

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.

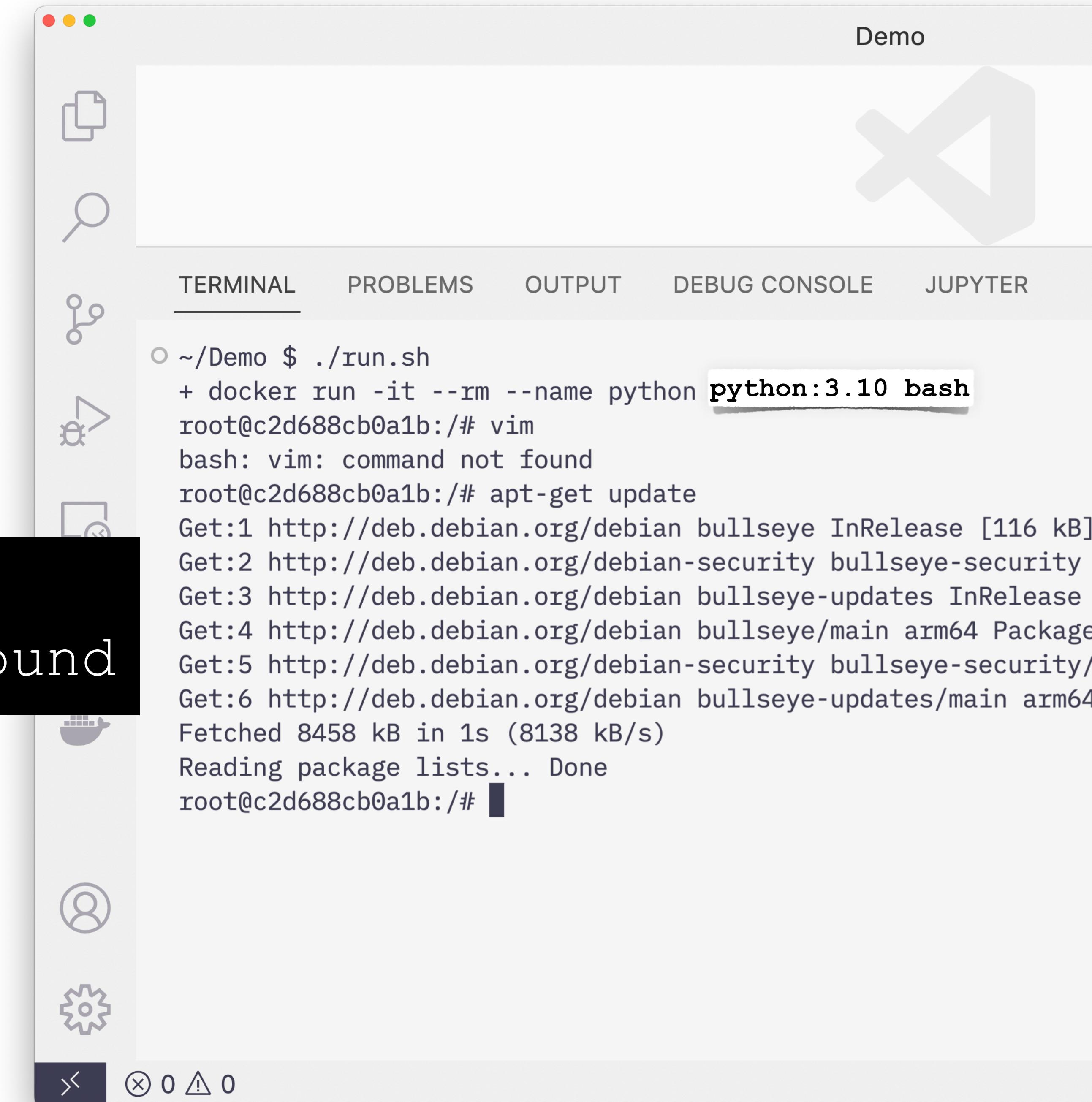
ubuntu

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



The screenshot shows a terminal window within a development interface. The window title is "Demo". The tab bar at the top includes "TERMINAL", "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "JUPYTER". The "TERMINAL" tab is selected. The terminal content displays the following command and its execution:

```
~/Demo $ ./run.sh  
+ docker run -it --rm --name python python:3.10 bash  
root@c2d688cb0a1b:/# vim  
bash: vim: command not found  
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 arm64 Packages  
Get:5 http://deb.debian.org/debian-security bullseye-security/  
Get:6 http://deb.debian.org/debian bullseye-updates/main arm64 Packages  
Fetched 8458 kB in 1s (8138 kB/s)  
Reading package lists... Done  
root@c2d688cb0a1b:/#
```

