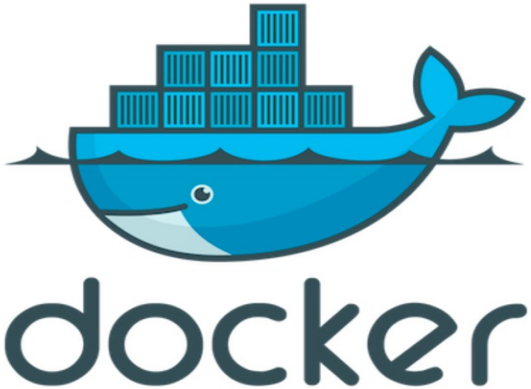

# Lab: Learning Dockers

University of CINCINNATI

## Dockers by Doing

Running





Containers

**Rashmi Kansakar**


# Lab: Learning Dockers

University of CINCINNATI

## Dockers by Doing



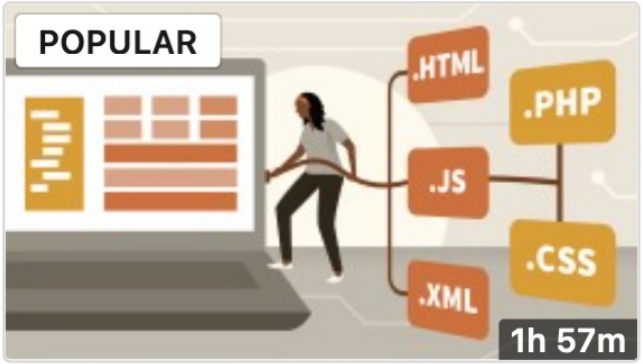

# Running Containers

**Rashmi Kansakar**

# Docker by Doing

University of CINCINNATI

**LEARNING OBJECTIVES:**




COURSE

## Learning Docker

LinkedIn · By: Carlos Nunez · 1 month ago

4.6 ★★★★½ (165) · 4,366 learners · Beginner

https://www.linkedin.com/learning/learning-docker-17236240

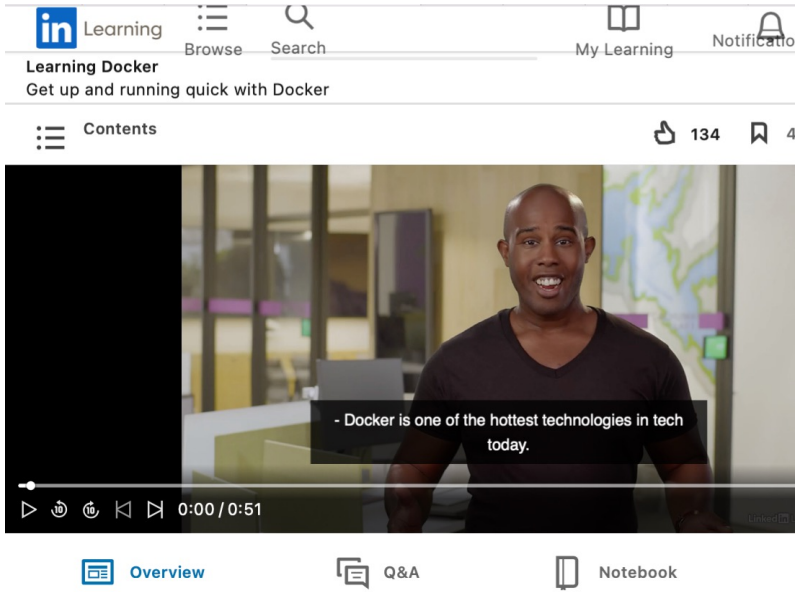
# Docker by Doing

University of CINCINNATI

**LEARNING OBJECTIVES:**

COURSE **Learning Docker**

By: Carlos Nunez




**Course details**

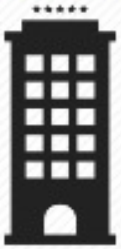
1h 57m
Beginner
Released: 12/14/2022

Docker is an open-source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system libraries and dependencies required to run that code in any environment. In this course, Carlos Nunez introduces the basics of Docker, including its containers, Dockerfiles (or base images), and capabilities. Watch and learn how to build your own containers.

https://www.linkedin.com/learning/learning-docker-2/what-is-docker?u=2133849

University of CINCINNATI

# Virtual Machines (Houses) vs Containers (Apartments)



- **Has its own infrastructure**
- Has more necessary things that make it a house, e.g:
  - Roof
  - At least one bedroom
  - Bathroom
  - Kitchen
  - Living area
  - Garage
  - Yard

- **Shares existing infrastructure**
- Comes in a variety of different setups:
  - Studio / 2 br / penthouse
  - Kitchen vs kitchenette
  - Living area?
  - Parking space?
  - Balcony?

| **Virtual Machine** | **Container** |
|---|---|
| Isolate address space | Isolate address space |
| isolate files and networks | isolate files and networks |
| Heavyweight | Lightweight |

## A Colony of Bungalows

On Premise Deployment:

1) Every app is hosted on a separate sever (bungalow)
2) No optimization of resources at all.
3) No shared services used.
4) High maintenance

## An Apartment Complex


Virtualization:

1) Some resources or amenities shared (hyper-visor, host -OS) by apps
2) Some amenities are still app specific (guest OS)
3) Moderate maintenance needed.

## A Hotel


Containerization:

1) Most amenities are shared
2) Guests have only a subset of amenities to use
3) Hotel has better control over managing resources

# What Docker Does

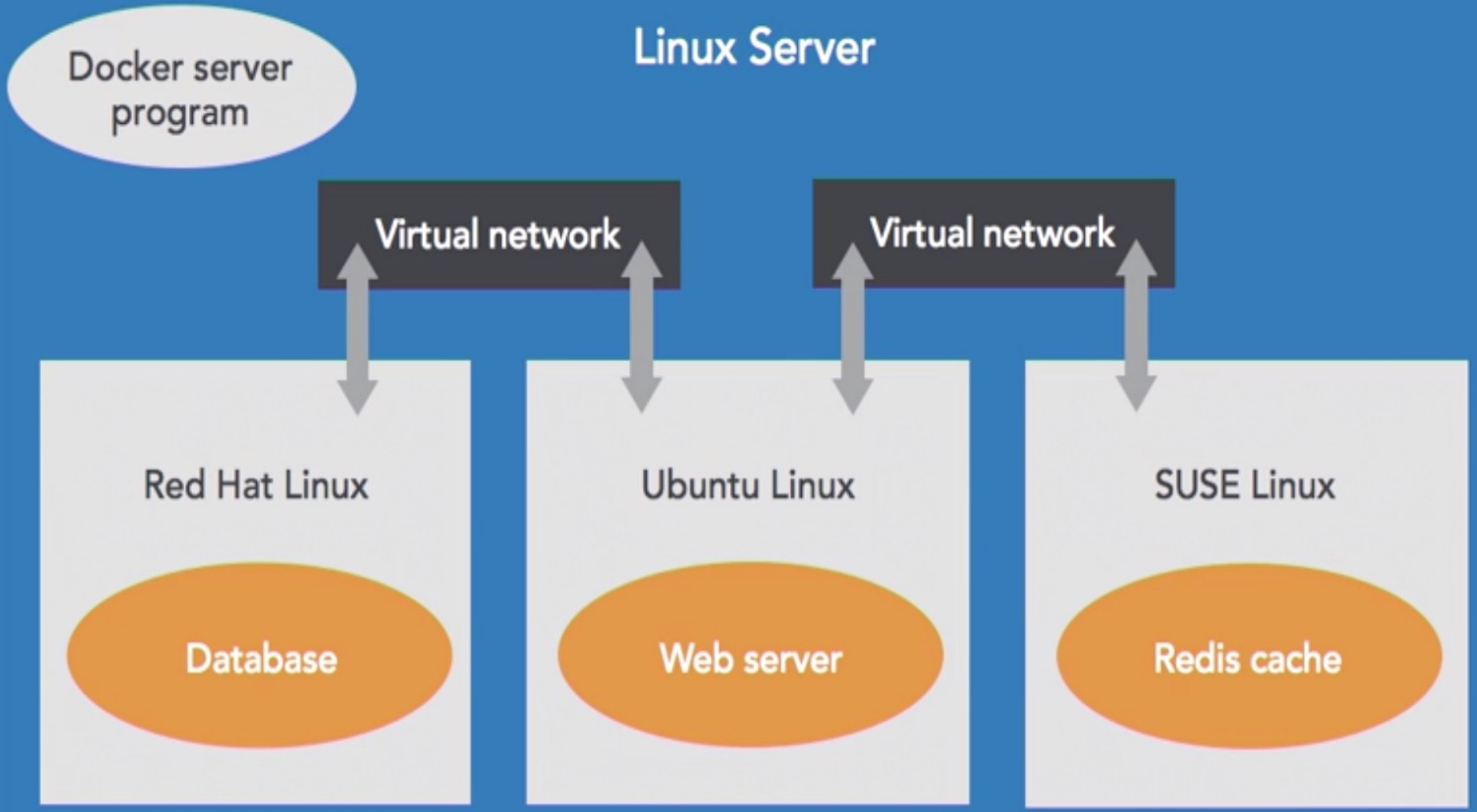
- Carves up a computer into sealed containers that run your code
- Gets the code to and from your computers
- Builds these containers for you
- Is a social platform for you to find and share containers, which are different from virtual machines

# What is Containers?

University of CINCINNATI

- A self-contained sealed unit of software
- Contains everything required to run the code
- Includes batteries and operating system

- A container includes
  - Code
  - Configs
  - Processes
  - Networking
  - Dependencies
  - Operating system


Docker server program
Linux Server
Virtual network
Virtual network
Red Hat Linux
Database
Ubuntu Linux
Web server
SUSE Linux
Redis cache

# What is Docker?

University of CINCINNATI

## About Docker

- A service that distributes containers
- A company that makes containers

- A client program named Docker
- A server program that manages a Linux system
- A program that builds containers from code

# Docker basics

University of CINCINNATI

**Image**

The basis of a Docker container. The content at rest.

