

CSC 346 - Cloud Computing

02 - SSH & Creating Docker Images

Docker Images

Docker Images

There are a few ways to make our own images

- Download from a docker image repository
 - This is what we've done so far with `docker run` commands.
- Using `docker commit` to save changes from a container to a new image.
 - Run a container, make some changes, then 'save' the changes
- Using a Dockerfile and the `docker build` command.
- Using `docker tag` to basically 'clone' an image and give it a new name.
 - This is not really creating a new image, it's just the same image with a different name

Docker Images

docker commit

- I've mentioned that images are immutable, and if you exit your container you'll lose all your changes unless you take special steps.
 - The `docker commit` command is one of those special steps.
 - First, let's make some changes.

ubuntu

Installing software with apt-get

- Run our familiar python container
 - See if the vim command exists
 - root@c2d688cb0a1b:/# vim
bash: vim: command not found
 - Use apt-get update first to update the repository sources
 - root@c2d688cb0a1b:~# apt-get update
Get:1 http://deb.debian.org/debian bullseye InRelease [91 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [91 kB]
Get:3 http://deb.debian.org/debian-bullseye-updates InRelease [91 kB]
Get:4 http://deb.debian.org/debian-bullseye/main arm64 Packages [1,022 kB]
Get:5 http://deb.debian.org/debian-bullseye/main arm64 DEP-Status [116 kB]
Get:6 http://deb.debian.org/debian-bullseye-updates/main arm64 DEP-Status [116 kB]
Fetched 8458 kB in 1s (8138 kB/s)
Reading package lists... Done
root@c2d688cb0a1b:/#

ubuntu

Installing software with apt-get

ubuntu
Installing software with apt-get

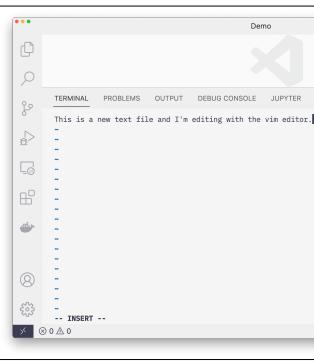
- After the installer finishes you can now use the vim command to create and edit text files.

```
root@c2d688cb0a1b:/# vim foo.txt
```



ubuntu
vim basics

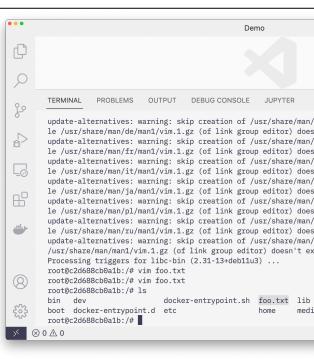
i	Enter insert mode
esc	Exit insert mode
arrow keys	Move the cursor around
:w	Save your changes (when not in insert mode)
:wq	Save your changes and exit vim (when not in insert mode)
:q!	Force quite vim and discard all changes (when not in insert mode)



ubuntu
Installing software with apt-get

- After saving changes and exiting vim the new file created is in our directory

```
update-alternatives: warning: skip creation of /usr/share/man/de  
le /usr/share/man/de/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
update-alternatives: warning: skip creation of /usr/share/man/fi  
le /usr/share/man/fi/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
update-alternatives: warning: skip creation of /usr/share/man/i  
t /usr/share/man/i/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
update-alternatives: warning: skip creation of /usr/share/man/ja  
panese /usr/share/man/japan/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
update-alternatives: warning: skip creation of /usr/share/man/p  
l /usr/share/man/pl/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
update-alternatives: warning: skip creation of /usr/share/man/r  
ussian /usr/share/man/rus/man1/vim.1.gz (of link group editor) doesn'  
t exist.  
Processing triggers for libc-bin (2.31-13+deb10u1) ...  
root@c2d688cb0a1b:/# vim foo.txt  
root@c2d688cb0a1b:/# ls  
bin dev docker-entrypoint.sh foo.txt lib  
boot docker-entrypoint.d etc home media  
root@c2d688cb0a1b:/#
```



ubuntu

Installing software with apt-get

- less is also not installed in this container, let's install that too

```
root@c2d688cb0a1b:/# less foo.txt
bash: less: command not found

root@c2d688cb0a1b:/# apt-get install less
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  less
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
9 kB of archives will be downloaded.
0kB of additional disk space will be used.
Selecting previously unselected package less.
Preparing to unpack .../archives/less_551-2_arm64.deb ...
Unpacking less (551-2) ...
Setting up less (551-2) ...

root@c2d688cb0a1b:/#
```