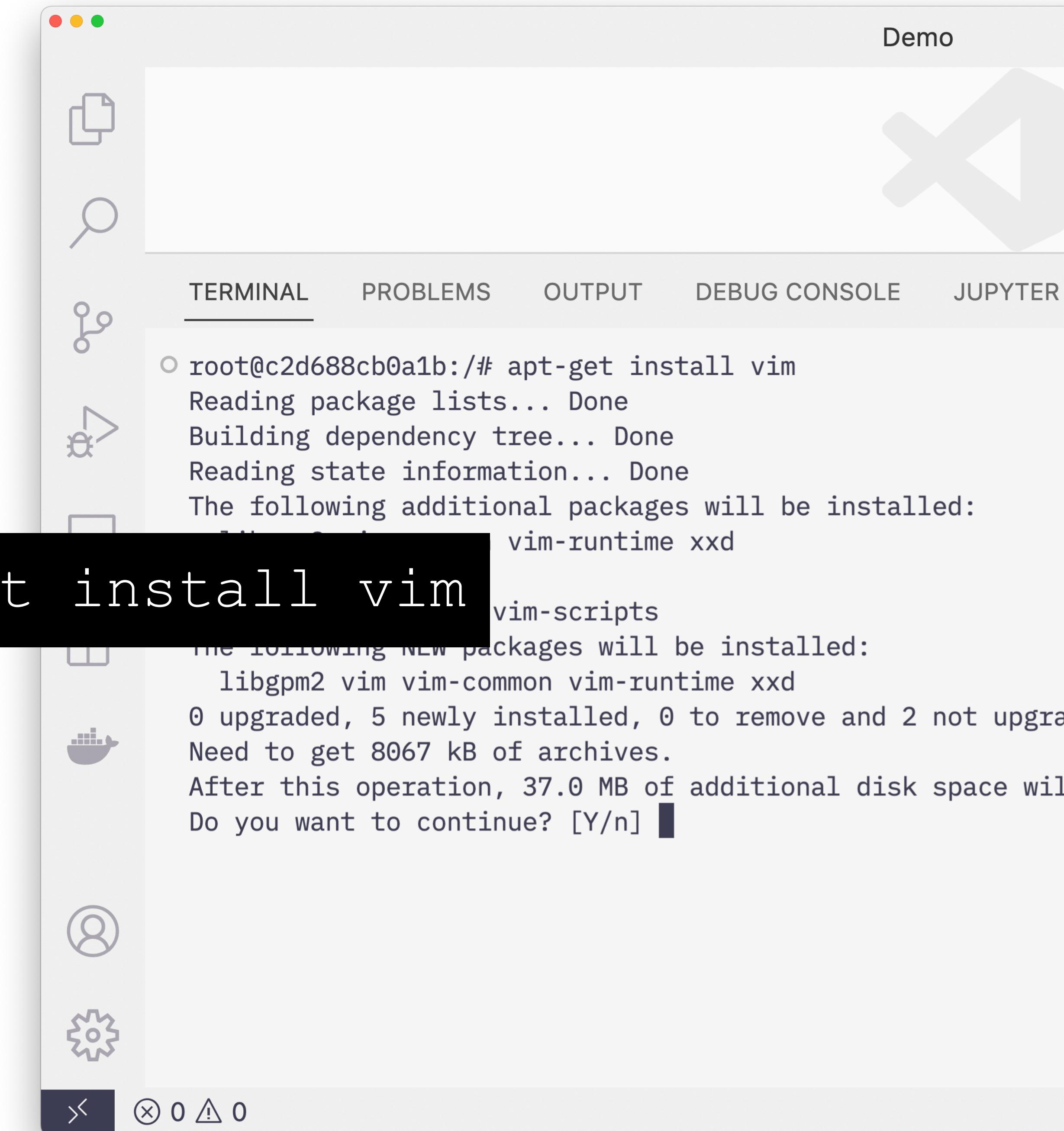
ubuntu

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



A screenshot of a terminal window titled "Demo". The window has a dark theme with light-colored text. The terminal tab is selected. The command "root@c2d688cb0a1b:/# apt-get install vim" is entered. The output shows the package lists being read, dependencies being built, and state information being checked. It then lists additional packages to be installed: vim-common, vim-runtime, vim-runtime, vim-scripts, libgpm2, vim, vim-common, vim-runtime, and xxd. It shows 0 upgraded, 5 newly installed, 0 to remove, and 2 not upgradeable. A message indicates 8067 kB of archives need to be downloaded. The final question asks if the user wants to continue [Y/n].