**Container**

The image when it is 'running.' The standard unit for app service

**Engine**

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

**Registry**

Stores, distributes and manages Docker images

**Control Plane**

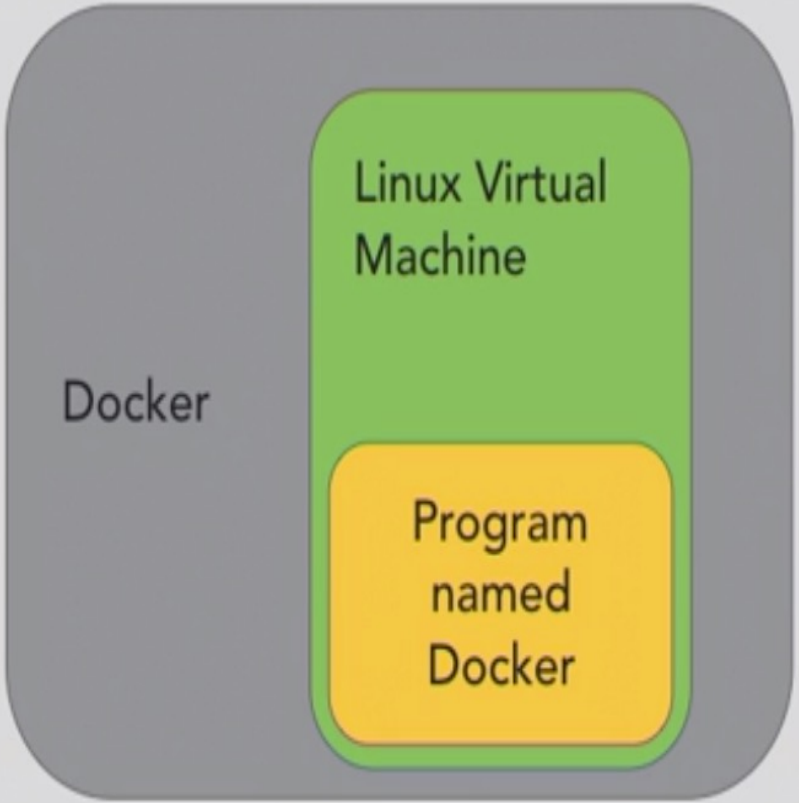
Management plane for container and cluster orchestration

# Installing Docker



- Docker needs a Linux server to manage
- Many people use a virtual machine on their laptop
- Docker helps run this Linux virtual machine
- If you have Boot2Docker installed, you must remove it first

Your Computer

# Docker Desktop

https://docs.docker.com/desktop/

*Estimated reading time: 2 minutes*

**Docker Desktop terms**

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) requires a paid subscription.

Docker Desktop is an easy-to-install application for your Mac, Linux, or Windows environment that enables you to build and share containerized applications and microservices.

It provides a simple interface that enables you to manage your containers, applications, and images directly from your machine without having to use the CLI to perform core actions.

What's included in Docker Desktop? | What are the key features of Docker Desktop?

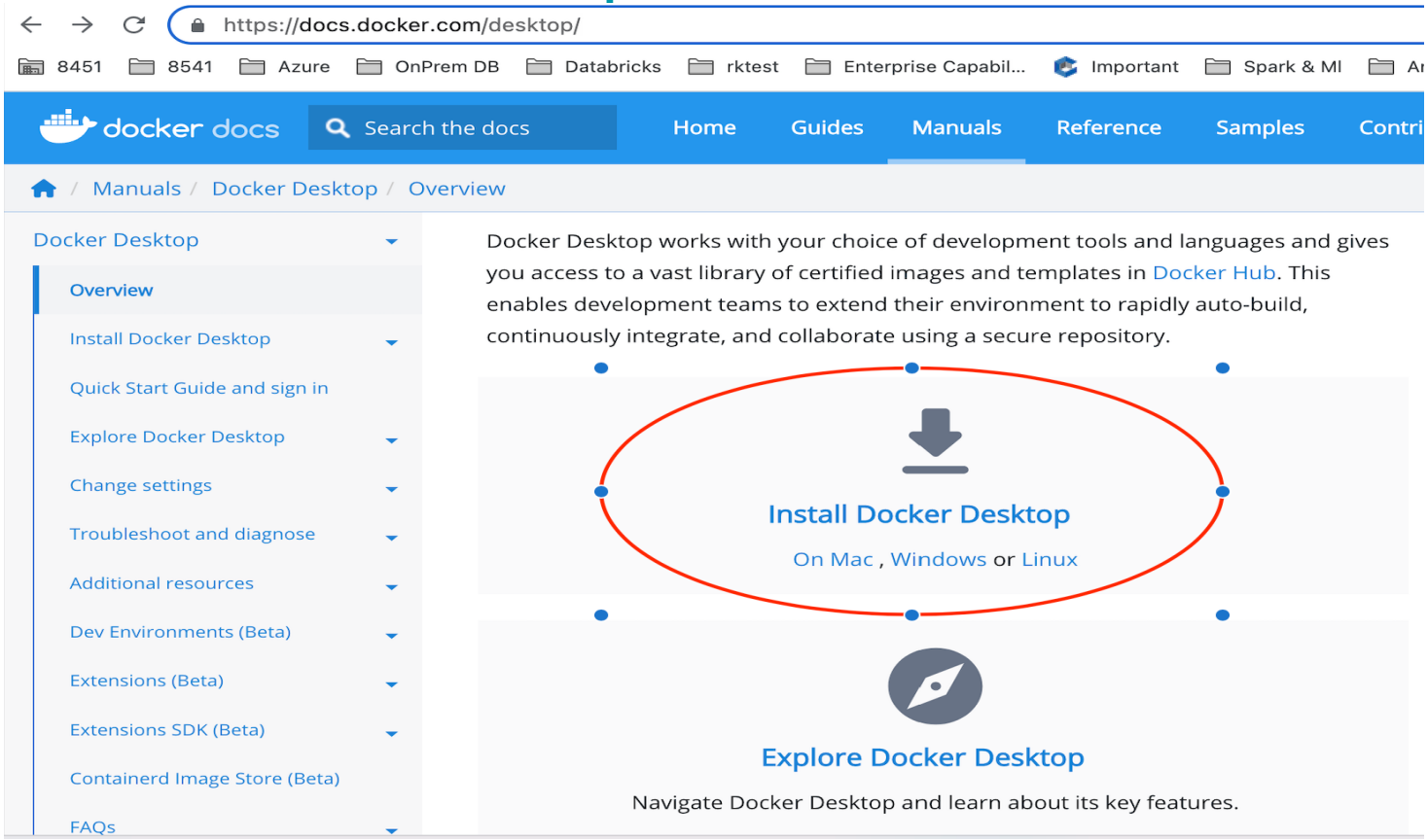
- Docker Engine
- Docker CLI client
- Docker Buildx
- Docker Compose
- Docker Content Trust
- Kubernetes
- Credential Helper

Docker Desktop works with your choice of development tools and languages and gives you access to a vast library of certified images and templates in Docker Hub. This enables development teams to extend their environment to rapidly auto-build, continuously integrate, and collaborate using a secure repository.