Docker

Multiple Container Connections

- When you use docker run -it you're creating a new container and making a shell connection to your container
- You can make more than one.
- You can use docker exec to run a command *inside of an existing container that is running*.
- Must be a running container

```
-->/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
91b549cdd0b python:3.10 "bash" 2 minutes ago Up 2 minutes
-->/Demo $ docker exec -it python bash
root@91b549cdd0b:/# ps aux
USER PID %CPU %MEM VSZ RSS STAT START TIME COMMAND
root 1 0.0 0.0 5852 3536 pts/0 S+ 04:56 0:00 bash
root 426 0.0 0.0 5852 3388 pts/1 S+ 04:59 0:00 bash
root 426 0.0 0.0 8340 2836 pts/1 R+ 04:59 0:00 bash
root@91b549cdd0b:/#
```

Docker

Multiple Container Connections

```
-->/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
91b549cdd0b python:3.10 "bash" 2 minutes ago Up 2 minutes
-->/Demo $ docker exec -it python bash
root@91b549cdd0b:/# ps aux
USER PID %CPU %MEM VSZ RSS STAT START TIME COMMAND
root 1 0.0 0.0 5852 3536 pts/0 S+ 04:56 0:00 bash
root 426 0.0 0.0 5852 3388 pts/1 S+ 04:59 0:00 bash
root 426 0.0 0.0 8340 2836 pts/1 R+ 04:59 0:00 bash
root@91b549cdd0b:/#
```

Docker

Multiple Container Connections

```
$ docker exec -it [python] bash
```

- You need to specify the name or ID of the running container
- You need to specify the command you want to execute in the new container
- In most cases, you want a new bash shell

The screenshot shows a Docker terminal window titled 'Demo'. It displays two separate terminal sessions connected to the same container. The first session is a standard bash shell, and the second session is a python shell. Both sessions are running commands like 'ps -a' and 'top' to show process details.

Docker

Multiple Container Connections

```
$ docker ps -a
```

- You can exit from this second connection and it won't kill the container
- There's still the first bash process running

This screenshot shows the same Docker terminal window as before, but now the first terminal session has been closed (indicated by a red X). Only the second terminal session, which is a python shell, is still active and running commands.

Docker

Saving with docker commit

```
$ docker ps -a
```

- From the second terminal with the container still running we can use the `docker commit` command so save the current container to a new image.
- Container can be running or stopped.
- All '`docker ...`' are run from outside of the container.

```
$ docker commit [container name] my_python:3.10
```

This screenshot shows the Docker terminal with the command '\$ docker commit [container name] my_python:3.10' entered at the prompt. The terminal also displays the output of the 'ps -a' command, showing the original container and the new image 'my_python:3.10'.

Docker

Saving with docker commit

- Now you can use the docker images command to see our newly created image

```
-/Demo $ docker images
REPOSITORY          IMAGE ID        CREATED       SIZE
my_python           3.10          2 days ago   929MB
-/Demo $
```

Docker

Saving with docker commit

- With our image "saved" we can now finally exit our other bash session in the other terminal, and exit the container
- Remember we ran the container with the --rm option, so it will be removed upon exit

```
Setting up libgbm1-amd64 (1.20.7-0) ...
Setting up less (551-2+deb10u1) ...
Setting up xxd (2:8.2.2434-3+deb10u1) ...
Setting up vim-runtime (2:8.2.2434-3+deb10u1) ...
Setting up vim (2:8.2.2434-3+deb10u1)
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/Processing triggers for liblc-bin (2.31.13+deb10u3) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
root@9ffbfld4d576:~# exit
root@9ffbfld4d576:~# exit
exit
-/Demo $ docker ps -a
CONTAINER ID        IMAGE         COMMAND      CREATED      STATUS      PORTS
-/Demo $
```

Docker

Saving with docker commit

- We can now run a new container based off of our new image
- Our foo.txt file is still there.

```
$ docker run -it my_python:3.10 bash
root@9ffbfld4d576:~# ls
root@9ffbfld4d576:~# cat foo.txt
This is a new text file and I'm editing with the vim editor.
root@9ffbfld4d576:~#
```

Docker

Stopping and Starting a container

- You don't have to throw away your container when you exit
- Without the `--rm` option, when you exit the container, it remains in an exited state
- You can re-start this container
- This is fine for prototyping, but don't depend on that stopped container. It's easy to accidentally remove it.

