

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.

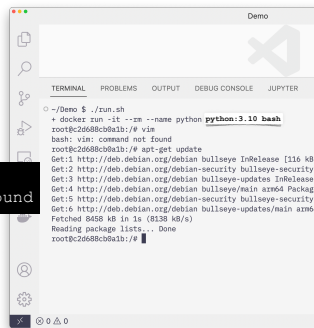
ubuntu

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 [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security
Get:3 http://deb.debian.org/debian bullseye-updates InRelease
Get:4 http://deb.debian.org/debian bullseye/main amd64 Package
Get:5 http://deb.debian.org/debian-security bullseye-security
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64
Fetched 8456 kB in 1s (8138 kB/s)
Reading package lists... Done
root@c2d688cb0a1b:/#
```

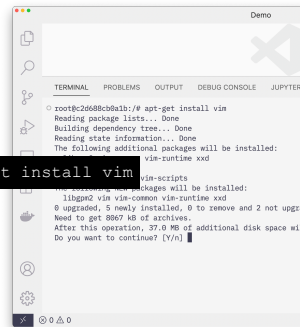
ubuntu

Installing software with `apt-get`

- Now use `apt-get install` to install `vim`

```
root@c2d688cb0a1b:/# apt-get install vim
```

- Type `y` then enter to continue and install



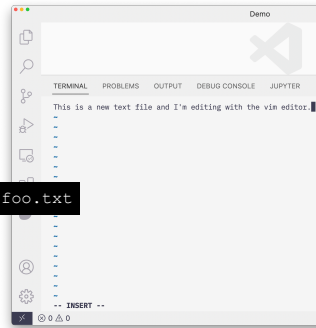
```
root@c2d688cb0a1b:/# apt-get install vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  vim-scripts
The following packages will be installed:
  libgnutls30 vim-common vim-runtime xxd
0 upgraded, 5 newly installed, 0 to remove and 2 not upgrd
Need to get 8867 kB of archives.
After this operation, 37.0 MB of additional disk space will
be required.
Do you want to continue? (Y/n)
```


ubuntu

Installing software with apt-get

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

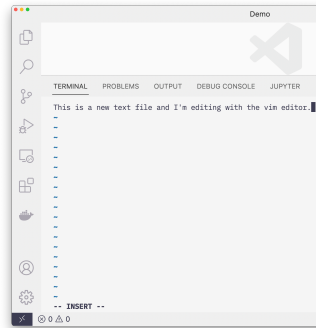
```
root@c2d688cb0a1b:/# vim foo.txt
```



ubuntu

vim basics

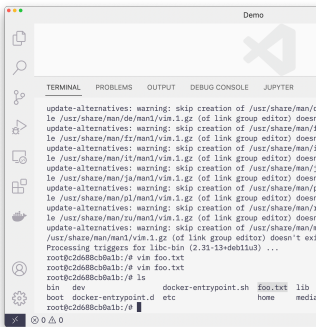
i	Enter insert mode
esc	Exit insert mode
arrow keys	Move the cursor around
:w	Save your changes (when not in insert mode)
:wq	Save your changes and exit vim (when not in insert mode)
:q!	Force quite vim and discard all changes (when not in insert mode)



ubuntu

Installing software with apt-get

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

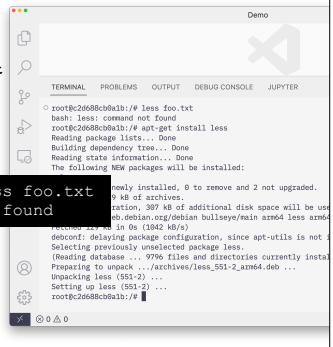


ubuntu

Installing software with apt-get

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

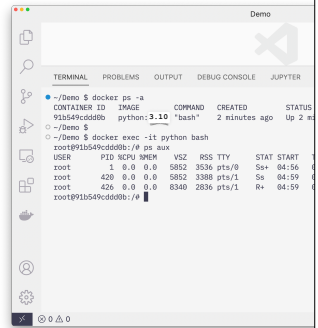
```
root@c2d688cb0a1b:/# less foo.txt
bash: less: command not found
```



Docker

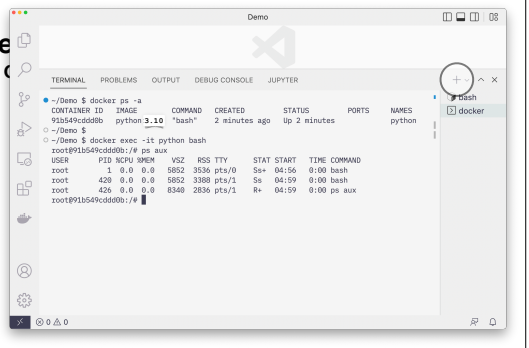
Multiple Container Connections

- 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



Docker

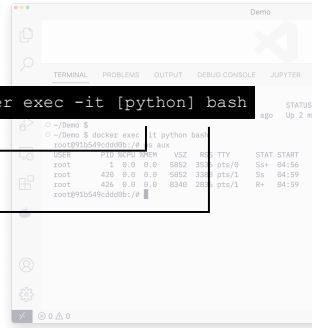
Multiple



Docker Multiple Container Connections

- 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

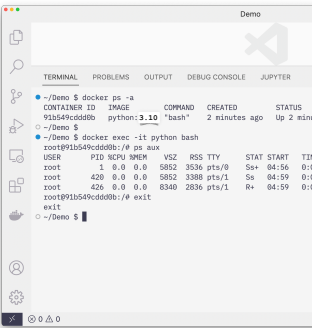
```
$ docker exec -it [python] bash
```