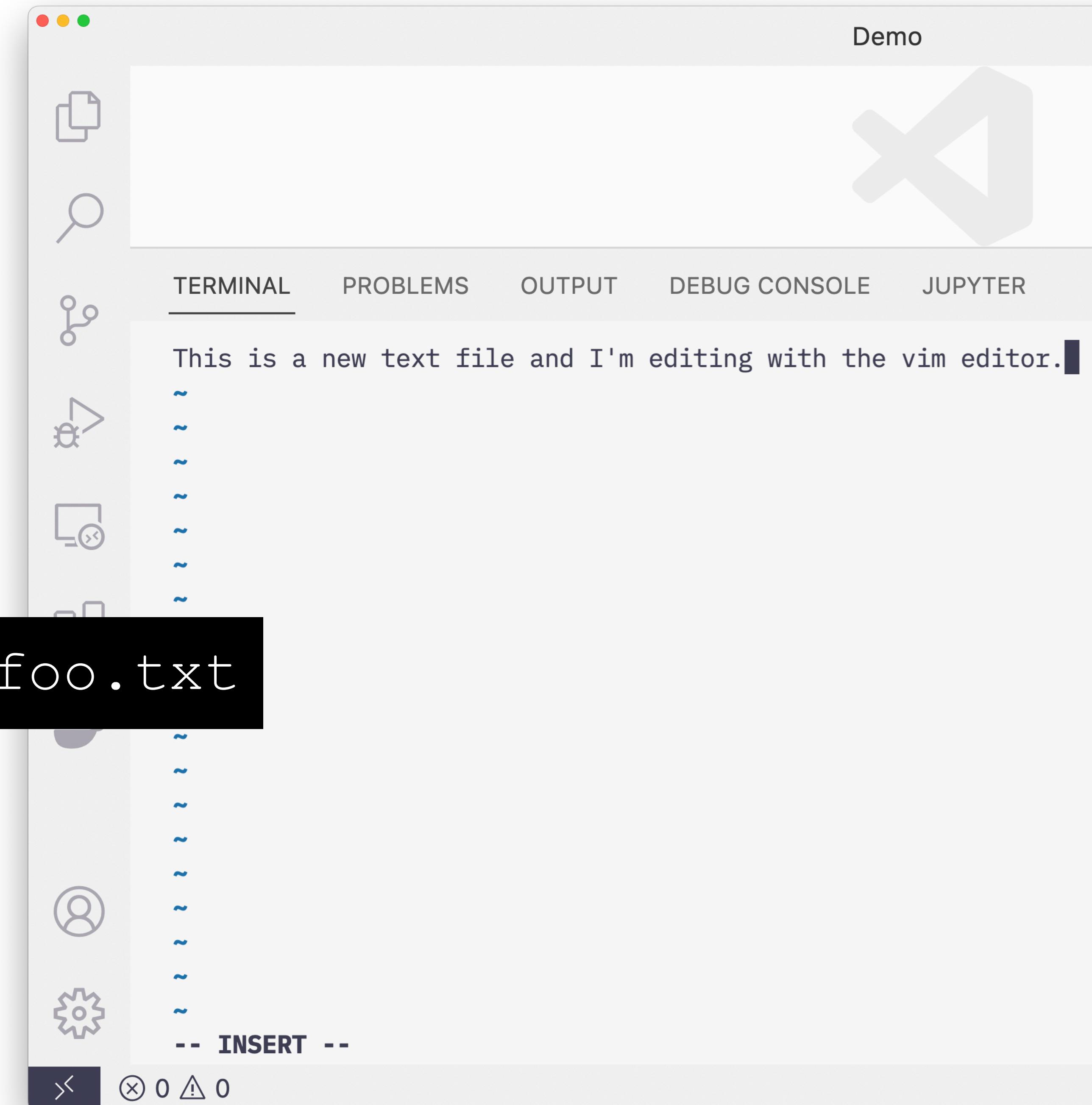
```
root@c2d688cb0a1b:/# apt-get install vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  vim-common vim-runtime vim-runtime vim-scripts
  libgpm2 vim vim-common vim-runtime xxd
0 upgraded, 5 newly installed, 0 to remove and 2 not upgradeable