The screenshot shows the VS Code interface with the Docker extension installed. The terminal tab is active, displaying a session where a new file is created and read:

```
~/Demo $ docker run -it --name python my_python:3.10 bash
root@b7d0411438d3:/# echo "I'm a new file!" > new.txt
root@b7d0411438d3:/# cat new.txt
I'm a new file!
root@b7d0411438d3:/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d0411438d3 my_python:3.10 "bash" 20 seconds ago Exited (0) 3 seconds ago
python
~/Demo $ docker start -i python
root@b7d0411438d3:/# cat new.txt
I'm a new file!
root@b7d0411438d3:/#
```

The containers sidebar shows one running container named "python".

This screenshot is identical to the one above, showing the same terminal session and containers sidebar. The difference is that the entire terminal output is highlighted in green, likely indicating it is selected or being viewed.

```
~/Demo $ docker run -it --name python my_python:3.10 bash
root@b7d041f438d3:/# echo "I'm a new file!" > new.txt
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d041f438d3 my_python:3.10 "bash" 20 seconds ago Exited (0) 3 seconds ago python
```

```
~/Demo $ docker run -it --name python my_python:3.10 bash
root@b7d041f438d3:/# echo "I'm a new file!" > new.txt
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d041f438d3 my_python:3.10 "bash" 20 seconds ago Exited (0) 3 seconds ago python
```

```
~/Demo $ docker run -it --name python my_python:3.10 bash
root@b7d041f438d3:/# echo "I'm a new file!" > new.txt
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d041f438d3 my_python:3.10 "bash" 20 seconds ago Exited (0) 3 seconds ago python
```

```
~/Demo $ docker run -it --name my_python:3.10 bash
root@b7d041f438d3:/# echo "I'm a new file!" > new.txt
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d041f438d3 "python:3.10" "bash" 20 seconds ago Exited (0) 3 seconds ago python
~/Demo $ docker start -i python
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/#
```

Docker

Using docker commit on a stopped container

- You can also use **docker commit** on a stopped container that hasn't been removed yet
- You can either give this commit a new image name and tag, or you can overwrite an existing one

```
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b7d041f438d3 "python:3.10" "bash" 20 seconds ago Exited (0) 3 seconds ago python
~/Demo $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
my_http latest 234179e0c159 About an hour ago 137MB
my_python 3.10 03223a1680eb 4 days ago 862MB
python 3.10 58008402c5ab 4 days ago 862MB
httpd 2.4-alpine 65543ef2f297 4 days ago 137MB
http 2.4-alpine 74d497929003 2 weeks ago 54.1MB
~/Demo $ docker commit -t my_python:3.9 bash
root@b7d041f438d3:/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
1ba1f5fc0d44 my_python:3.10 "bash" 6 seconds ago Exited (0) 1 second ago
~/Demo $ docker commit python my_python:3.10
sha256:151f9811f9811f946a10cc40932abc245d4532f5329a0bc57ec0821beafde04
~/Demo $
```

Docker

Moving files into and out of a container

- You can copy files into and out of a running or stopped container.
- Only works with **containers**, not **images**.
- Let's say we want to work with the apache web server image for **httpd**.
- If we want to modify the default config file from the image, it would be helpful to copy the default one out of the container and then change it.

Docker

Copying Files

- Run a new container using the `httpd:2.4` image
- Look at the default directory we start in
- Change to the `conf` directory
- Look for the `httpd.conf` file

The screenshot shows a VS Code interface with a terminal tab open. The terminal output shows:

```
-/Demo $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
python    3.9   580e0402c5a8  4 days ago  869MB
httpd     2.4   b5543ef2f25e7  4 days ago  137MB

-/Demo $ docker run -it --name httpd httpd:2.4 bash
root@a24222d03cba:/usr/local/apache2# ls
bin  cgi-bin  conf  icons  include  logs
root@a24222d03cba:/usr/local/apache2# cd conf
root@a24222d03cba:/usr/local/apache2/conf# ls
extra  httpd.conf  magic  mime.types  original
root@a24222d03cba:/usr/local/apache2/conf#
```

Docker

Copying Files

- Open a new Terminal
- use the `docker cp` command to copy from inside the container to the current directory
- The special `..` directory means “the directory I’m in”