```
~/Demo $ docker exec -it [python] bash
root@911b549cd00b:/# ps aux
USER         PID %CPU %MEM    VSZ   TTU   STAT START   TIME
root           1  0.0  0.0   5852   pts/0    Ss+  04:56   0:00
root        426  0.0  0.0   5852  3388 pts/1    Ss+  04:59   0:00
root        426  0.0  0.0   8340  2836 pts/1    R+   04:59   0:00
root@911b549cd00b:/#
```

Docker Multiple Container Connections

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



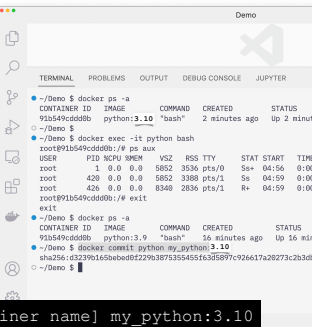
```
~/Demo $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
912549cd00b   python:3.10 "bash"    2 minutes ago   Up 2 minutes

~/Demo $ docker exec -it python bash
root@911b549cd00b:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME
root           1  0.0  0.0   5852  3536 pts/0    Ss+  04:56   0:00
root        420  0.0  0.0   5852  3388 pts/1    Ss+  04:59   0:00
root        426  0.0  0.0   8340  2836 pts/1    R+   04:59   0:00
root@911b549cd00b:/# exit
exit
~/Demo $
```

Docker Saving with docker commit

- 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
```



```
~/Demo $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
912549cd00b   python:3.9 "bash"    16 minutes ago   Up 16 min

~/Demo $ docker commit python_my_python:3.10
sha256:102770160be0e0522793873354516355971c926617a20273c2b30b

~/Demo $
```

Docker

Saving with `docker commit`

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

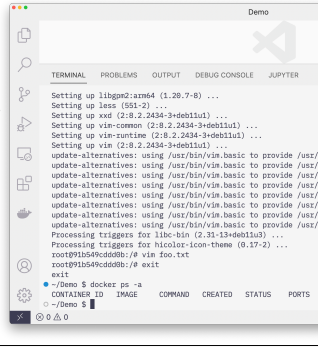


```
Terminal
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  JUPYTER
~/demo $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
my_python    3.10     632f961659eb   2 minutes ago  919MB
python       3.10     5880482c5a8    3 days ago    862MB
```

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



```
Terminal
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  JUPYTER
Setting up lispn2:amd64 (1.28-7-8) ...
Setting up less (591-2) ...
Setting up xxd (2:8.2.2434-3-deb11u) ...
Setting up vim-common (2:8.2.2434-3-deb11u) ...
Setting up vim-runtime (2:8.2.2434-3-deb11u) ...
Setting up vim (2:8.2.2434-3-deb11u) ...
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
update-alternatives: using /usr/bin/vim.basic to provide /usr/bi
Processing triggers for libc-bin (2.31-13+deb11u3) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
root@9116e9cd08b:/# vim foo.txt
root@9116e9cd08b:/# exit
exit
~/demo $ docker ps -a
CONTAINER ID   IMAGE    COMMAND                  CREATED   STATUS    PORTS
```

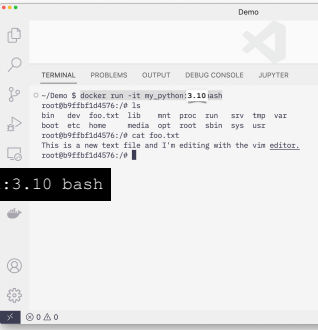
Docker

Saving with `docker commit`

- We can now run a new container based off of our new image

```
$ docker run -it my_python:3.10 bash
```

- Our `foo.txt` file is still there.