Need to get 8067 kB of archives.
After this operation, 37.0 MB of additional disk space will be used.
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.

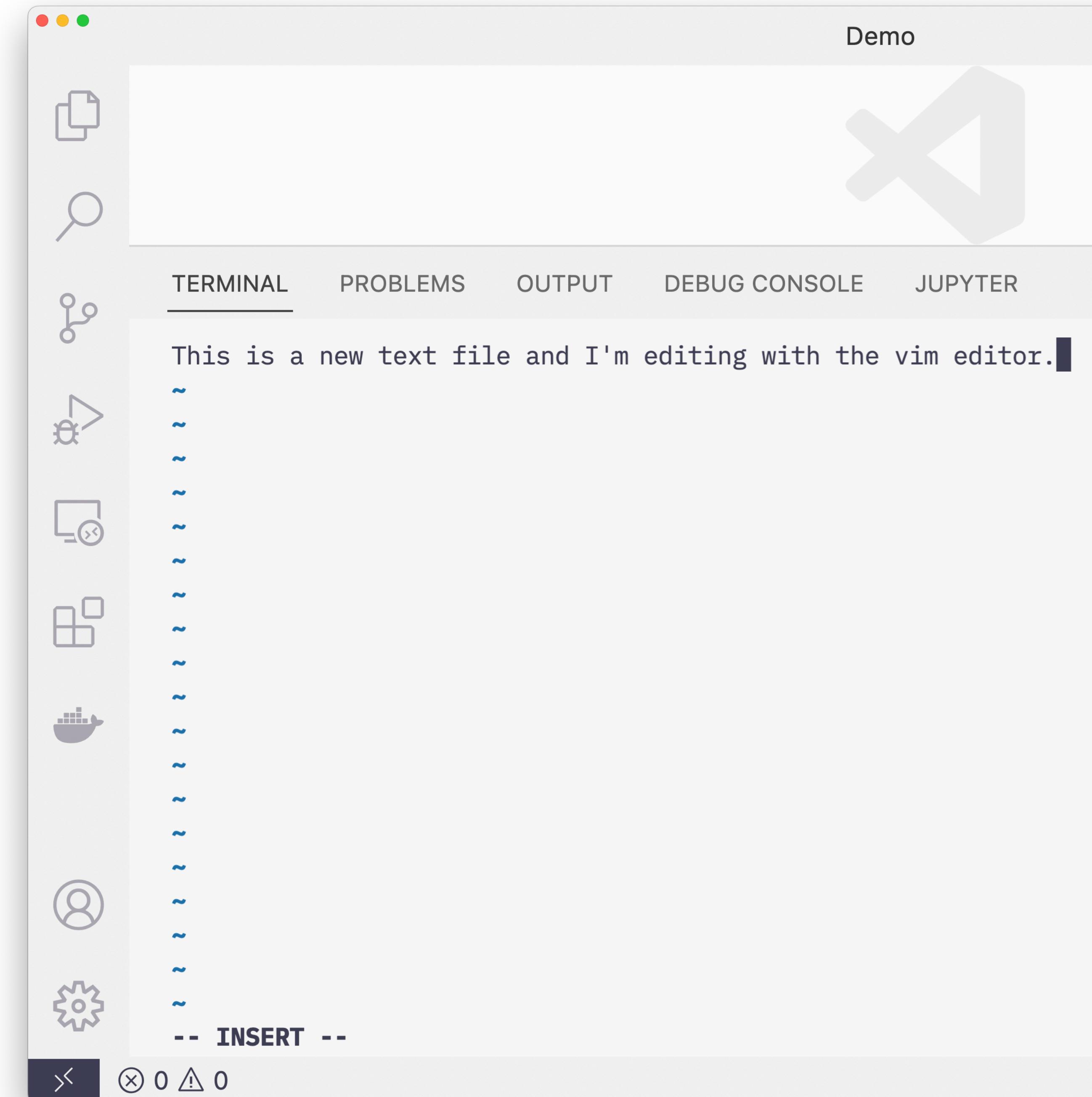
```
root@c2d688cb0a1b:/# vim foo.txt