The screenshot shows a VS Code interface with a terminal tab open. The terminal output shows:

```
-/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a24222d03cba "httpd:2.4 " "bash" 3 minutes ago Up 3 minutes 80/tcp http

-/Demo $ docker cp httpd:/usr/local/apache2/conf/httpd.conf .
root@a24222d03cba:/usr/local/apache2/conf# ls
httpd.conf
root@a24222d03cba:/usr/local/apache2/conf# cat httpd.conf
#
# This is the main Apache HTTP server configuration file. It contains the
# configuration directives that give the server its instructions.
# See http://httpd.apache.org/docs/2.4/ for detailed information.
# See http://httpd.apache.org/docs/2.4/mod/directives.html
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do. They're here only as hints or reminders. If you are unsure
# consult the online docs. You have been warned.

-/Demo $
```

At the bottom of the terminal window, a black bar displays the command:

```
docker cp [container ID]:[container path] [host path]
```

Docker

Copying Files

- This works the other way too. You can copy files from your host into a running or stopped container. Just reverse the order of the arguments

```
docker cp [host path] [container ID]:[container path]
```

Docker

Other Container Commands

- You don't have to just run a new bash shell inside of a container.
 - We can just run the `ls` command
 - Or just a `cat` command

The screenshot shows a terminal window with the following command history:

```
-l@Domo $ docker run -it --name httpd httpd:2.4 ls -1
total 48
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 bin
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 build
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 cgi-bin
drwxr-xr-x  4 root  root  4096 Aug 23 04:23 conf
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 etc
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 htdocs
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 icons
drwxr-xr-x  3 root  root  4096 Aug 23 04:23 include
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 logs
drwxr-xr-x  2 root  root  4096 Aug 23 04:23 modules
-l@Domo $
```

Docker Run Errors

- Hmm what happened to cause our error?
 - We tried to run a new container with a name of `httpd`, but we did not remove the first one
 - You can't have two containers with the same name on a host at the same time

```
git:(master) ✘ 4 root 4096 Aug 23 02:23 conf
drwxr-xr-x 3 root root 4096 Aug 23 02:23 icons
drwxr-xr-x 3 root root 4096 Aug 23 02:23 include
drwxr-xr-x 2 root root 4096 Aug 23 02:23 modules
drwxr-xr-x 2 root root 4096 Aug 23 02:23 modules
drwxr-xr-x 3 root root 4096 Aug 23 02:23 static
drwxr-xr-x 3 root root 4096 Aug 23 02:23 static

dockershim: Error response from daemon: Conflict: container "9eef773c18ac989f" already in use by container "9eef773c18ac989f". You have to remove (or rename) that container to be able to reuse that name.
  - Remove $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
9eef773c18ac http://2.4 "ls -l" About a minute ago Exited (0) About a minute ago httpd
```

Docker

Other Container Commands

- After we remove the old image, you can run the command successfully.
 - By including the `--rm` option we can make sure these ephemeral commands don't leave old exited containers around

Docker

Other Container Commands

- On macOS, Linux, and Windows with `ws12` setup, you can use redirection on the host to capture the output of your docker commands

```
docker run -it --name httpd --rm httpd:2.4 cat conf/httpd.conf > ./httpd.conf
```

- Gets us the same result as `docker cp` in a different way

Docker

Volume Mounting

- Copying files back and forth from a container is tedious
- Having to commit your changes to an image each time you're done is error prone
- We can avoid both of these problems by mounting a directory from your host computer inside the running container
- This is done with the `-v` or `--volume` option to the `docker run` command

```
--volume [host path]:[container path]
```

Docker

Volume Mounting

- The host path must be a full absolute path
- Many times you want to mount your current directory, or something in it
- Can use the `$PWD` environment variable on macOS, Linux, and WSL2
- Can use the `%cd%` environment variable in PowerShell
- The following two commands are equivalent

```
docker run --volume $PWD:/root python:3.10
```

```
docker run --volume /Users/mark/Demo:/root python:3.10
```