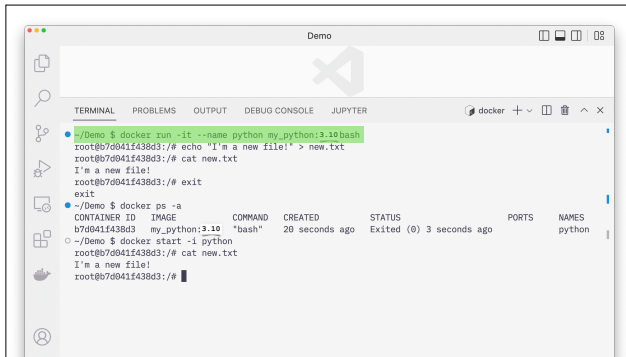


```
Terminal
~/demo $ docker run -it my_python:3.10 bash
root@9116e9cd08b:/# ls
bin dev foo.txt lib mnt proc run svx tmp var
boot etc home media opt root/sbin sys usr
root@9116e9cd08b:/# cat foo.txt
This is a new text file and I'm editing with the vim editor.
root@9116e9cd08b:/#
```

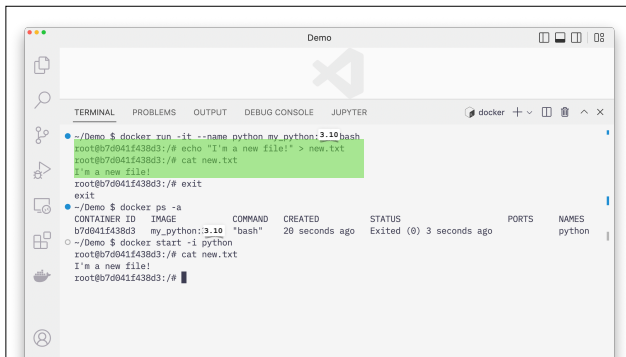
Docker

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.



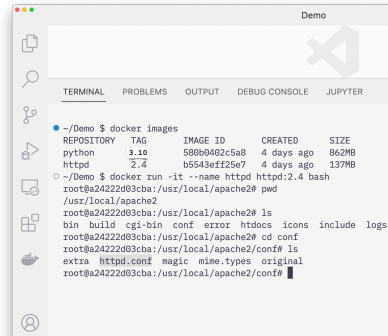
```
~/Demo $ docker run -it --name python my_python:3.10bash
root@b7d041f438d3:/# echo "I'm a new file!" > new.txt
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS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



```
~/Demo $ docker run -it --name python my_python:3.10bash
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   my_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 Copying Files

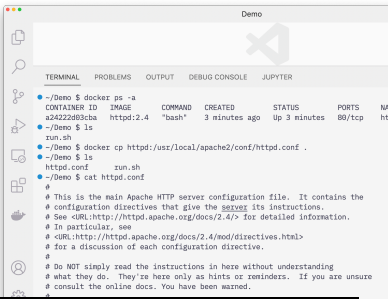
- Run a new container using the `httpd:2.4` image
- Look at the default directory we start in
- Change to the `conf` directory
- Look for the `httpd.conf` file



```
~/Demo $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
python 3.10 580b0402c5a8 4 days ago 862MB
httpd 2.4 b5543eff25e7 4 days ago 137MB
~/Demo $ docker run -it --name httpd httpd:2.4 bash
root@a24222d03cba:/usr/local/apache2# pwd
/usr/local/apache2
root@a24222d03cba:/usr/local/apache2# ls
bin build cgi-bin conf error htdocs 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"



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

~/Demo $ ls
run.sh
~/Demo $ docker cp httpd:/usr/local/apache2/conf/httpd.conf .
~/Demo $ ls
httpd.conf run.sh
~/Demo $ cat httpd.conf
#
# This is the main Apache HTTP server configuration file. It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.4/road/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.
```