# Download Docker Desktop

University of CINCINNATI

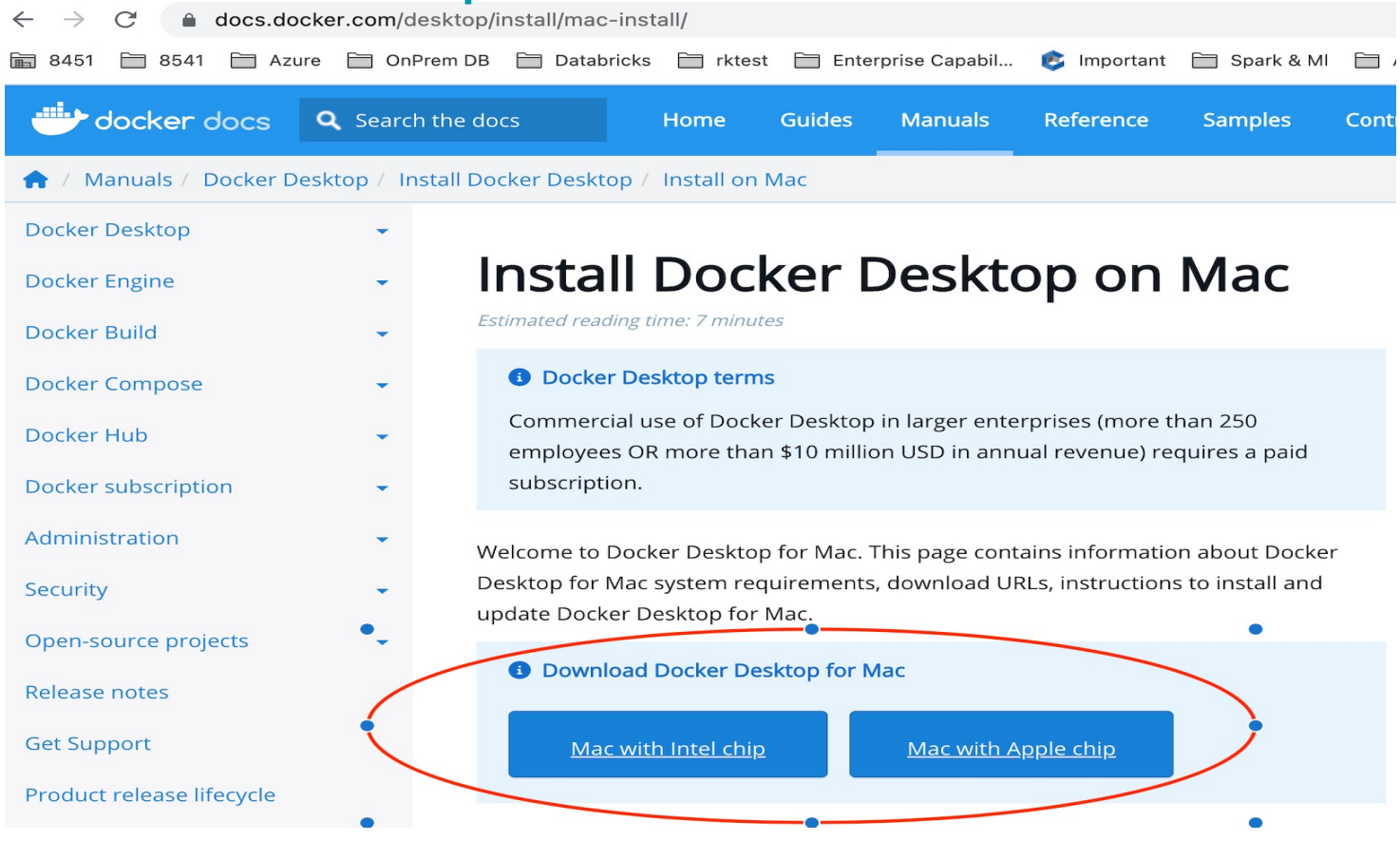
**https://docs.docker.com/desktop/**



# Installing Docker Desktop

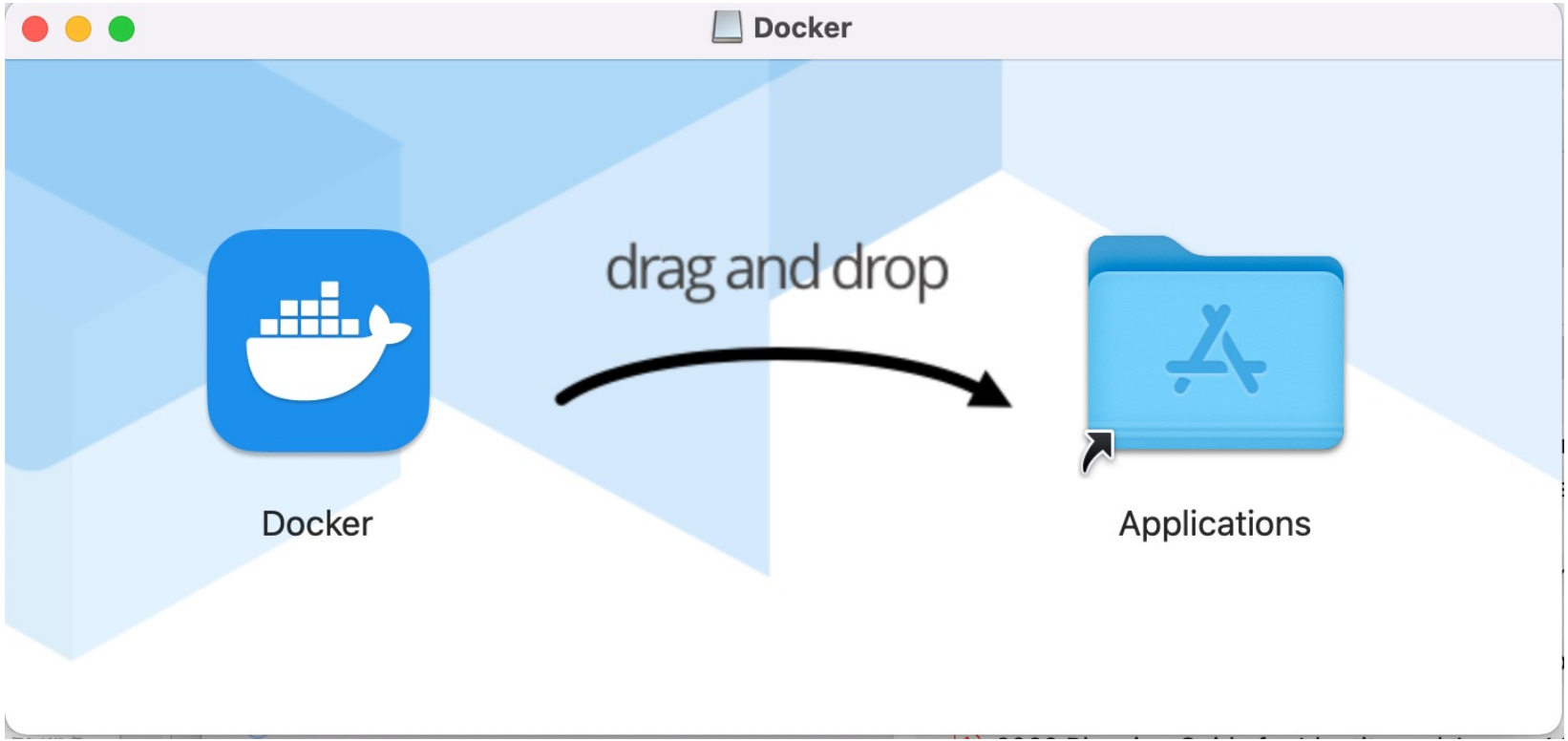
University of CINCINNATI

**https://docs.docker.com/desktop/install/mac-install/**



# Installing Docker

University of CINCINNATI




## Create a Docker ID.

Already have an account? **Sign In**

Docker ID

Email

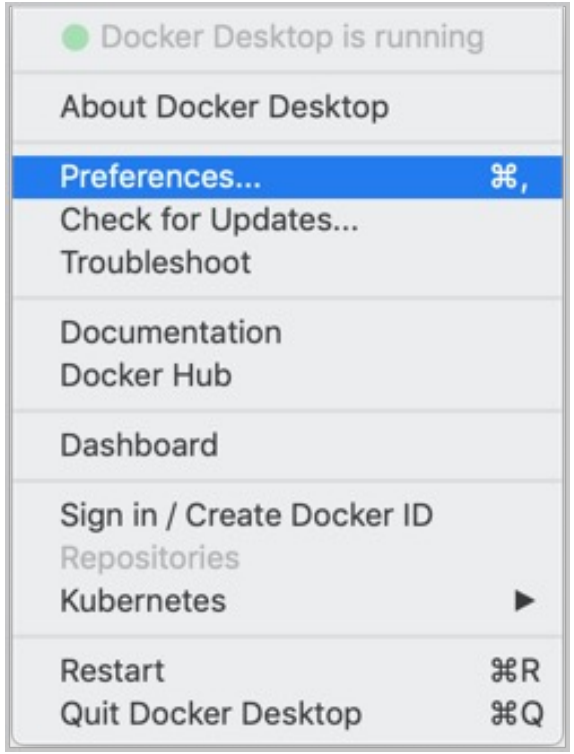
Password

Send me occasional product updates and announcements.

# Docker Desktop 4 windows/mac

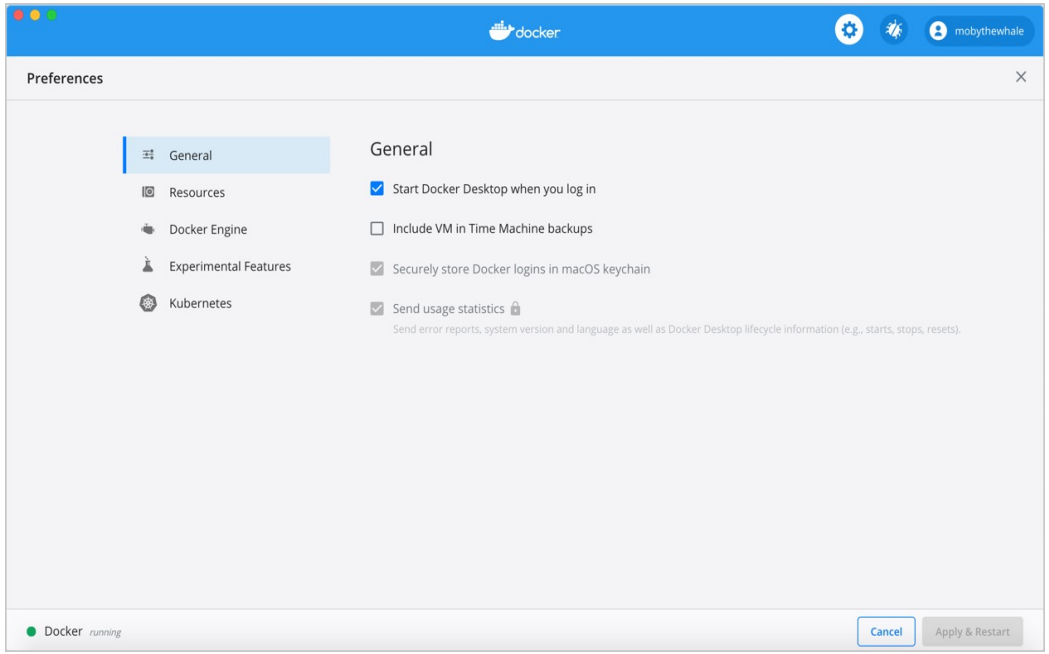
University of CINCINNATI

## Preferences
From Docker menu




## General

- On the **General** tab, you can configure:
- **Start Docker Desktop when you log in**: Automatically
- **Automatically check for updates**:

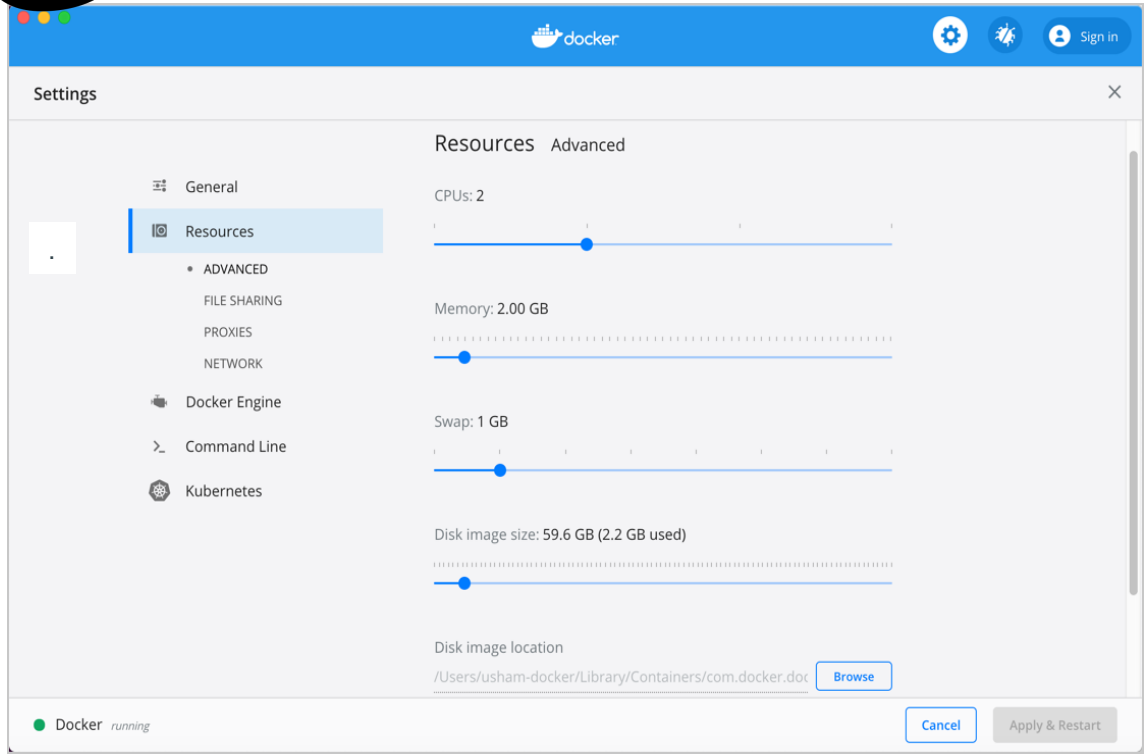


# Docker Desktop for windows/mac

University of CINCINNATI

**Limit/configure CPU, memory, disk, proxies, network, and other resources available to Docker**

## Resources




**Advanced settings:**

- can be increase / decrease:
- **CPUs**: default, 1/2 of processors on host machine.
- **Memory**: default, 2 GB memory,
- **Swap**: default, 1 GB.
- **Disk image size**:.
- **Disk image location**: Location of the Linux volume where containers and images are stored.

# Dockers by Doing

University of CINCINNATI

## LEARNING OBJECTIVES:

Introduction
- ✓ What is docker?