Docker Volume Mounting

- Inside the /root directory in our container you can see the same files from our host.
- This is a live two way mapping. Changes are available in both places.

```
run.sh - Demo
$ run.sh
1 #!/bin/bash -e
2 docker run \
3   -it \
4   --rm \
5   -v name python \
6   -v volumes=/root \
7   -v python:3.10 \
8   bash
9

TUTORIAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER
> bash
>/Demo $ ls -l
total 0
drwxr-xr-x  1 root  staff  21370 Aug 27 19:58 httpd.conf
drwxr-xr-x  1 root  staff  222 Aug 27 11:45 .bash_history
o /Demo $ ./run.sh
+ docker run -it --rm -v name python -volumes=/root python:3.10 bash
root@78993d35d3:/# cd /root/
root@78993d35d3:/# ls -l
total 0
drwxr-xr-x  1 root root 21370 Aug 27 17:58 httpd.conf
drwxr-xr-x  1 root root 222 Aug 27 18:45 .bash_history
root@78993d35d3:#
```

Docker Volume Mounting

- This is really useful
- Lets us get files into a container without having to copy them each time
- Changes made inside the container to those files are reflected on the host
 - Note they're not copied, its the same file in both places. Filesystem magic!
- Changes made outside the container to the files are reflected inside the container
- Let's us work on the files in our GUI, but run them inside the container

```
hello.py - Demo
$ run.sh
$ hello.py
work > # hello.py >.
1 from datetime import datetime, time
2
3 current_time = datetime.now()
4
5 format = "%A, %B %d at %I:%M %p"
6
7 print("Hello! It is currently " + current_time.strftime(format) + ".")
8

TUTORIAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER
> bash
>/Demo $ ./run.sh
+ docker run -it --rm --name python -volumes=/Users/nash/Demo/work:/root/work python:3.10 bash
root@78993d35d3:/# cd /root/
root@78993d35d3:/# ls -l
total 0
drwxr-xr-x  3 root root 21370 Aug 27 18:54 work
root@78993d35d3:/# cd work/
root@78993d35d3:~/work# ls -l
total 0
drwxr-xr-x  1 root root 21370 Aug 27 19:03 hello.py
root@78993d35d3:~/work# python hello.py
Hello! It is currently Saturday, August 27 at 07:04 PM.
root@78993d35d3:~/work#
```

Demo

Creating Images

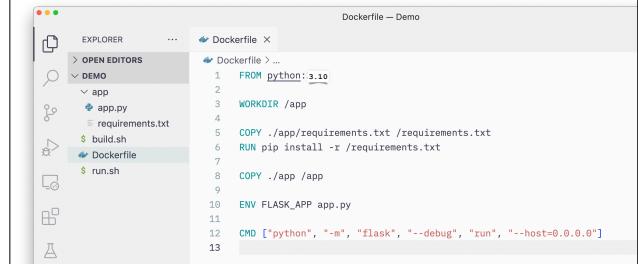
The Dockerfile

- Probably the most common ways we create Docker images for our projects are with a `Dockerfile` and the `docker build` command.

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

Creating Images

The Dockerfile



A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with a folder named 'DEMO' containing 'app', 'build.sh', 'Dockerfile', and 'run.sh'. The 'Dockerfile' tab is selected. The main editor area shows the following Dockerfile code:

```
FROM python:3.10
WORKDIR /app
COPY ./app/requirements.txt /requirements.txt
RUN pip install -r /requirements.txt
COPY ./app /app
ENV FLASK_APP app.py
CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
```