```



ubuntu

vim basics

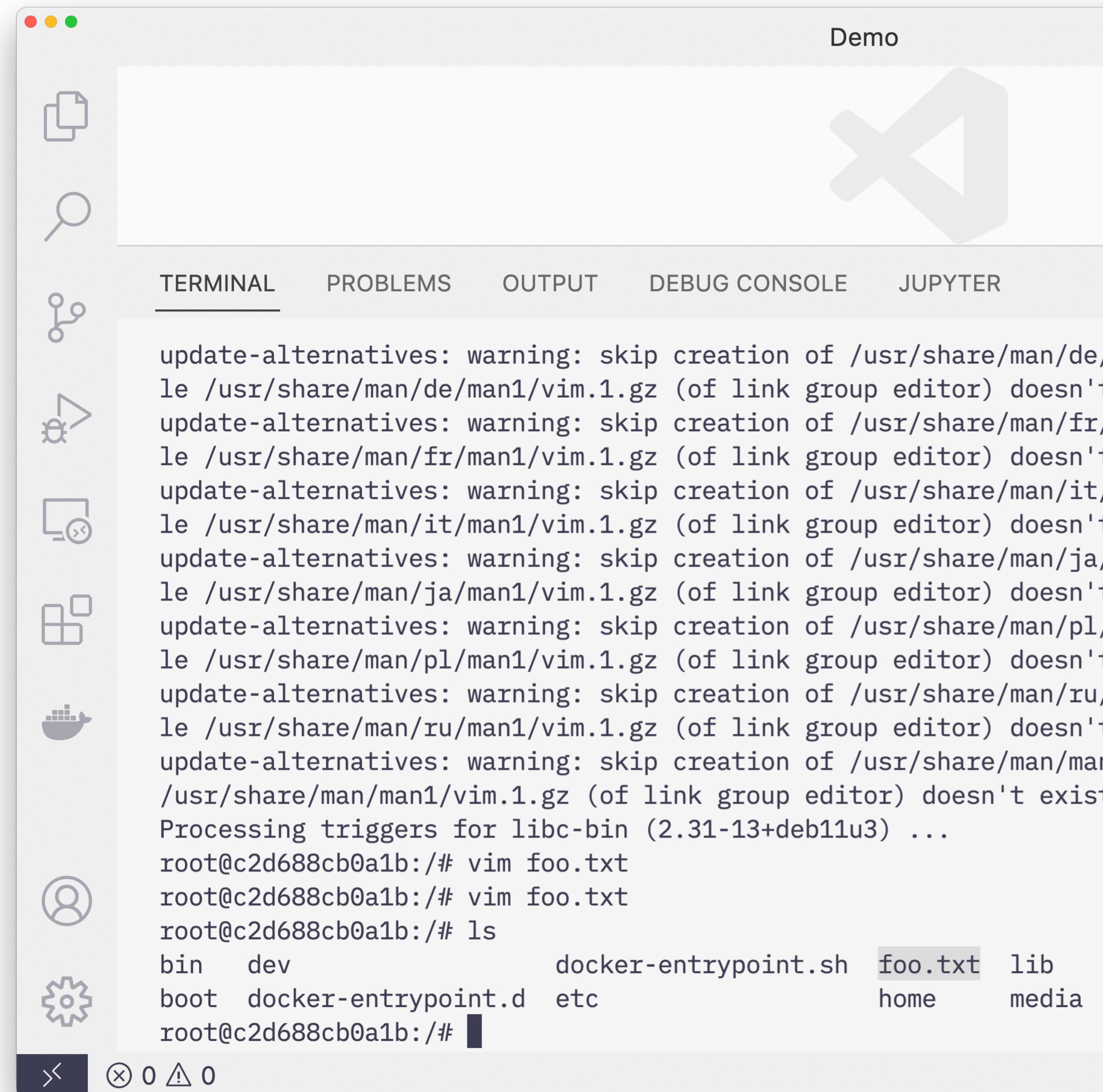
i	Enter insert mode
esc	Exit insert mode
arrow keys	Move the cursor around
:w	Save your changes (when not in insert mode)
:wq	Save your changes and exit vim (when not in insert mode)