1. Installing Docker
   - ✓ Setting up Docker
   - ✓ Docker Desktop
   - ✓ Installing Docker on Mac, Windows, **Linux**
2. Using Docker
   - ✓ The Docker flow: Images to containers
   - ✓ The Docker flow: Containers to images
   - ✓ Run processes in containers
   - ✓ Manage containers
   - ✓ Exposing ports
   - ✓ Container networking
   - ✓ Docker registries
3. Building Docker Images
   - ✓ What is Dockerfiles?
   - ✓ Building Dockerfiles
   - ✓ Dockerfile syntax
   - ✓ Multi-project Docker files
   - ✓ Avoid golden images
4. Building entire systems with Docker
   - ✓ Docker the program
   - ✓ Storage
5. Orchestration: E2E Systems w/ Docker
   - ✓ Registries in detail
   - ✓ Into to orchestration
   - ✓ Kubernetes in Cloud
   - ✓ Next steps

# Get started with Docker !

University of CINCINNATI

## Check versions

$ docker

```
list of all Docker commands
```

$ docker --version

```
Docker version 19.03.5, build 633a0ea
```

## Explore the application

***Open a command-line terminal, test installation by running the simple Docker image, hello-world:***

$ docker run hello-world

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:ca0eeb6fb05351dfc8759c20733c91def84cb8007aa89a5bf606bc8b315b9fc7
Status: Downloaded newer image for hello-world:latest

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

***Starting a Dockerized web server. Like the hello-world image above, if the image is not found locally, Docker pulls it from Docker Hub.***

```
(base) CINMAC11673:~ r188694$ docker run hello-world

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (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/

(base) CINMAC11673:~ r188694$
```

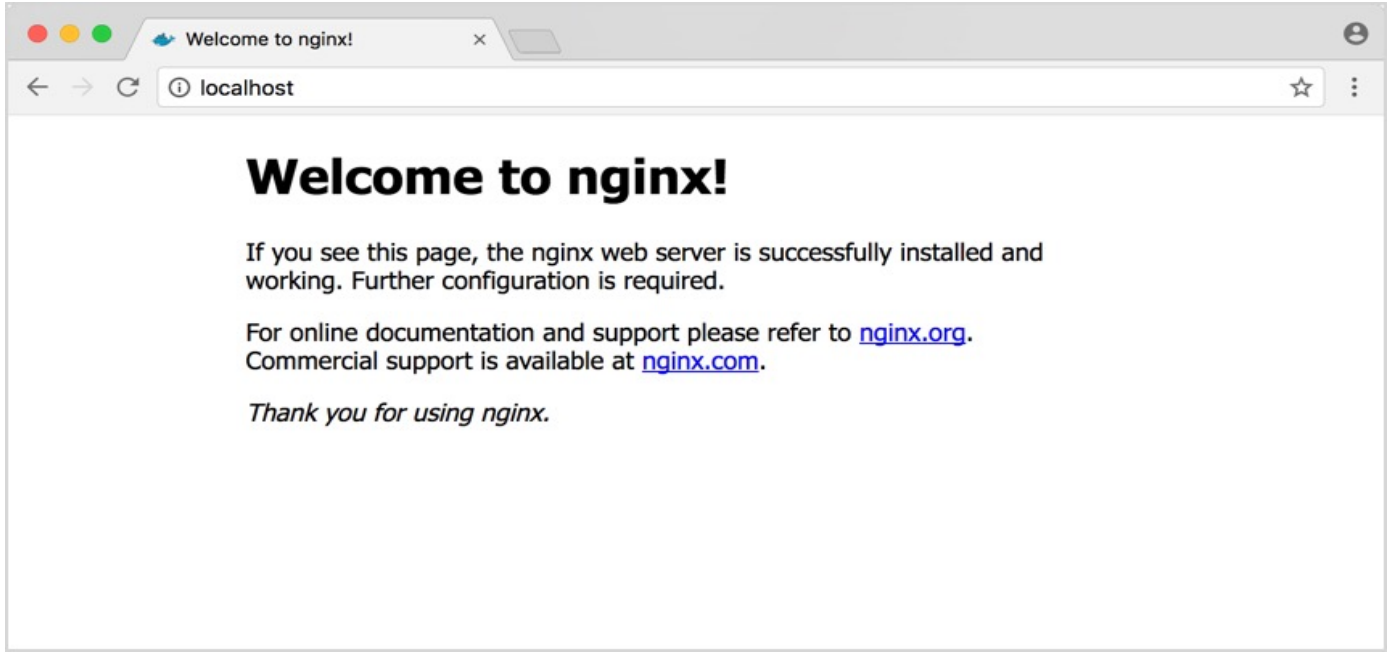
$ docker run hello-world

# Get started with Docker App !

University of CINCINNATI

**$ docker run --detach --publish=80:80 --name=webserver nginx**

*In a web browser, go to http://localhost/ to view the nginx homepage. Because we specified the default HTTP port, it isn't necessary to append :80 at the end of the URL.*



University of CINCINNATI

```
(base) CINMAC11673:~ r188694$ docker info
Client:
 Debug Mode: false

Server:
 Containers: 5
  Running: 0
  Paused: 0
  Stopped: 5
 Images: 15
 Server Version: 19.03.5
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339
 runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 4.19.76-linuxkit
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 6
 Total Memory: 1.943GiB
 Name: docker-desktop
 ID: 3XHB:X675:Z2VE:EWZY:IZNP:QIXH:5DEA:PGPQ:IRRS:IKHF:WNZG:T353
 Docker Root Dir: /var/lib/docker
 Debug Mode: true
  File Descriptors: 34
  Goroutines: 51
  System Time: 2020-02-22T19:40:12.2198064Z
  EventsListeners: 3
 HTTP Proxy: gateway.docker.internal:3128
 HTTPS Proxy: gateway.docker.internal:3129
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
 Product License: Community Engine
```

**$ docker info**

# Docker Image



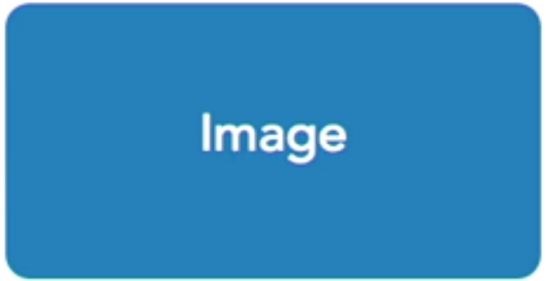



Image is instantiated to form container

# Docker Image



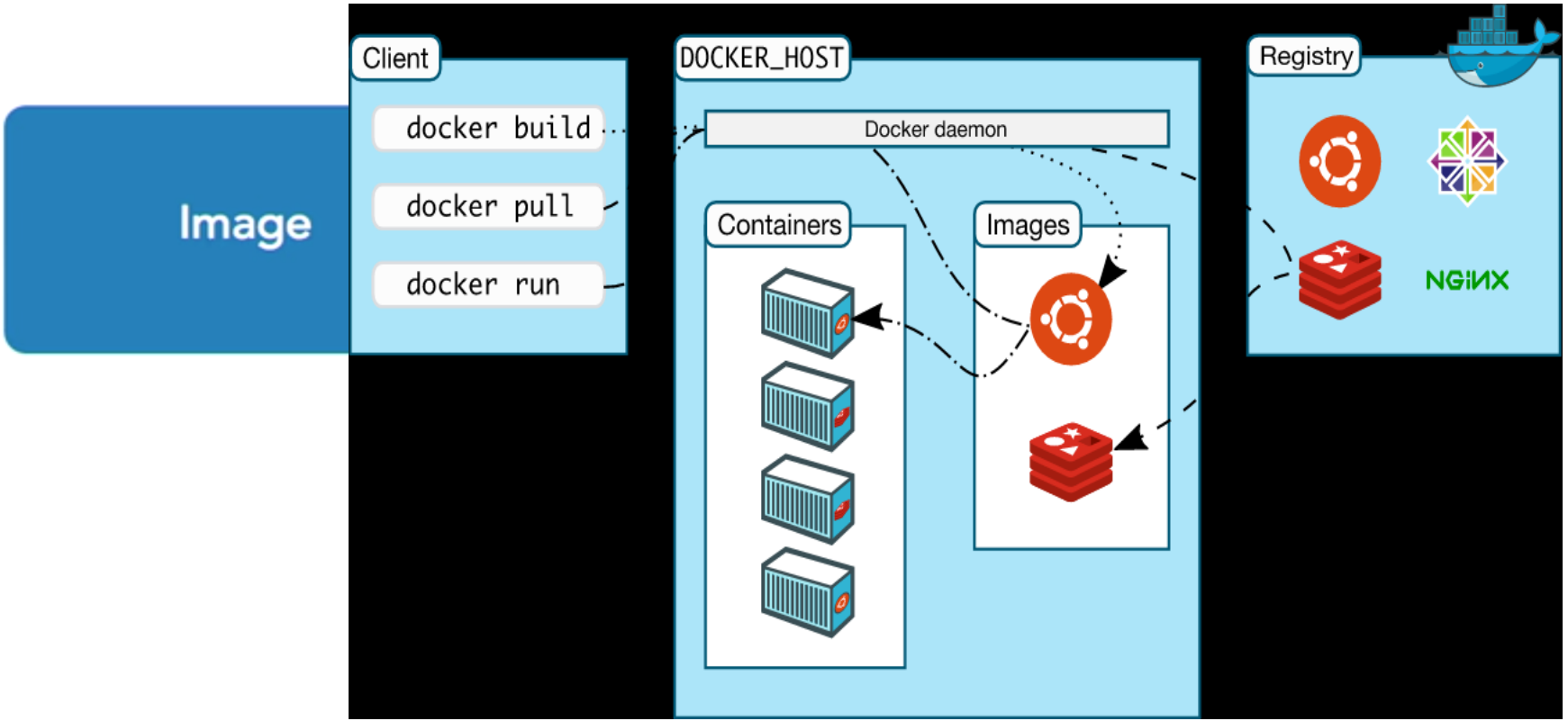
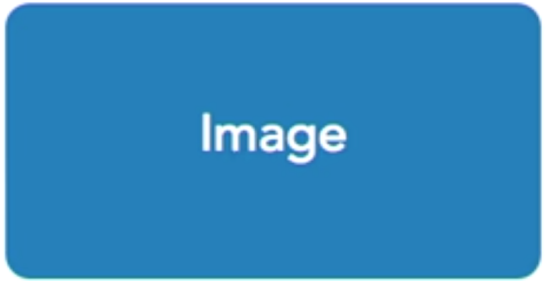



Image is instantiated to form container

# Docker Image






**Layers**

A Docker container **image** is structured in terms of “layers”.