Creating Images

The Dockerfile

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

Creating Images

The Dockerfile

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

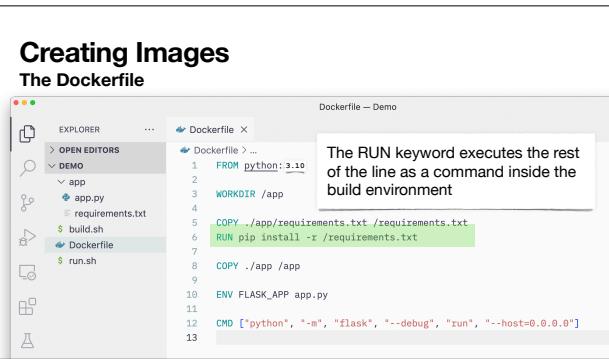
Creating Images

The Dockerfile

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

Creating Images

The Dockerfile

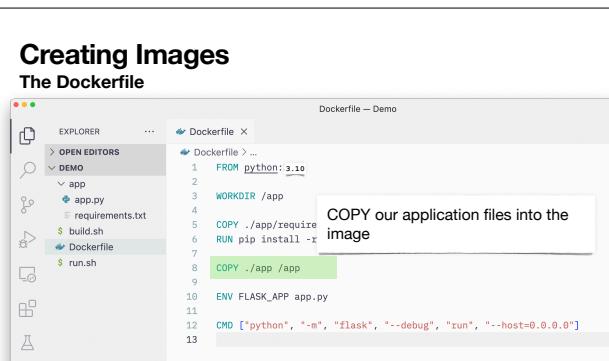


```
Dockerfile - Demo
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

The RUN keyword executes the rest of the line as a command inside the build environment

Creating Images

The Dockerfile

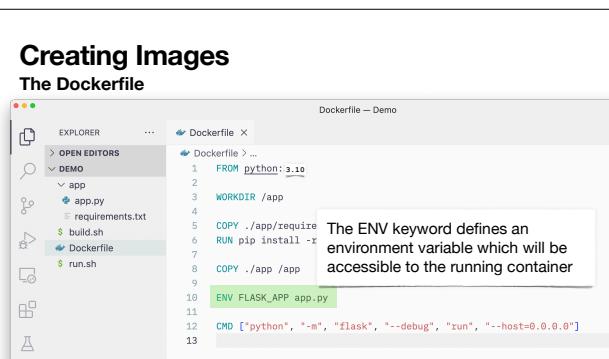


```
Dockerfile - Demo
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

COPY our application files into the image

Creating Images

The Dockerfile



```
Dockerfile - Demo
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /requirements.txt
6 RUN pip install -r /requirements.txt
7
8 COPY ./app /app
9
10 ENV FLASK_APP app.py
11
12 CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
13
```

The ENV keyword defines an environment variable which will be accessible to the running container

Creating Images

The Dockerfile

A screenshot of the Visual Studio Code interface. The left sidebar shows an 'EXPLORER' view with a tree structure: 'OPEN EDITORS' (empty), 'DEMO' (selected), 'app' (containing 'app.py' and 'requirements.txt'), 'build.sh', 'Dockerfile' (selected), and 'run.sh'. The main editor area is titled 'Dockerfile - Demo' and contains the following Dockerfile code:

```
FROM python:3.10
WORKDIR /app
COPY ./app/requirements.txt .
RUN pip install .
COPY ./app /app
ENV FLASK_APP app.py
CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]
```

A tooltip is displayed over the 'CMD' command: 'The CMD keyword defines the default command to be executed when this image is run as a container'.

Creating Images

The Dockerfile

- The `docker build` command is what turns our Dockerfile into an image

```
docker build --tag [image name]:[tag] [location]
docker build --tag my_app:latest .
```

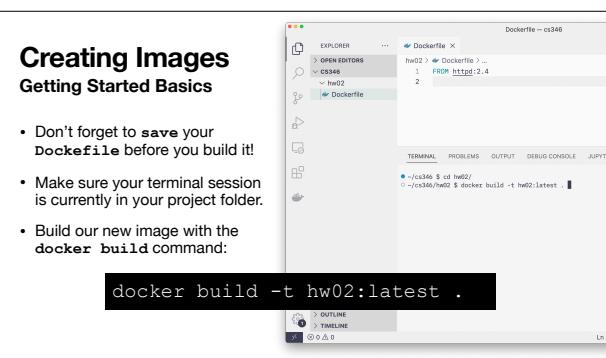
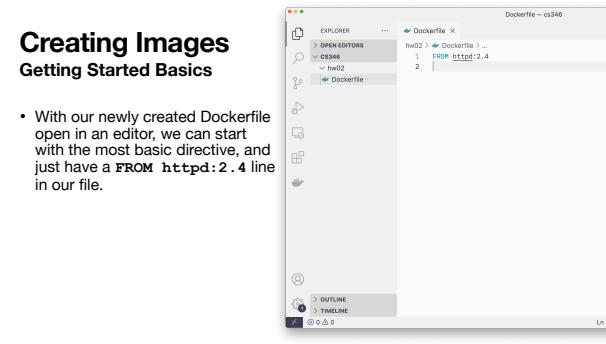
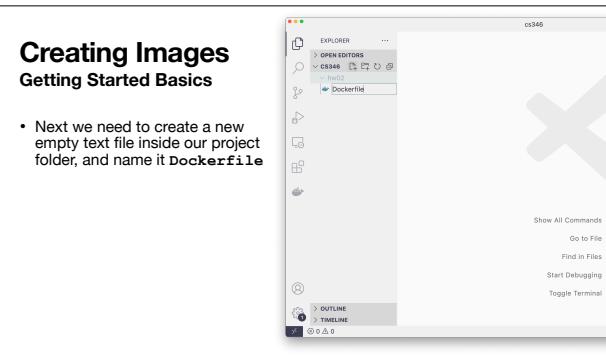
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Creating Images

Getting Started Basics

- When working on any project, such as an application or homework assignment, the first step is often to create a new directory to hold all the stuff relating to the project.
- So to start with, figure out where on your laptop you want to keep all your work for this class, and make a new folder in there. I'm going to call mine `hw02`.