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

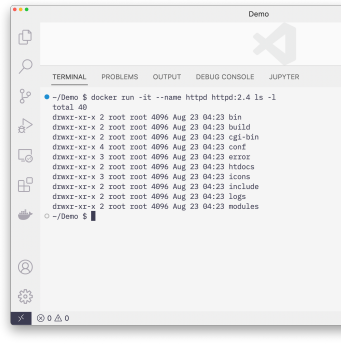
Docker Copying Files

- 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

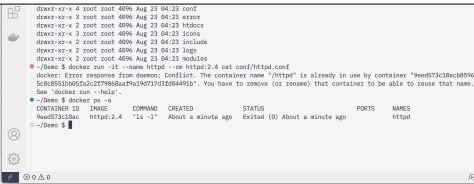
- You don't have to just run a new bash shell inside of a container.
- We can just run the `ls` command
- Or just a `cat` command.



```
Terminal Problems Output Debug Console Jupyter
~/demo $ docker run -it --name httpd:2.4 ls -l
total 40
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 3 root root 4096 Aug 23 04:23 error
drwxr-xr-x 2 root root 4096 Aug 23 04:23 htdocs
drwxr-xr-x 3 root root 4096 Aug 23 04:23 icons
drwxr-xr-x 2 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
~/demo $
```

Docker Run Errors

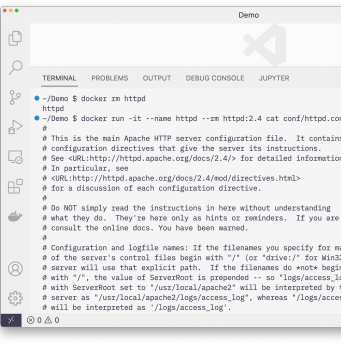
- Hmm what happened to cause our error?
- We tried to run a new container with a name of `httpd`, but we did not remove the first one
- You can't have two containers with the same name on a host at the same time



```
~/demo $ docker run -it --name httpd --rm httpd:2.4 cat conf/httpd.conf
docker: Error response from daemon: Conflict. The container name 'httpd' is already in use by container '9ead97313bc4809615cb81951d0915a2c3f7984ba9a19713d3d84641d'. You have to remove (or rename) that container to be able to reuse that name. See 'docker run --help'.
~/demo $ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
9ead97313bc4      httpd:2.4          "ls -l"            About a minute ago   Exited (0)          About a minute ago   httpd
~/demo $
```

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



```
~/demo $ docker rm httpd
httpd
~/demo $ docker run -it --name httpd --rm httpd:2.4 cat conf/httpd.conf
#
# This is the main Apache HTTP server configuration file. It contains
# configuration directives that give the server its instructions.
# See http://httpd.apache.org/docs/2.4/ for detailed information.
# In particular, see
# http://httpd.apache.org/docs/2.4/readingdirectives.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.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:" for windows)
# the server will use that explicit path. If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "logs/access_log"
# with ServerRoot set to "/usr/local/apache" will be interpreted by the
# server as "/usr/local/apache/logs/access_log", whereas "/logs/access
# will be interpreted as "/logs/access_log".

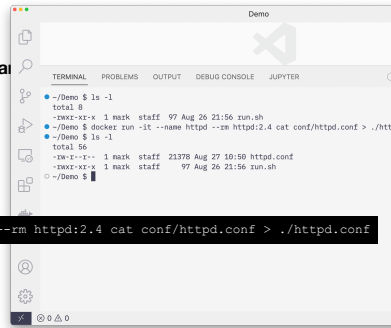
```

Docker Other Container Commands

- On macOS, Linux, and Windows with **WSL2** 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
EXPLORER
  OPEN EDITORS
  DEMO
  .bash_history
  httpd.conf
  run.sh
  run.sh
  1 $ /bin/bash -c#
  2 docker run \
  3 -it \
  4 --rm \
  5 --name python \
  6 --volume $PWD:/root \
  7 python3 test.py
  8 bash
  9
  TERMINAL
  PROBLEMS
  OUTPUT
  DEBUG CONSOLE
  JUPYTER
  bash
  ~/Demo $ ls -l
  total 56
  -rw-r--r-- 1 mack staff 21376 Aug 27 10:50 httpd.conf
  -rwxr-xr-x 1 mack staff 122 Aug 27 11:45 run.sh
  ~/Demo $ ./run.sh
  + docker run -it --rm --name python --volume /Users/mack/Demo:/root python 3.10 bash
  root@750c7188f1:/# cd /root
  root@750c7188f1:/# ls -l
  total 56
  -rw-r--r-- 1 root root 21376 Aug 27 17:59 httpd.conf
  -rwxr-xr-x 1 root root 122 Aug 27 18:45 run.sh
  root@750c7188f1:/#
```

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
EXPLORER
  OPEN EDITORS
  DEMO
  work
  hello.py
  run.sh
  run.sh
  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
  TERMINAL
  PROBLEMS
  OUTPUT
  DEBUG CONSOLE
  JUPYTER
  bash
  ~/Demo $ ./run.sh
  + docker run -it --rm --name python --volume /Users/mack/Demo/work:/root/work python 3.10 bash
  root@e47899335d1:/# cd /work/
  root@e47899335d1:/# ls -l
  total 0
  -rwxr-xr-x 3 root root 96 Aug 27 18:54 work
  root@e47899335d1:/# cd work/
  root@e47899335d1:/work# ls -l
  total 4
  -rw-r--r-- 1 root root 173 Aug 27 19:03 hello.py
  root@e47899335d1:/work# python hello.py
  Hello! It is currently Saturday, August 27 at 07:04 PM.
  root@e47899335d1:/work#
```