:q!	Force quite vim and discard all changes (when not in insert mode)



ubuntu

Installing software with apt-get

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



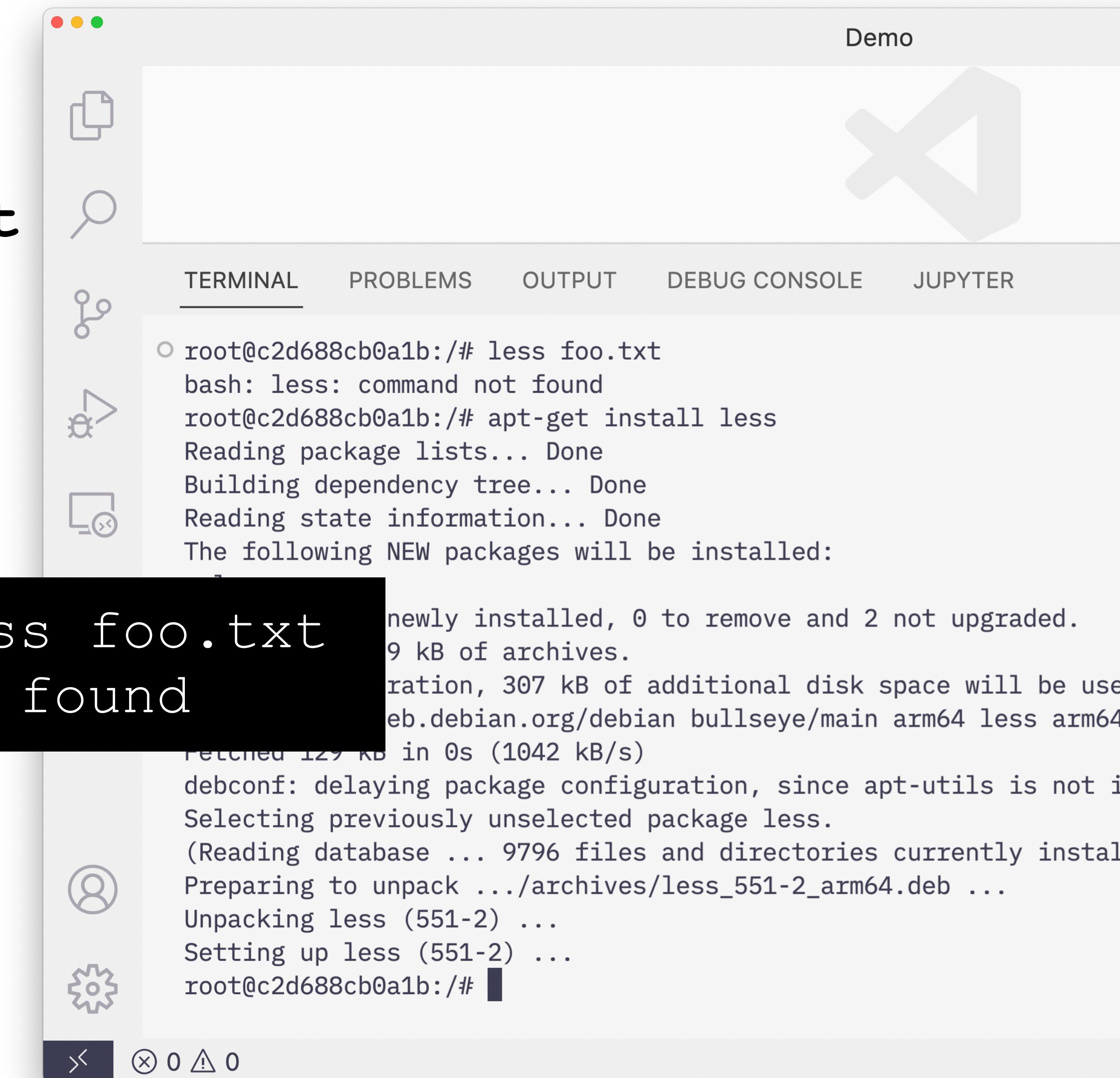
The screenshot shows a terminal window with the following output:

```
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/fr/le /usr/share/man/fr/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/it/le /usr/share/man/it/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ja/le /usr/share/man/ja/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/pl/le /usr/share/man/pl/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ru/le /usr/share/man/ru/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/le /usr/share/man/man1/vim.1.gz (of link group editor) doesn't exist
Processing triggers for libc-bin (2.31-13+deb11u3) ...
root@c2d688cb0a1b:/# vim foo.txt
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



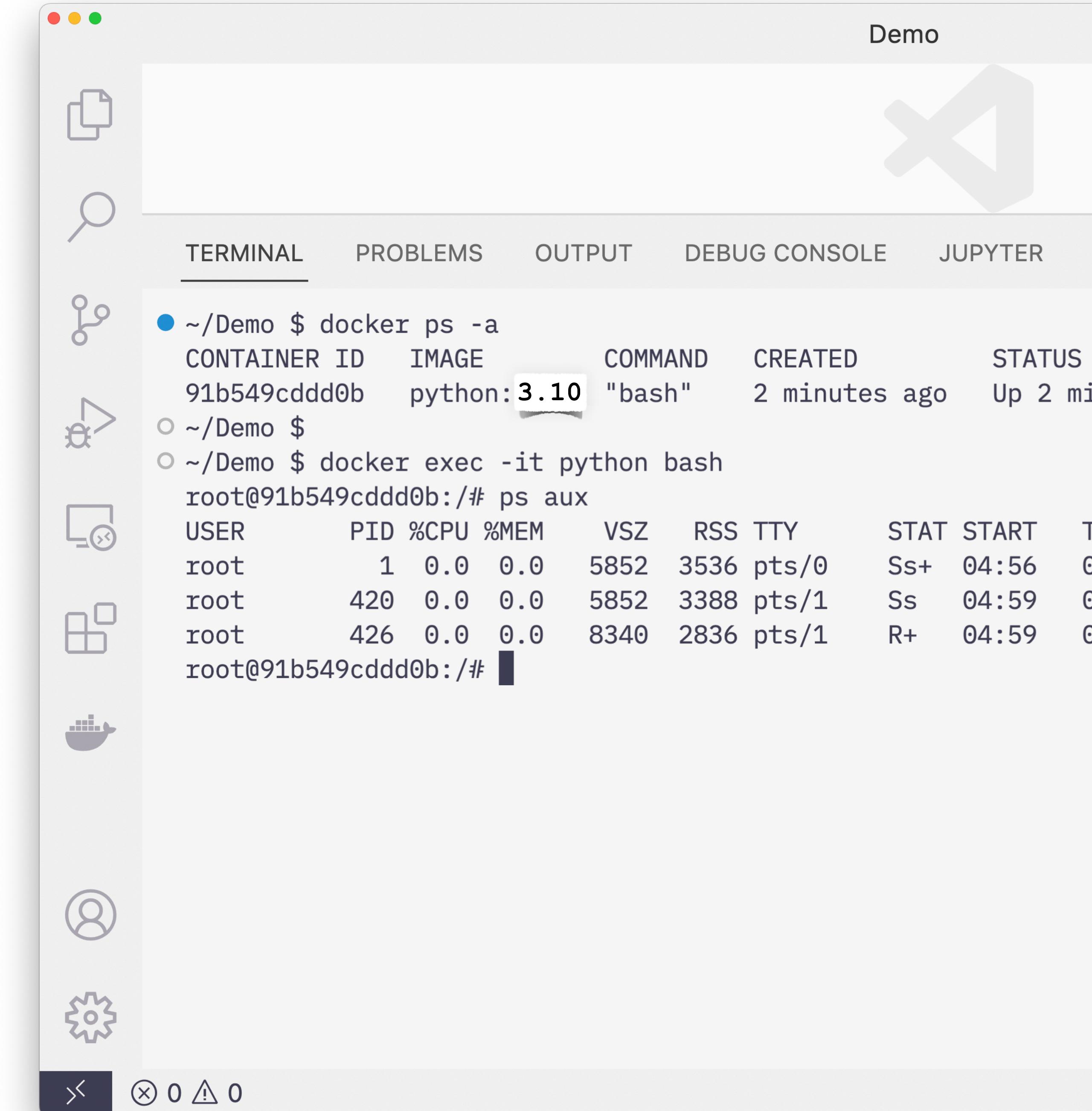
The screenshot shows a terminal window in the VS Code interface. The terminal tab is selected at the top. The output of the command `apt-get install less` is displayed, showing that the package was successfully installed.

```
root@c2d688cb0a1b:/# less foo.txt
bash: less: command not found
root@c2d688cb0a1b:/# apt-get install less
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  less
newly installed, 0 to remove and 2 not upgraded.
9 kB of archives.
  ration, 307 kB of additional disk space will be used.
  http://deb.debian.org/debian bullseye/main arm64 less arm64
Fetched 129 kB in 0s (1042 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package less.
(Reading database ... 9796 files and directories currently installed)
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



The screenshot shows the VS Code interface with a terminal tab open. The terminal window displays the output of a `docker ps -a` command, which lists a single container created by the user. The container ID is `91b549cddd0b`, it runs the `python:3.10` image, and its command is `"bash"`. It was created 2 minutes ago and is currently up. The terminal then shows the user executing `ps aux` inside the container, listing three processes: root at PID 1, root at PID 420, and root at PID 426. The terminal window has a dark theme and includes standard VS Code navigation icons (back, forward, search, etc.) at the bottom.