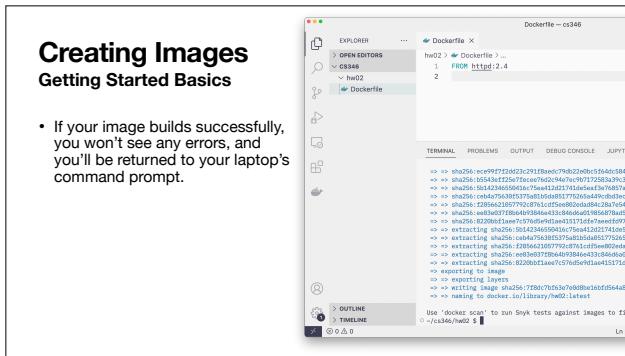
A screenshot of the Visual Studio Code interface. The left sidebar shows an 'EXPLORER' view with a tree structure: 'OPEN EDITORS' (empty), 'cs346' (selected), and 'hw02'. The main editor area has a large red 'X' icon. A context menu is open at the bottom right with options: 'Show All Commands', 'Go to File', 'Find in Files', 'Start Debugging', and 'Toggle Terminal'.



Creating Images

Getting Started Basics

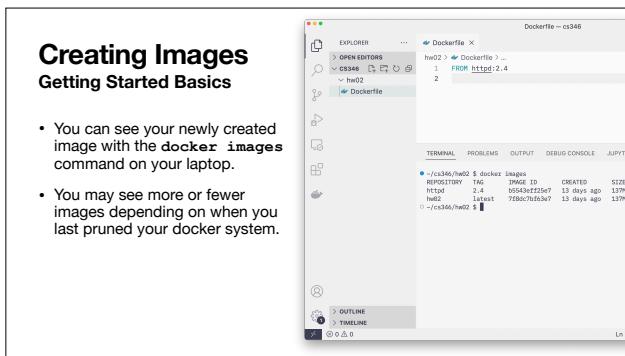
- If your image builds successfully, you won't see any errors, and you'll be returned to your laptop's command prompt.



Creating Images

Getting Started Basics

- You can see your newly created image with the `docker images` command on your laptop.
 - You may see more or fewer images depending on when you last pruned your docker system.

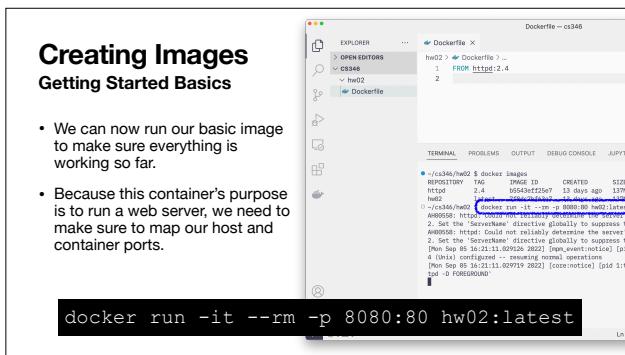


Creating Images

Getting Started Basics

- We can now run our basic image to make sure everything is working so far.
 - Because this container's purpose is to run a web server, we need to make sure to map our host and container ports.

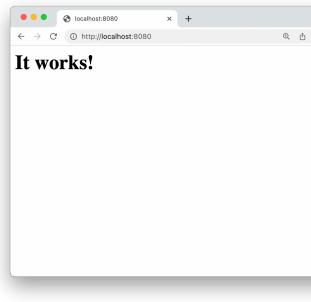
```
docker run -it --rm -p 8080:80 hw02:latest
```



Creating Images

Getting Started Basics

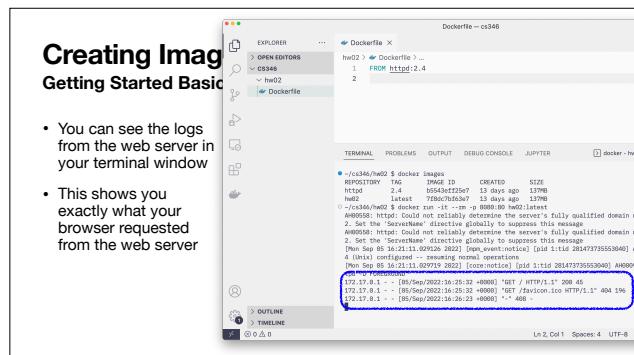
- If everything worked out, you should be able to open a new browser tab and go to `http://localhost:8080` and see the default web page served up by the `httpd:2.4` container.