Demo

Creating Images

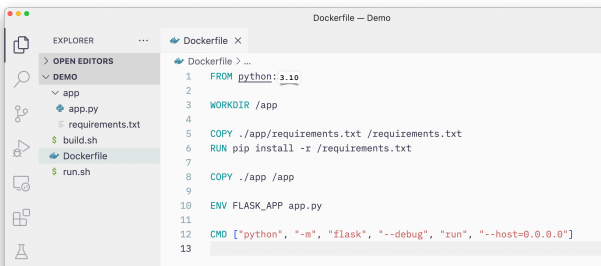
The Dockerfile

- 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

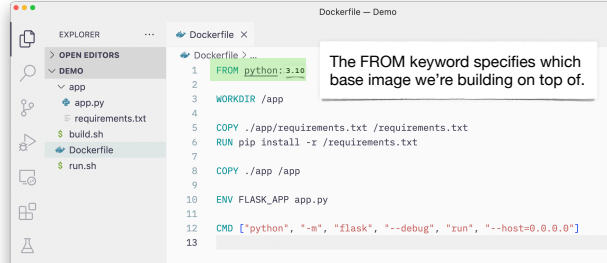
The Dockerfile



```
Dockerfile - Demo
EXPLORER
  OPEN EDITORS
  DEMO
    app
      app.py
      requirements.txt
    build.sh
    Dockerfile
    run.sh
  Dockerfile X
  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

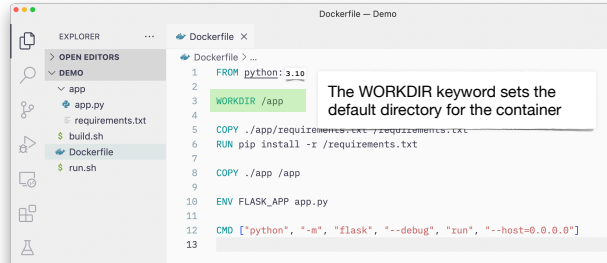


The screenshot shows a code editor with a Dockerfile. The first line is `FROM python:3.10`, which is highlighted in green. A callout box points to this line with the text: "The FROM keyword specifies which base image we're building on top of." The rest of the Dockerfile content is visible in the background.

```
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

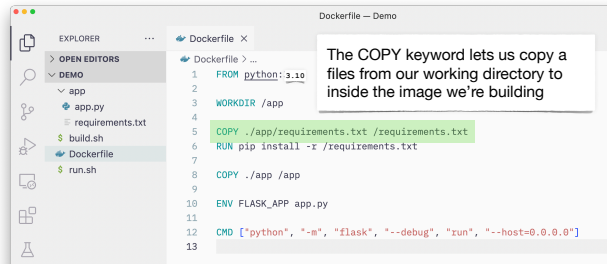


The screenshot shows a code editor with a Dockerfile. The second line is `WORKDIR /app`, which is highlighted in green. A callout box points to this line with the text: "The WORKDIR keyword sets the default directory for the container." The rest of the Dockerfile content is visible in the background.

```
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

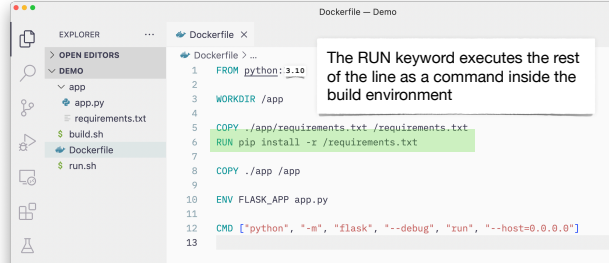


The screenshot shows a code editor with a Dockerfile. The fifth line is `COPY ./app/requirements.txt /requirements.txt`, which is highlighted in green. A callout box points to this line with the text: "The COPY keyword lets us copy a files from our working directory to inside the image we're building." The rest of the Dockerfile content is visible in the background.