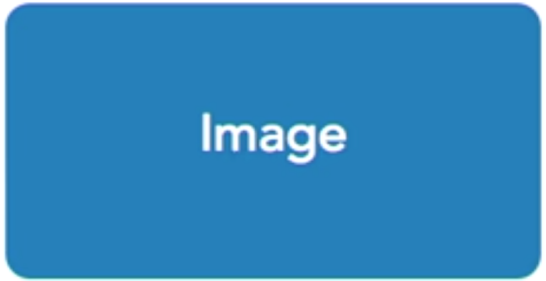
Process for building image

1. Start with base image
2. Load software desired
3. Commit base image+software to form new image
4. New image can then be base for more software

**Image is what is transferred**

# Docker Image






## Exploiting layers

- ✓ When an image is updated, only update new layers
- ✓ Unchanged layers do not need to be updated
- ✓ Consequently, less software is transferred, and an update is faster

## Trade offs

**Virtual machine** gives you all the freedom you have with bare metal

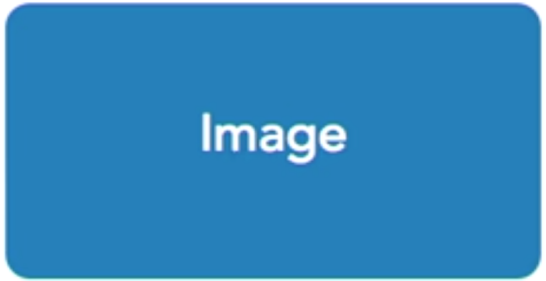
- ✓ Choice of operating system
- ✓ Total control over networking arrangement and file structures

**Container** is constrained in terms of operating systems available

- ➢ Operating systems, primarily Linux, limited Windows
- ➢ Provides limited networking options
- ➢ Provides limited file structuring options

# Docker Image






## # docker pull ubuntu

- Execute “docker pull Ubuntu”
- This loads an image from the docker library
- The image contains bare copy of ubuntu

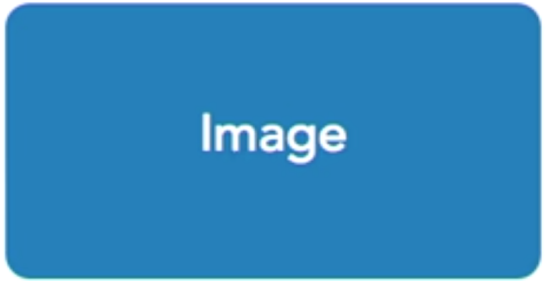
## # docker images

- Execute “docker images”
- This generates a list of images known to Docker on your machine
- You should see Hello World and ubuntu

**https://docs.docker.com/engine/reference/commandline/images/**

# Docker Image !

University of CINCINNATI




## $ docker pull ubuntu

CINMAC11673:~ r188694$ **docker pull ubuntu**

```
Using default tag: latest
latest: Pulling from library/ubuntu
423ae2b273f4: Pull complete
b6b53be908de: Pull complete
Digest: sha256:04d48df82c938587820d7b6006f5071dbbffceb7ca01d2814f81857c631d44df
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

## $ docker images

CINMAC11673:~ r188694$ **docker images**

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| ubuntu | latest | 72300a873c2c | 22 hours ago | 64.2MB |

## $ docker rmi ubuntu

CINMAC11673:~ r188694$ **docker rmi ubuntu**

```
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:04d48df82c938587820d7b6006f5071dbbffceb7ca01d2814f81857c631d44df
Deleted: sha256:72300a873c2ca11c70d0c8642177ce76ff69ae04d61a5813ef58d40ff66e3e7c
```

# Docker Flow: docker run

University of CINCINNATI

# Docker Flow: docker run. #




**# docker ps –a**

Execute “docker ps –a”
This generates a list of all of the containers that have been run

**# docker ps**

Running container

**#**

**docker run -i -t ubuntu**

Execute docker run –i –t Ubuntu
This executes an image. An executing image is called a “container”.
You are now inside the container.
Execute “ls”.

✓ A directory structure is set up but only a bare bones OS has been loaded

# Docker Flow: docker run !

University of CINCINNATI

**$ docker run -ti ubuntu:latest bash** Terminal 1

```
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
423ae2b273f4: Pull complete
b6b53be908de: Pull complete
Digest: sha256:04d48df82c938587820d7b6006f5071dbbffceb7ca01d2814f81857c631d44df
Status: Downloaded newer image for ubuntu:latest
```

**root@ba69ba2c9654:/#**

**$ ls**

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

**docker ps**

**docker ps –l**

**docker rm ba69ba2c9654 (-f to force)**

**docker ps –a.** Terminal 2

```
CONTAINER ID   IMAGE           COMMAND    CREATED       STATUS         PORTS                    NAMES
ba69ba2c9654   ubuntu:latest   "bash"     2 hours ago   Up 2 hours                              jolly_swirles
009966e3263e   hello-world     "/hello"   4 hours ago   4 hours ago                             peaceful_turing
baa74501a1ce   postgres        "docker-ent…"  7 months ago  27 months  0.0.0.0:5433->5432/tcp  postgres_test
```

# Docker Flow: docker run !

University of CINCINNATI


## Install software on container

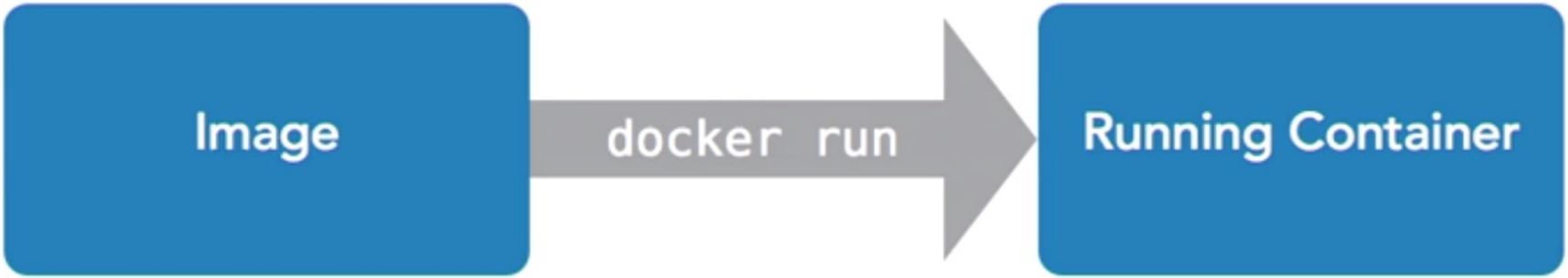
Execute
#$
apt-get update
apt-get install wget
apt-get install nodejs
apt-get install npm
<cntl d>
This installs the software you will use during this session and exits the container

## #$ cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"

# Docker flow: Images to containers

University of CINCINNATI




**$docker image**

**$docker ps -a**

# docker images <List images>

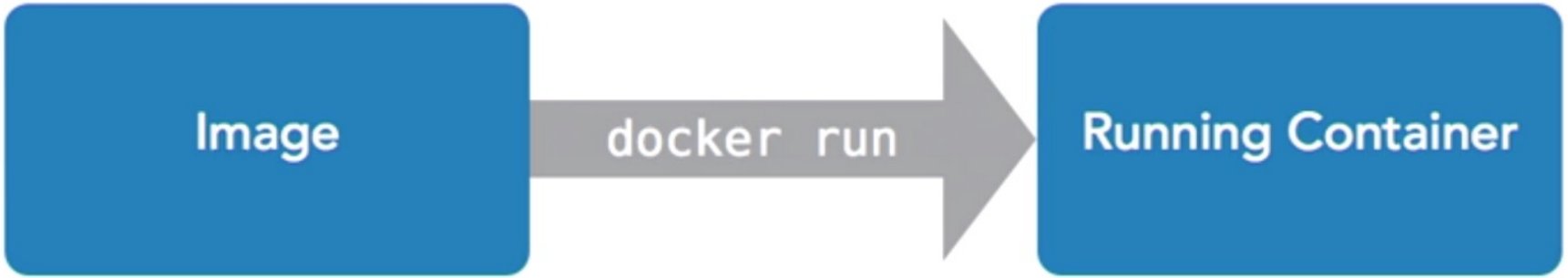
```
REPOSITORY | TAG    | IMAGE ID     | CREATED      | SIZE
ubuntu     | latest |72300a873c2c |26 hours ago |64.2MB
```

# docker ps <List containers>

```
CONTAINER ID | IMAGE     |COMMAND | CREATED       | STATUS    | PORTS | NAMES
ac294f838c45 |uc-cc-tag | "bash" |7 minutes ago |Up 7 minutes |     |vigorous_hertz
```

# Docker flow: Images to containers !

University of CINCINNATI




Terminal 1

docker run -ti ubuntu:latest bash

root@**ba69ba2c9654**:/#

**#$ touch** RASHMI_FILE_HERE

**#$ ls**

**RASHMI_FILE_HERE**

bin boot dev etc home lib lib64 media mnt
opt proc root run sbin srv sys tmp usr var

Terminal 2

docker run -ti ubuntu:latest bash

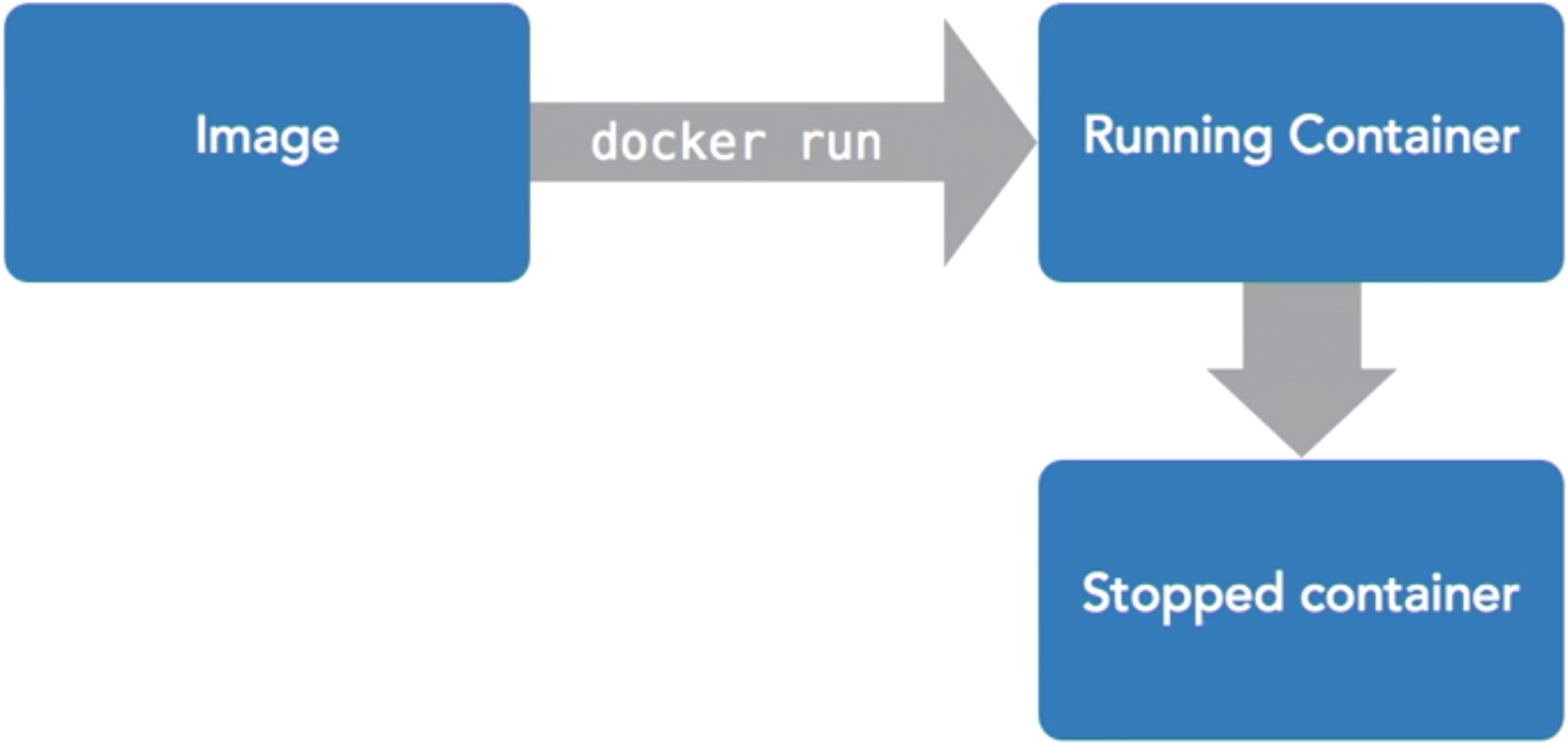
root@**2b6b974b9358**:/#

#$ ls

bin boot dev etc home lib lib64 media mnt
opt proc root run sbin srv sys tmp usr var

# The Docker flow: Images to containers

University of CINCINNATI



# The Docker flow: Images to containers !

University of CINCINNATI

Terminal 1

$ exit