Creating Images

Getting Started Basics

- Remember, everything we did here was done **from the host computer** (i.e. your laptop). We aren't building or running anything from *inside* of a container.
 - With the exception of certain automated build environments you'll likely never run any `docker ...` command from *inside* of a container.



Creating Images Getting Started Basic

- To exit the container, press the **control** and **c** key together.
- This is often abbreviated as just **ctrl-c** or **^C**
- You can see the **^C** in the screenshot before the shutdown line

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a file tree with a Dockerfile under hw02.
- TERMINAL**: Displays the command \$ docker images followed by the output of the Docker image listing.
- OUTPUT**: Shows logs from a Docker container, including messages about SSL/TLS and a shutdown line with a control-C character.
- DEBUG CONSOLE**: Not visible in the screenshot.
- JUPYTER**: Not visible in the screenshot.

Demo

SSH Basics

Connecting to Remote Hosts

ssh - The Secure Shell

- “Back in my day” we connected to remote unix hosts with the `telnet` command
 - Plain text network traffic
 - No encryption
 - It’s horribly insecure!
- Can still be useful, but is often not installed by default anymore
 - Did I mention it’s *horribly insecure*?

Connecting to Remote Hosts

ssh - The Secure Shell

- The ssh program is better
 - End-to-end encryption
 - Can use passwords or public keys
 - ssh + public keys is very secure

```
ssh [username]@[hostname]  
ssh [username]@[IP Address]
```

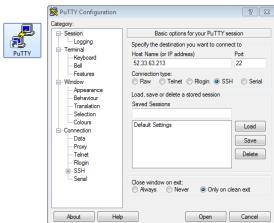
Connecting to Remote Hosts

ssh - The Secure Shell

- The ssh program is installed by default on macOS, Linux desktops, recent version of Windows, and the Windows Subsystem for Linux 2 (WSL2).
- If you prefer GUI apps on Windows, Putty is the default go-to

Connecting to Remote Hosts

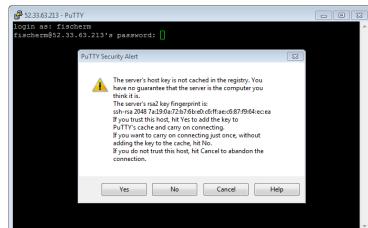
Putty



<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

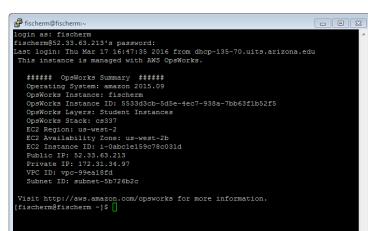
Connecting to Remote Hosts

Putty



Connecting to Remote Hosts

Putty



```
root@e6789f3d5d0:~/.work# ssh fischem@lectura.cs.arizona.edu
The authenticity of host 'lectura.cs.arizona.edu (192.12.69.186)' can't be established.
ECDSA key fingerprint is SHA256:eeHzzalbyHjs14kxr0ZINFCAX2jJnqAbPyREZE%Qg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'lectura.cs.arizona.edu,192.12.69.186' (ECDSA) to the list of known hosts.
fischem@lectura.cs.arizona.edu's password:

System information as of Sat 27 Aug 2022 04:17:45 PM MST
...
System load: 0.0      Processes:        491
Usage of /: 41.1K of 9.786GB  Users logged in: 15
Memory usage: 13%          IPv4 address for ens18: 192.12.69.186
Swap usage: 1K

=> There is 1 zombie process.
Welcome to:
```

Lectura

Shared Computer Science Host

- Our department hosts a shared UNIX server, named **lectura**.
- Before logging in, **create/reset your password**:
 - <https://helpdesk.cs.arizona.edu/selfservice>
- Your username will be same as NetID But your password can be different

```
ssh netid@lectura.cs.arizona.edu
```