```
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



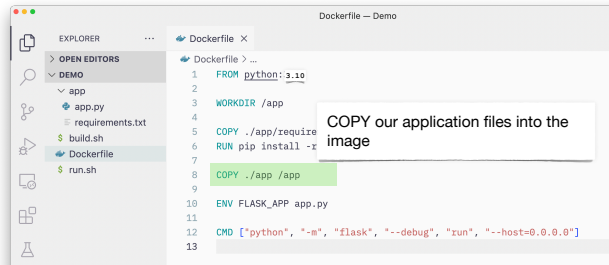
The screenshot shows a VS Code editor with a Dockerfile open. The Explorer sidebar on the left shows a project structure with files like app.py, requirements.txt, build.sh, Dockerfile, and run.sh. The Dockerfile content is as follows:

```
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
```

A callout box points to line 6, containing the text: "The RUN keyword executes the rest of the line as a command inside the build environment".

Creating Images

The Dockerfile



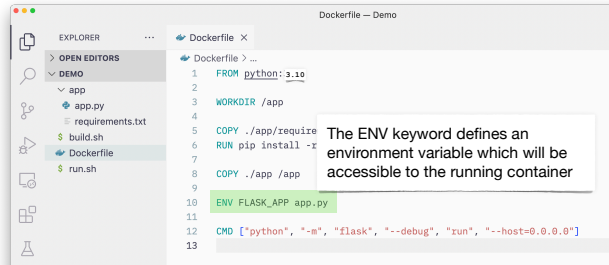
The screenshot shows a VS Code editor with a Dockerfile open. The Explorer sidebar on the left shows a project structure with files like app.py, requirements.txt, build.sh, Dockerfile, and run.sh. The Dockerfile content is as follows:

```
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
```

A callout box points to line 5, containing the text: "COPY our application files into the image".

Creating Images

The Dockerfile



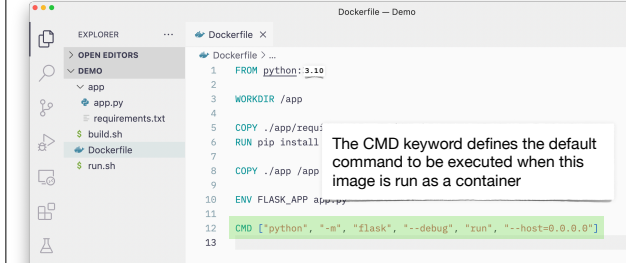
The screenshot shows a VS Code editor with a Dockerfile open. The Explorer sidebar on the left shows a project structure with files like app.py, requirements.txt, build.sh, Dockerfile, and run.sh. The Dockerfile content is as follows:

```
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
```

A callout box points to line 10, containing the text: "The ENV keyword defines an environment variable which will be accessible to the running container".

Creating Images

The Dockerfile



Creating Images

The Dockerfile

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

```
docker build --tag [image name]:[tag] [location]
```

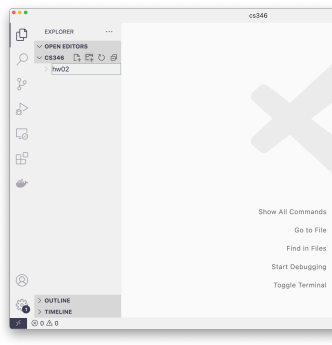
```
docker build --tag my_app:latest .
```

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

Creating Images

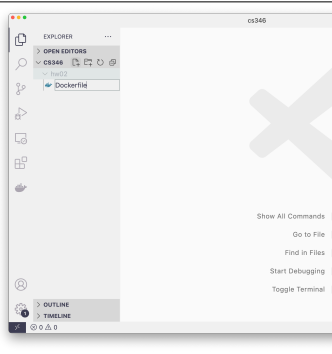
Getting Started Basics

- 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`.



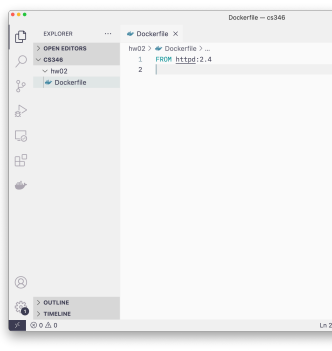
Creating Images Getting Started Basics

- Next we need to create a new empty text file inside our project folder, and name it **Dockerfile**



Creating Images Getting Started Basics

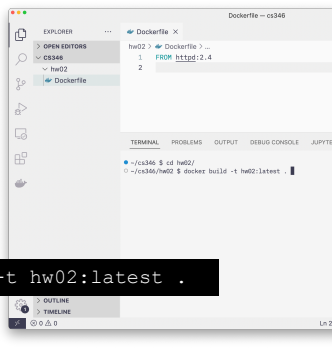
- With our newly created Dockerfile open in an editor, we can start with the most basic directive, and just have a **FROM httpd:2.4** line in our file.