```
exit
```

$ docker ps -l

```
CONTAINER ID | IMAGE         | COMMAND | CREATED    | STA      | PORTS | NAMES
2b6b974b9358 |ubuntu:latest| "bash"  |13 mins ago| Exited(0)8secs| | suspicious_mayer
```

**$ docker commit 2b6b974b9358**

```
sha256:19f33e73e32f8970d5f7commit7392f0eb5bda0b8956bf62689ebed5283d8abbe1d14b
```

$ docker images

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| <none> | <none> | 19f33e73e32f | 8 seconds ago | 64.2MB |
| ubuntu | latest | 72300a873c2c | 26 hours ago | 64.2MB |

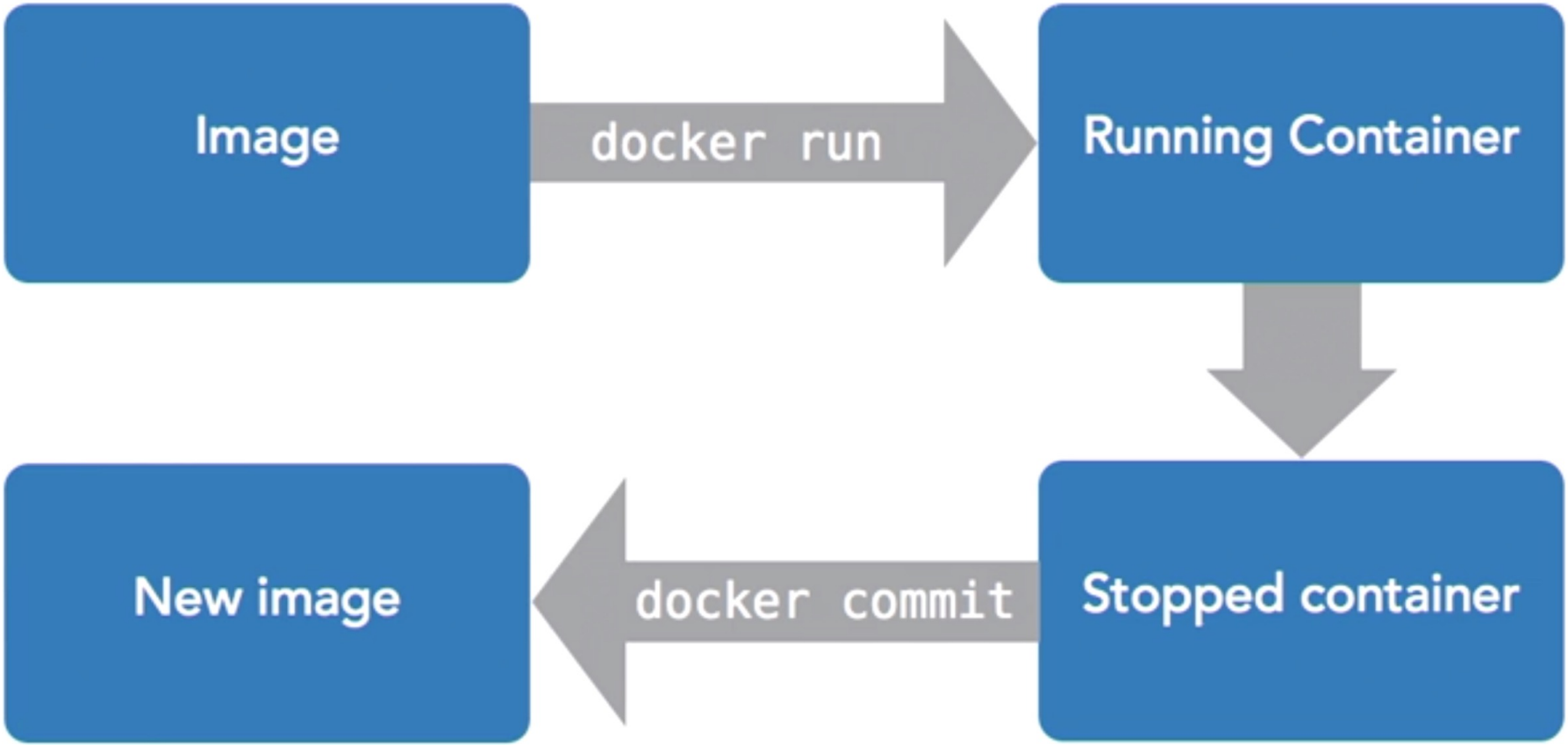
**$ docker tag 19f33e73e32f8970d5f77392f0eb5bda0b8956bf62689ebed5283d8abbe1d14b uc-cc-tag**

$ docker images

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| uc-cc-tag | latest | 19f33e73e32f | 7 minutes ago | 64.2MB |
| ubuntu | latest | 72300a873c2c | 26 hours ago | 64.2MB |

# The Docker flow: Containers to images

University of CINCINNATI



# The Docker flow: Containers to images

University of CINCINNATI

## Make an image called uc-cc-tag2

**docker commit vigorous_hertz uc-cc-tag2**

- ✓ Note that the ubuntu container has a name of "**vigorous_hertz** " (on my machine). It will be different on yours.
- ✓ "docker commit **vigorous_hertz uc-cc-tag2**" creates an image with the name **uc-cc-tag2**

# The Docker flow: Containers to images

University of CINCINNATI

Terminal 1

$ docker images

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
uc-cc-tag           latest              19f33e73e32f        7 minutes ago       64.2MB
ubuntu              latest              72300a873c2c        26 hours ago        64.2MB
```

$ docker run -ti uc-cc-tag

```
root@ac294f838c45:/#
```

$ ls

```
RASHMI_FILE_HERE  bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
```

Terminal 2

$docker ps

```
CONTAINER ID        IMAGE       COMMAND      CREATED             STATUS          PORTS       NAMES
ac294f838c45 uc-cc-tag  "bash"       7 minutes ago       Up 7 minutes                    vigorous_hertz
```

$ docker commit vigorous_hertz uc-cc-tag2

```
sha256:73b153056ee805ff86974265fcac178f0bb29e66fd94a0112c82c59b01f06097
```

$ docker images

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
uc-cc-tag2          latest              73b153056ee8        23 seconds ago      64.2MB
uc-cc-tag           latest              19f33e73e32f        25 minutes ago      64.2MB
```

# Execute app in Dockers #

University of CINCINNATI

## docker run

- ✓ **Containers have a main process**
- ✓ **The container stops when that process stops**
- ✓ **Containers have names**

## docker run -rm. Delete the container when you are done running

docker run --rm -ti ubuntu sleep 5

docker run --rm -ti ubuntu sleep 5; echo all done

## > docker run -ti uc-cc-tag3 bash

You are back inside a container. Load application:

```
wget https://raw.githubusercontent.com/cmudevops/ipshow.js/master/initialization_script
wget https://raw.githubusercontent.com/cmudevops/ipshow.js/master/ipshow.js
```

# Leaving Things Running in a Containers

University of CINCINNATI

## docker attach

- ✓ Detached containers
- ✓ Interactive containers
- ✓ Detach with Control-p Control-q
- ✓ Docker ps
- ✓ Docker attach container_name

Exit the container <Cntl d> or “exit”

# Leaving Things Running in a Containers #

University of CINCINNATI

Detached containers

# docker run -d -ti ubuntu bash

root@09c56d2b07dc:/# (base)

Control-p Control-q

# docker ps

```
CONTAINER ID | IMAGE | COMMAND | CREATED       |STATUS   | PORTS | NAMES
09c56d2b07dc | ubuntu| "bash"  |34 seconds ago|Up 33 seconds | | tender_moser
```

# docker attach 09c56d2b07dc

root@**09c56d2b07dc**:/#

# Leaving Things Running in a Containers

University of CINCINNATI

Terminal 1

```
$ docker ps
CONTAINER ID | IMAGE | COMMAND | CREATED       |STATUS   | PORTS | NAMES
09c56d2b07dc | ubuntu| "bash"  |34 seconds ago|Up 33 seconds | | tender_moser
$ docker attach tender_moser
root@09c56d2b07dc:/#
#$ touch foobar
#$ ls
bin boot dev etc foobar home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
```

Terminal 2

```
$ docker exec -ti tender_moser bash
root@09c56d2b07dc:/#
#$ ls
bin boot dev etc foobar home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
```

Terminal 3

```
$ docker ps
CONTAINER ID | IMAGE | COMMAND | CREATED       |STATUS   | PORTS | NAMES
09c56d2b07dc | ubuntu| "bash"  |34 seconds ago|Up 33 seconds | | tender_moser
```

# Running More Things in a Containers



**docker exec**

- ✓ Starts another process in an existing container
- ✓ Great for debugging and DB administration
- ✓ Can’t add ports, volumes, and so on

# Looking at Container Output

University of CINCINNATI

**docker logs**

- ✓ Keep the output of containers
- ✓ View with docker log container_name

# Looking at Container Output !

University of CINCINNATI

$ docker run --name example5 -d ubuntu bash –c “typo /etc/passwd”

```
fab83d3a7c40851b57195772664b6986cb2963b60719e6fca84a70c0afdcde79
```

$ docker logs example

bash: –c: No such file or directory

```
See 'docker --help'
```

# Looking at Container Output

University of CINCINNATI

**docker logs**

- ✓ Keep the output of containers
- ✓ View with docker log container_name
- ✓ Don’t let the output get too large

# Stopping and Removing Containers

University of CINCINNATI

## Killing and Removing containers

- ✓ docker kill container_name
- ✓ docker rm container_name

# Stopping and Removing Containers !

University of CINCINNATI

$ docker run -ti ubuntu bash

Terminal 1

