutes ago      256MB
ubuntu              16.04               096efd74bb89        3 weeks ago         127MB
hello-world         latest              bf756fb1ae65        9 months ago        13.3kB
student@ubuntu16srvr:~/dockerfiles$
```



```
Removing intermediate container 820cfabb24e1
 ---> d8a3a36031a4
Step 4/7 : RUN echo 'Hi there, what is love?' > /var/www/html/index.html
 ---> Running in 9f7157d246f8
Removing intermediate container 9f7157d246f8
 ---> 2948e7d47481
Step 5/7 : RUN echo 'It is just a song ...' > /var/www/html/index.html

ND"]

 ---> 625990fbaa5e
Successfully built 625990fbaa5e
Successfully tagged student:v1
student@ubuntu16srvr:~/dockerfiles$
```



```
student@ubuntu16srvr:~/dockerfiles$ docker run -d -p 7777:80  student:v08
938639ff8f54f1cc8f8e3ecfe7f4385c49f03459d7f6b55f3b52632cc65b2ead
student@ubuntu16srvr:~/dockerfiles$ docker run -i -t  student:v03 .
docker: Error response from daemon: OCI runtime create failed: container_linux.go:349: starting conta
iner process caused "exec: \".\": executable file not found in $PATH": unknown.
student@ubuntu16srvr:~/dockerfiles$
```

Docker practice basics

Create directory for Dockerfile(-s) and and dive into it.

```
$ mkdir dockerfiles
$ cd dockerfiles
```

Edit it and add the commands with nano:

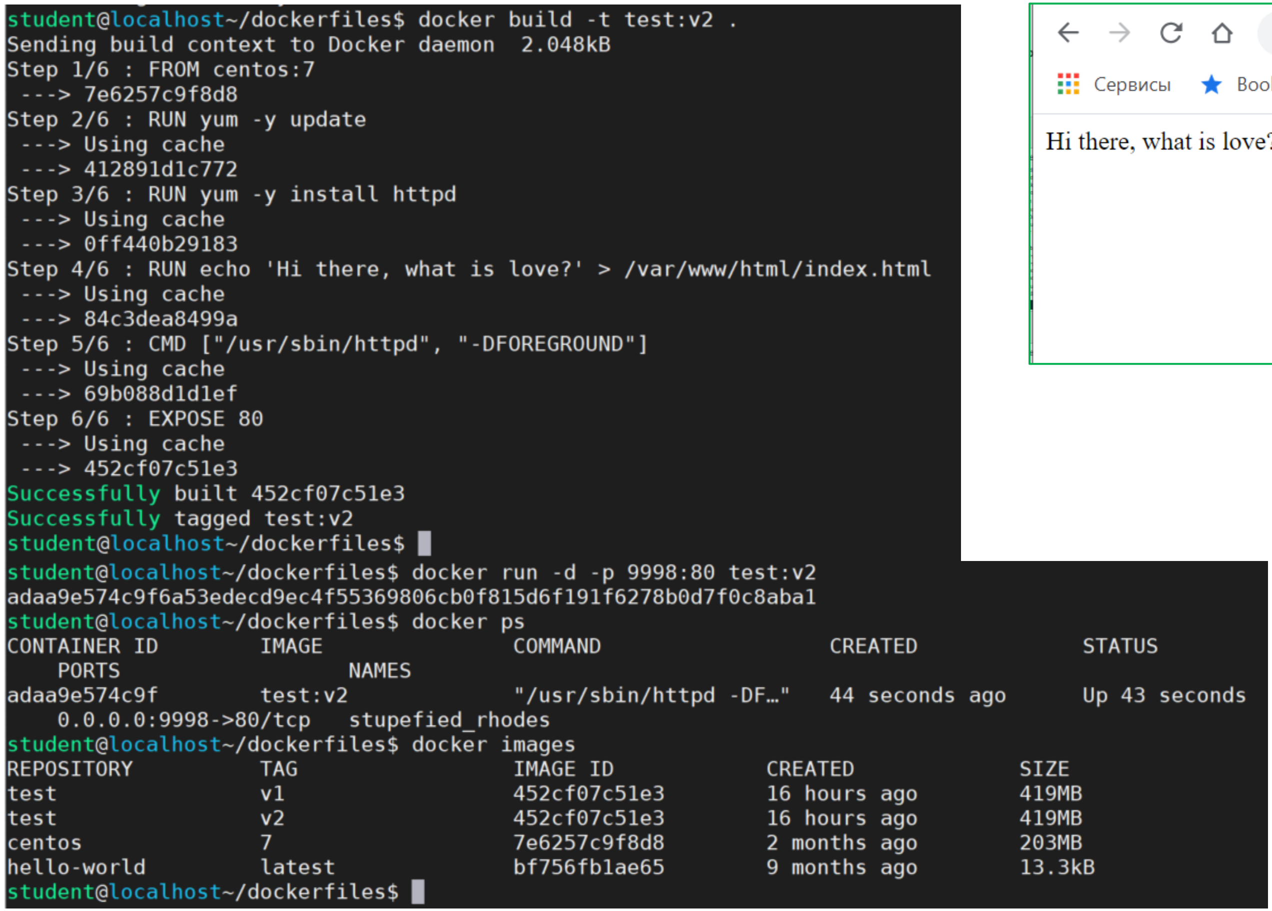
```
$ nano Dockerfile
```

Finally build it:

```
$ docker build -t tag .
```

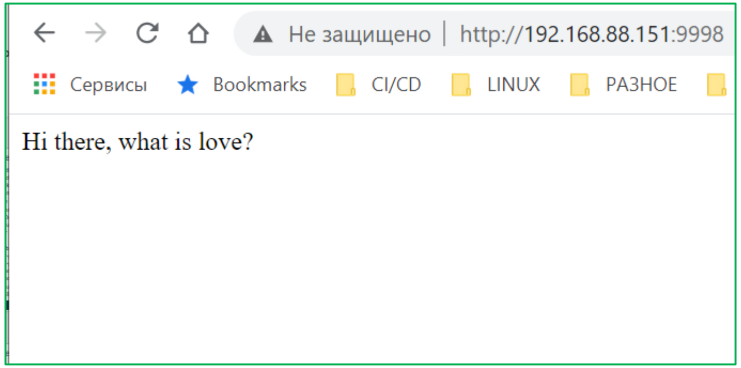
```
FROM centos:7

RUN yum -y update
RUN yum -y install httpd
RUN echo 'Hi there, what is love?' >
/var/www/html/index.html

CMD ["/usr/sbin/httpd", "-DFOREGROUND"]

EXPOSE 80
```

Docker practice basics



```
student@localhost~/dockerfiles$ docker build -t test:v2 .
Sending build context to Docker daemon  2.048kB
Step 1/6 : FROM centos:7
 ---> 7e6257c9f8d8
Step 2/6 : RUN yum -y update
 ---> Using cache
 ---> 412891d1c772
Step 3/6 : RUN yum -y install httpd
 ---> Using cache
 ---> 0ff440b29183
Step 4/6 : RUN echo 'Hi there, what is love?' > /var/www/html/index.html
 ---> Using cache
 ---> 84c3dea8499a
Step 5/6 : CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
 ---> Using cache
 ---> 69b088d1d1ef
Step 6/6 : EXPOSE 80
 ---> Using cache
 ---> 452cf07c51e3
Successfully built 452cf07c51e3
Successfully tagged test:v2
student@localhost~/dockerfiles$
```

```
student@localhost~/dockerfiles$ docker run -d -p 9998:80 test:v2
adaa9e574c9f6a53edecd9ec4f55369806cb0f815d6f191f6278b0d7f0c8aba1
student@localhost~/dockerfiles$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
    PORTS                  NAMES
adaa9e574c9f        test:v2             "/usr/sbin/httpd -DF…"   44 seconds ago      Up 43 seconds
    0.0.0.0:9998->80/tcp   stupefied_rhodes
student@localhost~/dockerfiles$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
test                v1                  452cf07c51e3        16 hours ago        419MB
test                v2                  452cf07c51e3        16 hours ago        419MB
centos              7                   7e6257c9f8d8        2 months ago        203MB
hello-world         latest              bf756fb1ae65        9 months ago        13.3kB
student@localhost~/dockerfiles$
```



Hi there, what is love?

Docker practice basics

Purging All Unused or Dangling Images, Containers, Volumes, and Networks
Docker provides a single command that will clean up any resources — images, containers, volumes, and networks — that are dangling (not associated with a container):

docker system prune

To additionally remove any stopped containers and all unused images (not just dangling images), add the -a flag to the command:

docker system prune -a

Q&A

Thank you!