Creating Images Getting Started Basics

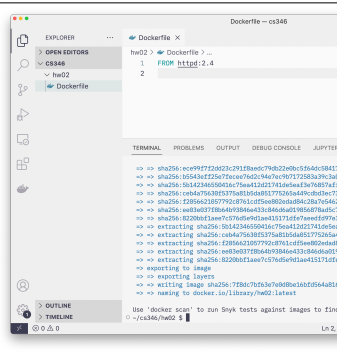
- Don't forget to **save** your **Dockerfile** before you build it!
- Make sure your terminal session is currently in your project folder.
- Build our new image with the **docker build** command:

```
docker build -t hw02:latest .
```



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.

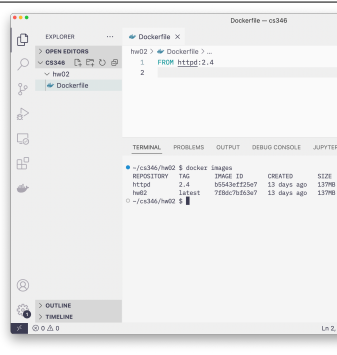


The screenshot shows the Dockerfile build process in VS Code. The terminal output displays the following steps:

```
=> sha256:ec99f72d023c791f8e6c79b20e0c16464841f
=> sha256:0645422274eac760c7440c0c1730303c3a6
=> sha256:0a4234609045c75ea12d21f11ea43b708741f
=> sha256:0a4234609045c75ea12d21f11ea43b708741f
=> sha256:2f864210977910f1c45e802648428705441
=> sha256:0a4234609045c75ea12d21f11ea43b708741f
=> sha256:82200f1aee7c5769591aa411517516f
=> extracting sha256:5d4334609045c75ea12d21f11ea43b708741f
=> extracting sha256:0a4234609045c75ea12d21f11ea43b708741f
=> extracting sha256:2f864210977910f1c45e802648428705441
=> extracting sha256:0a4234609045c75ea12d21f11ea43b708741f
=> extracting sha256:82200f1aee7c5769591aa411517516f
=> exporting image
=> exporting layers
=> writing image sha256:7f86c70c3476b846b46464841f
=> naming to docker.io/library/hw02:latest
```

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.

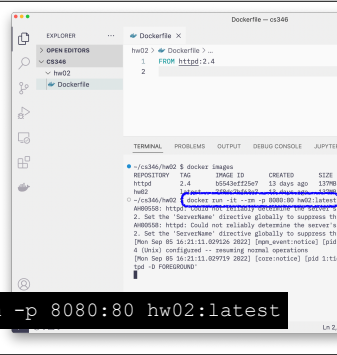


The screenshot shows the Dockerfile build process in VS Code. The terminal output displays the following steps:

```
./cs346/hw02 $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hw02 2.4 8564825507 13 days ago 1379B
hw02 latest 7f86c70c347 13 days ago 1379B
```

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.



The screenshot shows the Dockerfile build process in VS Code. The terminal output displays the following steps:

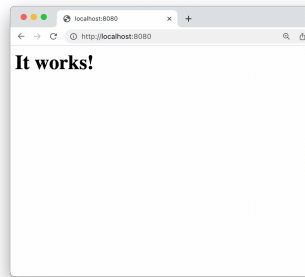
```
./cs346/hw02 $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hw02 2.4 8564825507 13 days ago 1379B
hw02 latest 7f86c70c347 13 days ago 1379B

./cs346/hw02 $ docker run -it --rm -p 8080:80 hw02:latest
AWROSD: http://10.0.0.1:8080/
2. Set the "serveindex" directive globally to suppress the
AWROSD: http:// Could not reliably determine the server's
2. Set the "serveindex" directive globally to suppress the
[Sun Sep 05 16:21:11.807236 2020] [mpm_event_notified] pid 4
[init] configured -- resuming normal operations
[Sun Sep 05 16:21:11.807236 2020] [core_notice] pid 1:11
[init] -D FOREGROUND
```

```
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.

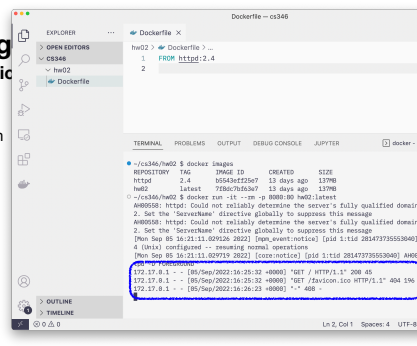


Creating Images Getting Started Basics

- 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.

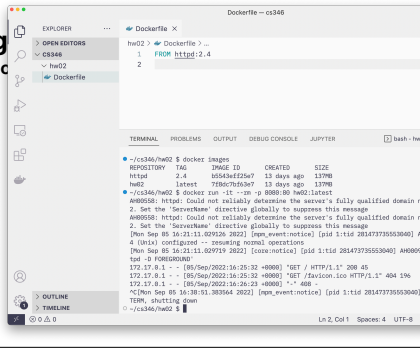
Creating Images Getting Started Basics

- You can see the logs from the web server in your terminal window
- This shows you exactly what your browser requested from the web server



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 Docker Desktop interface. The Explorer on the left lists 'hw02' under 'Dockerfile'. The main area displays the terminal output of the 'hw02' container. The output shows the container creation process, including the Dockerfile used to create it, and the execution of a 'curl' command to access 'http://localhost:8080'. The container is shown as 'Running' and 'Terminated'.