```
Digest: sha256:04d48df82c938587820d7b6006f5071dbbffceb7ca01d2814f81857c631d44df
```

$ docker ps

Terminal 2

```
CONTAINER ID | IMAGE | COMMAND | CREATED       |STATUS   | PORTS | NAMES
178b876c318f | ubuntu| "bash"  |34 seconds ago|Up 33 seconds | | nervous_kalam
```

$ docker kill nervous_kalam

```
nervous_kalam
```

$ docker ps

```
CONTAINER ID | IMAGE | COMMAND | CREATED       |STATUS   | PORTS | NAMES
178b876c318f | ubuntu| "bash" |5 minutes ago|Exited(137)47 seconds ago|| nervous_kalam
```

$ docker rm nervous_kalam

```
nervous_kalam
```

$ docker ps -a

**<No enrty for nervous_kalam >**

# What is a Dockerfile?

University of CINCINNATI

- ✓ **Scripting:** Creating an image by hand is tedious and error prone
- ✓ You can create a script to do this (Dockerfile)
- ✓ Dockerfile is a small “program” to create image
- ✓ You run this program with

**$ docker build –t name-of-result .  <---- Note the . Here**

- ✓ When it finishes, the result will be in your local docker registry

# Producing Nest Image with each Step

University of CINCINNATI

- ✓ Each line takes the image from the previous line and make another image
- ✓ The previous image is unchanged
- ✓ It does not edit the state form the previous line
- ✓ You don’t want large file to span lines,  or your image will be huge

# Not Shell Scripts

University of CINCINNATI

- ✓ Dockerfiles look like shell scripts
- ✓ Dockerfiles are not shell scripts
- ✓ Processes you start on one line will not be running on the next line
- ✓ Environment variable you set will be set on the next line

If you use the ENV command, remember that each line is its own call to docker run

# How do containers get to hosts?

University of CINCINNATI

**Three options**

- ✓ **Containers can be copied at each invocation.**
  - Copying time is overhead
  - Makes hosts flexible with respect to which containers they run
- ✓ **Containers can be preloaded on hosts**
  - No copying time at invocation
  - When there are multiple different containers, allocator is constrained to allocate to hosts with appropriate containers.
- ✓ **Some layers can be preloaded on hosts**
  - Only copying time for additional layers
  - Allocator is constrained to allocate to appropriate preloaded software

# The Most Basic Dockerfile

University of CINCINNATI

- ✓ Create a new file named Dockerfile

```
FROM busybox

RUN echo “building simple docker image.”

CMD echo “Hell Container”
```

# The Most Basic Dockerfile

University of CINCINNATI

```
$ mkdir example1
$ cd example1
$ ls

$ vi Dockerfile
FROM busybox
RUN echo “building simple docker image.”
CMD echo “Hell Container”
```

# The Most Basic Dockerfile

University of CINCINNATI

## $ docker build -t example1 .

```
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM busybox
 ---> 6d5fcfe5ff17
Step 2/3 : RUN echo "building simple docker image"
 ---> Running in 1773b8c4ad5c
building simple docker image
Removing intermediate container 1773b8c4ad5c
 ---> e24161e16111
Step 3/3 : CMD echo "hello container"
 ---> Running in ef600467ccd5
Removing intermediate container ef600467ccd5
 ---> eda7426ad9bc
Successfully built eda7426ad9bc
Successfully tagged example1:latest
```

## $ docker images

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| example1 | latest | eda7426ad9bc | 8 seconds ago | 1.22MB |
| ubuntu | latest | 72300a873c2c | 31 hours ago | 64.2MB |

## $ docker run --rm example1

```
hello container
```

# Installing a Program with Docker Build #



Create a new Dockerfile

$ cd example2

$ vi Dockerfile

```
FROM ubuntu
RUN apt-get -y update
RUN apt-get -y install vim nano
CMD ["nano”, "/tmp/notes"]
```

# Installing a Program with Docker Build #

University of CINCINNATI

## $ docker build -t example2/nanoer .

```
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu
 ---> 72300a873c2c
Step 2/4 : RUN apt-get -y update
 ---> Using cache
 ---> 5a95fadcdbad
Step 3/4 : RUN apt-get -y install vim nano
 ---> Using cache
 ---> 5abe5d43924b
Step 4/4 : CMD ["nano" "/tmp/notes"]
 ---> Using cache
 ---> 071e7da83c28
Successfully built 071e7da83c28
Successfully tagged example2/nanoer:latest
```

## $ docker images

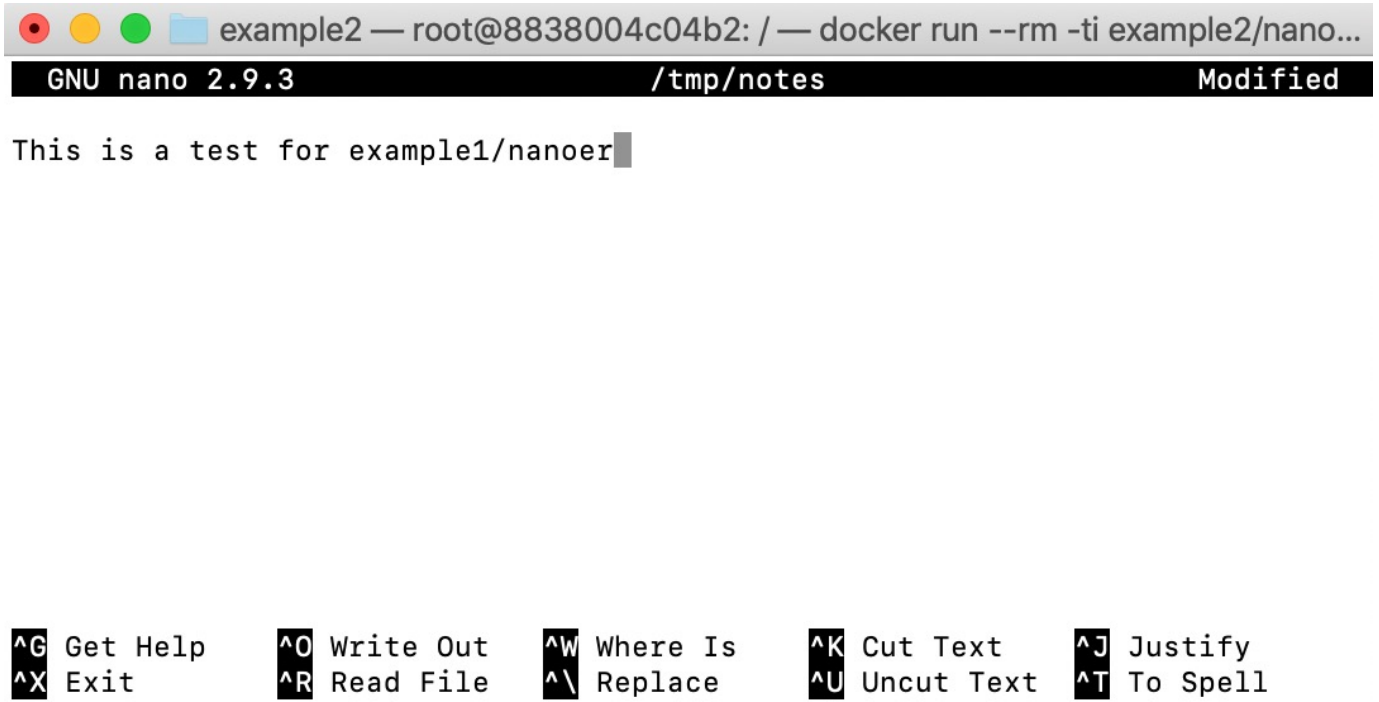
| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| example2/nanoer | latest | 071e7da83c28 | About a minute ago | 153MB |

## $ docker run --rm -ti example2/nanoer

# Installing a Program with Docker Build

University of CINCINNATI

$ docker run --rm example2/nanoer




$ docker run --rm -ti example2/nanoer bash

# Adding a File though Docker Build

University of CINCINNATI

- Add this in a Dockerfile:

```
FROM edaple2/nanoer
ADD notes.txt /notes.txt
CMD “nano” “/notes.txt”
```

# Adding a File though Docker Build #

University of CINCINNATI

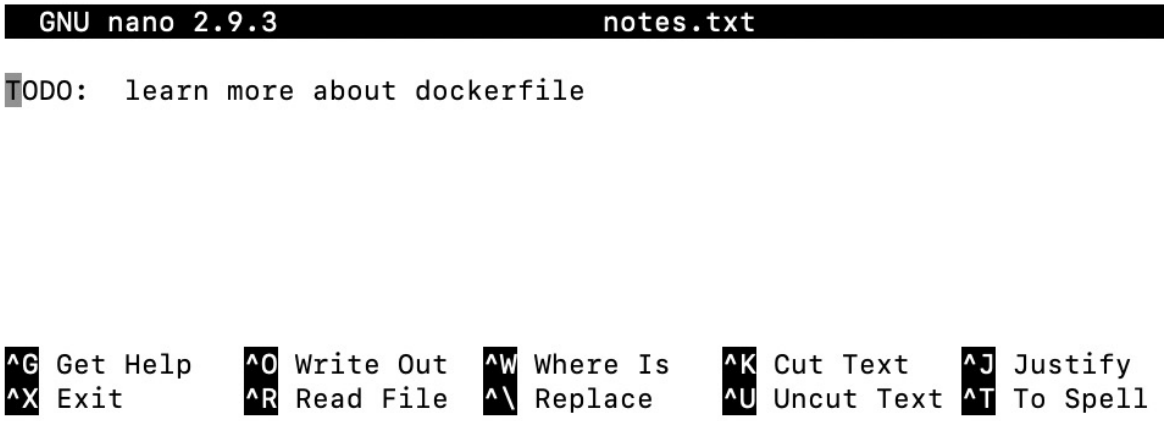
$ cat Dockerfile

```
FROM example2/nanoer
ADD notes.txt /notes.txt
CMD ["/bin/nano", "/notes.txt"]
```

$ cat notes.txt

$ docker build -t example3/todo .

$ docker run --rm –ti example3/todo



# Reflect…

University of CINCINNATI

## What have we seen

- ✓ Distinction between docker images and containers
- ✓ Creating a docker image in layers
- ✓ Provisioning the docker image from the internet
- ✓ Scripting a Dockerfile

## What is left?

- ✓ Container Networking
- ✓ Sharing of images
- ✓ Scaling of images
- ✓ Kubernetes
- ✓ Containers in Cloud

# Resource Constraints

University of CINCINNATI

**Memory limits**

- ✓ docker run --memory maximum-allowed-memory image-name command

**CPU limits**

- ✓ docker run --cpu-share relative to other containers
- ✓ docker run --cpu-quota to limit it in general

**Orchestration**

- ✓ Generally requires resource limiting

# Container Networking



Expose ports to let connection in

Private network to connect between containers

Exposing a Specific port

- ✓ Explicitly specifies the port inside the container and outside
- ✓ Exposes as many ports as you want
- ✓ Requires coordination between containers
- ✓ Make it easy to find the exposed ports

# Container Networking !

University of CINCINNATI

**$ docker run --rm -ti -p 44444:44444 -p 55555:55555 --name echo-server ubuntu:14.04 bash** Terminal 1