```
~/c346/hw02 $ docker images
REPOSITORY          TAG             IMAGE ID         CREATED         SIZE
httpd                2.4             6554b4f25e7     13 days ago    137MB
hw02                 latest         7f86c7856a7     13 days ago    137MB

~/c346/hw02 $ docker run -it --rm -p 8080:80 hw02:latest
#000000: httpd: Could not reliably determine the server's fully qualified domain n
#000000: httpd: Could not reliably determine the server's fully qualified domain n
#000000: httpd: Could not reliably determine the server's fully qualified domain n
2: Set the 'ServerName' directive globally to suppress this message
#000008: httpd: Could not reliably determine the server's fully qualified domain n
[Mon Sep 05 16:21:11.029126 2022] [warn_event-notice] [pid 1:tid 28147373503040] #
4 (initial config) -- receiving normal, operation
[Mon Sep 05 16:21:11.029126 2022] [score-notice] [pid 1:tid 28147373503040] AH000
test 2
#0000000000
172.17.0.1 - - [05/Sep/2022:16:26:32 +0000] "GET / HTTP/1.1" 200 45
172.17.0.1 - - [05/Sep/2022:16:26:32 +0000] "GET favicon.ico HTTP/1.1" 404 196
172.17.0.1 - - [05/Sep/2022:16:26:29 +0000] "-" 409 -
^C[Mon Sep 05 16:26:13.383864 2022] [warn_event-notice] [pid 1:tid 28147373503040]
TERM, shutting down

~/c346/hw02 $
```

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]
```

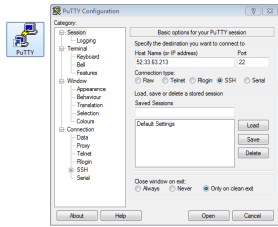
```
ssh [username]@[IP Address]
```

Connecting to Remote Hosts

ssh - The Secure Shell

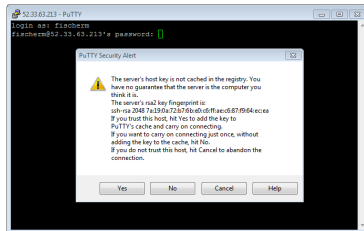
- The ssh program is installed by default on macOS, Linux desktops, recent version of Windows, and the Windows Subsystem for Linux 2 (WSL2).
- If you prefer GUI apps on Windows, Putty is the default go-to

Connecting to Remote Hosts Putty

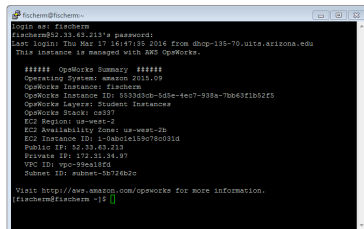


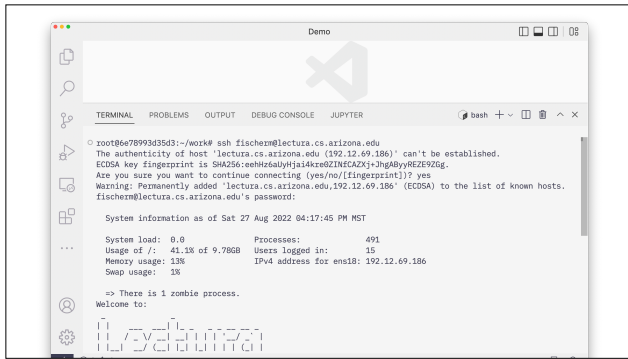
<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

Connecting to Remote Hosts Putty



Connecting to Remote Hosts Putty





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
```