```
Digest: sha256:ffc76f71dd8be8c9e222d420dc96901a07b61616689a44c7b3ef6a10b7213de4
```

**#$ nc -lp 44444 | nc -lp 55555**

**$ nc localhost 44444** Terminal 2

Lets go!
UC Cloud Computing the best class ever!
Docker is Fun!

^C
$

**$ nc localhost 55555** Terminal 3

Lets go!
UC Cloud Computing the best class ever!
Docker is Fun!

$

# Container Networking

University of CINCINNATI

**$ docker run --rm -ti -p 44444:44444 -p 55555:55555 --name echo-server ubuntu:14.04 bash** Terminal 1

```
Digest: sha256:ffc76f71dd8be8c9e222d420dc96901a07b61616689a44c7b3ef6a10b7213de4
```

**#$ nc -lp 44444 | nc -lp 55555**

Terminal 2

**$ docker run --rm -ti ubuntu:14.04 bash**

root@26c3d37bbba7:/#

**#$ nc host.docker.internal 44444**

Docker is fun!

^C

$

Terminal 3

**$ docker run --rm -ti ubuntu:14.04 bash**

root@4b9519071a35:/#

**#$ nc host.docker.internal 55555**

Docker is fun!

$

You may still be able to use "host.docker.internal"

# Container Networking

University of CINCINNATI

## Expose ports Dynamically

- ✓ The port inside the container is fixed
- ✓ The port on the host is chosen from the unused ports
- ✓ This allows many containers running programs with fixed ports
- ✓ This often is used with a service discovery program

# Container Networking

University of CINCINNATI

**$ docker run --rm -ti -p 44444 -p 55555 --name echo-server ubuntu:14.04 bash** Terminal 1

```
Digest: sha256:ffc76f71dd8be8c9e222d420dc96901a07b61616689a44c7b3ef6a10b7213de4
```

**#$ nc -lp 44444 | nc -lp 55555**

Terminal 2

$ docker port echo-server

44444/tcp -> 0.0.0.0:32769

55555/tcp -> 0.0.0.0:32768

$ nc localhost 32769

Third try

exit

$

Terminal 3

$ nc localhost 32768

Third try

exit

$

# Container Networking

University of CINCINNATI

## Exposing UPU ports

- ✓ docker run –p outside-port:inside-port/protocol (tcp/udp)
- ✓ The port on the host is chosen from the unused ports
- ✓ This allows many containers running programs with fixed ports
- ✓ This often is used with a service discovery program

$ docker run –p outside-port:inside-port/protocol (tcp/udp)
$ docker run –p 1234:1234/udp

# The FROM Statement

University of CINCINNATI

- Which image to download and start from
- Must be the fist command in your Dockerfile

```
FROM java:8

FROM ubuntu:latest
```

# The MAINTAINER Statement

University of CINCINNATI

- Defines the author of this Dockerfile

```
MAINTAINER Firstname Lastname email@example.com
```

# The RUN Statement

University of CINCINNATI

- Runs the command line, waits for it to finish, and saves the result

```
RUN unzip install.zip /opt/install/
RUN echo hello docker
```

# The ADD Statement

University of CINCINNATI

- Adds local files

```
ADD run.sh /run.sh
```

- Adds the contents of tar archives

```
ADD project.tar.gz /install/
```

- Works with URLs as well

ADD https://project.example.com/download/1.0/project.rpm /project/

# The ENV Statement

University of CINCINNATI

- Sets environment variables
- Both during the build and when running the result

```
ENV DB_HOST=db.production.example.com
ENV DB_PORDT=5432
```

# The ENTRYPOINT & CMD Statement

University of CINCINNATI

- ENTRYPOINT specifies the start of the command to run
- CMD specifies the whole command to run
- If you have both ENTRYPOINT and CMD, the are combined together
- If your container acts like a command-line program, you can use ENTRYPOINT
- If you are unsure, you can use CMD

# Shell Form vs. Exec Form

University of CINCINNATI

- ENTRYPOINT Run and CMD can use either form
- Shell form looks like this:

```
nano notes.txt
```

- Exec form looks like this:

```
["/bin/nano", "notes.txt"]
```

# The EXPOSE Statement

University of CINCINNATI

- Maps a port into the container

```
EXPOSE 8080
```

# The VOLUME Statement

University of CINCINNATI

- Defines shared or ephemeral volumes

```
VOLUME [“/host/path/” “/container/path/”]

VOLUME [“/shared-data”]
```

- Avoid defining shared folders in Dockerfiles
  - Use tmpfs mount to avoid storing the data anywhere permanently
  - Volumes preferable to bind mounts

# The WORKDIR Statement

University of CINCINNATI

- Sets the directory the container starts in

```
WORKDIR /install/
```

# The USER Statement

University of CINCINNATI

- Sets which user the container will run the as

```
USER rashmi

USER 1000
```

*Estimated reading time: 74 minutes*

# Dockerfile reference

Docker can build images automatically by reading the instructions from a `Dockerfile`. A `Dockerfile` is a text document that contains all the commands a user could call on the command line to assemble an image. Using `docker build` users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a `Dockerfile`. When you are done reading this page, refer to the `Dockerfile` Best Practices for a tip-oriented guide.

## Usage

The docker build command builds an image from a `Dockerfile` and a *context*. The build's context is the set of files at a specified location `PATH` or `URL`. The `PATH` is a directory on your local filesystem. The `URL` is a Git repository location.

A context is processed recursively. So, a `PATH` includes any subdirectories and the `URL` includes the

https://docs.docker.com/engine/reference/builder

```
$ docker build .
Sending build context to Docker daemon  6.51 MB
...
```

# Sharing image

University of CINCINNATI

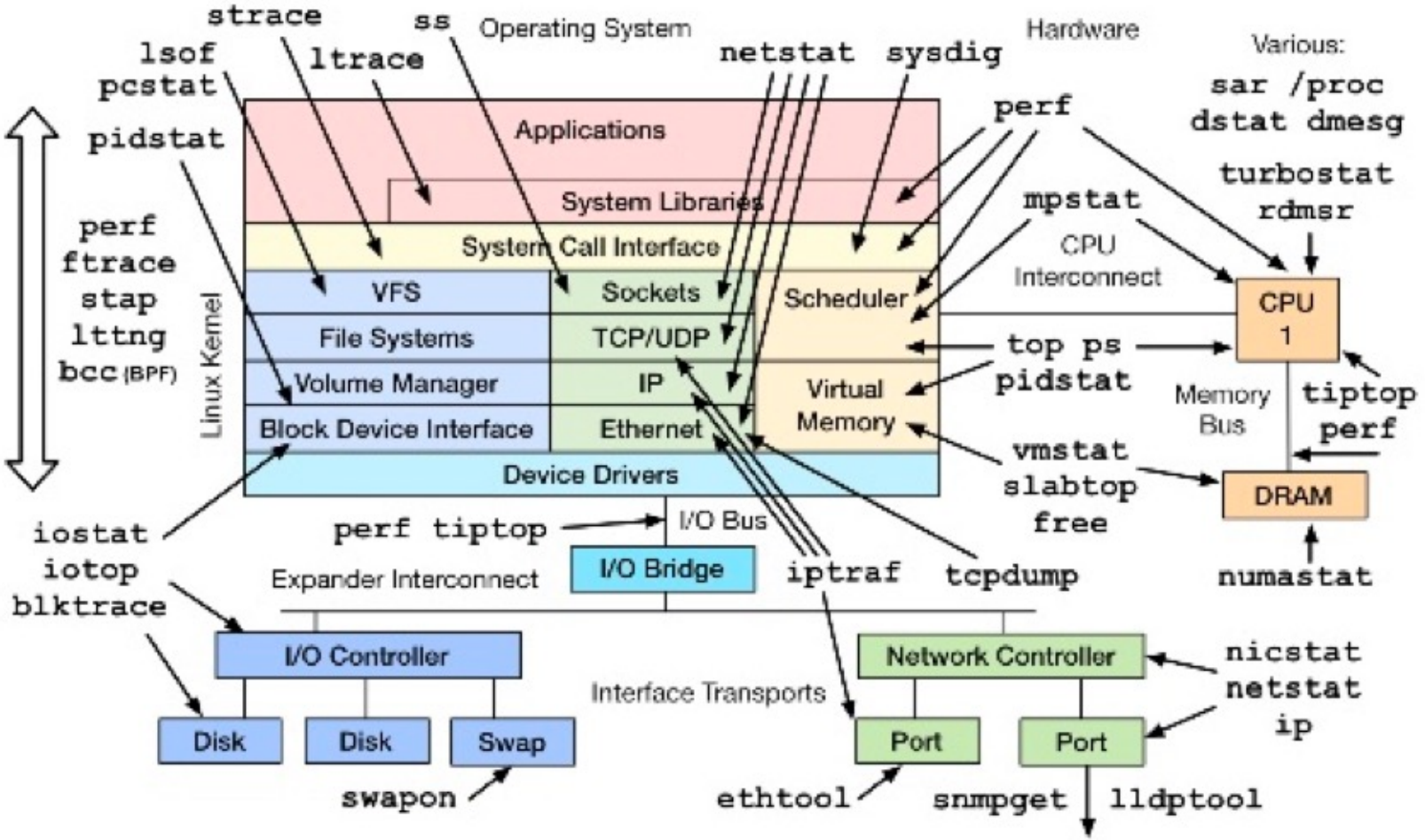
- ✓ Multiple team members may wish to share images
- ✓ Images can be in production, under development or under test
- ✓ Docker Hub is a repository where images can be stored and shared
  - Each image is tagged to allow versioning
  - Any image can be “pulled” to any host (with appropriate credentials)
  - Tagging as “latest” allows updates to be propagated. Pull <image name>:latest gets the last image checked into repository with that name

# What Kernels Do



- Respond to messages from the hardware
- Start and schedule programs
- Control and organize storage
- Pass messages between programs
- Allocates resource, memory, CPU, network, and so on
- Create containers by Docker configuring the kernel

# Linux Perf Tools




Where can we begin?

# What is Docker Compose?

University of CINCINNATI

- 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. Then, with a single command, you create and start all the services from your configuration.
- Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker-compose up and Compose starts and runs your entire app.

# a docker-compose.yml looks like this

University of CINCINNATI

```
version: '3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Backend Service

Specify Volumes/Network

Environmental variables

Frontend Service

Specify Volumes/Network

Environmental variables

docker

# Summary



- ✓ A container is a lightweight virtual machine that provides address space, network, file isolation
- ✓ Docker allows building images in layers and deployment of a new version just requires deploying layers that have changed.
- ✓ Containers can be managed either on VMs through autoscaling or on preallocated pool for short duration, quick loading
- ✓ Development workflow is supported through an image repository.