```
~/Demo $ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS     NAMES
91b549cddd0b        python:3.10        "bash"              2 minutes ago   Up 2 minutes          0.0.0.0:49153->5000/tcp
~/Demo $
~/Demo $ docker exec -it python bash
root@91b549cddd0b:/# 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@91b549cddd0b:/#
```

Docker Multiple C

Demo

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

● ~/Demo \$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
91b549cddd0b	python:3.10	"bash"	2 minutes ago	Up 2 minutes		python

○ ~/Demo \$

○ ~/Demo \$ docker exec -it python bash

```
root@91b549cddd0b:/# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	5852	3536	pts/0	Ss+	04:56	0:00	bash
root	420	0.0	0.0	5852	3388	pts/1	Ss	04:59	0:00	bash
root	426	0.0	0.0	8340	2836	pts/1	R+	04:59	0:00	ps aux

```
root@91b549cddd0b:/#
```

+ ^ X

bash

docker

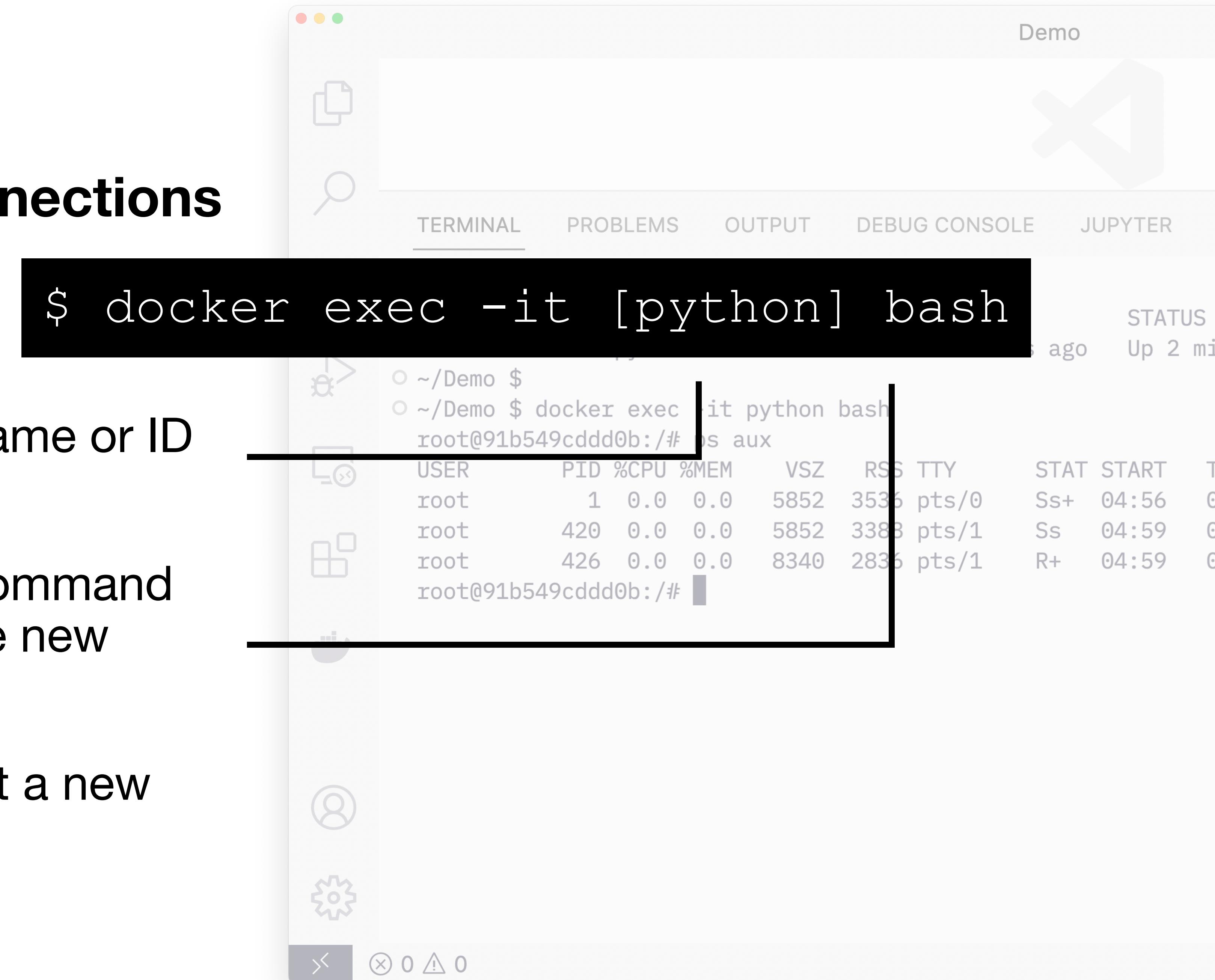
< > 0 ! 0

👤 🔔

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



The screenshot shows a Docker terminal session in VS Code. The terminal tab is active, displaying the command:

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

The output of the command is shown below:

```
~ /Demo $  
~ /Demo $ docker exec -it python bash  
root@91b549cddd0b:/# ls aux  
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME  
root 1 0.0 0.0 5852 3536 pts/0 Ss+ 04:56 0s  
root 420 0.0 0.0 5852 3383 pts/1 Ss 04:59 0s  
root 426 0.0 0.0 8340 2836 pts/1 R+ 04:59 0s  
root@91b549cddd0b:/#
```

The terminal interface includes icons for file operations (copy, paste, search) and navigation (back, forward, refresh). Status indicators at the bottom show 0 errors and 0 warnings.

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

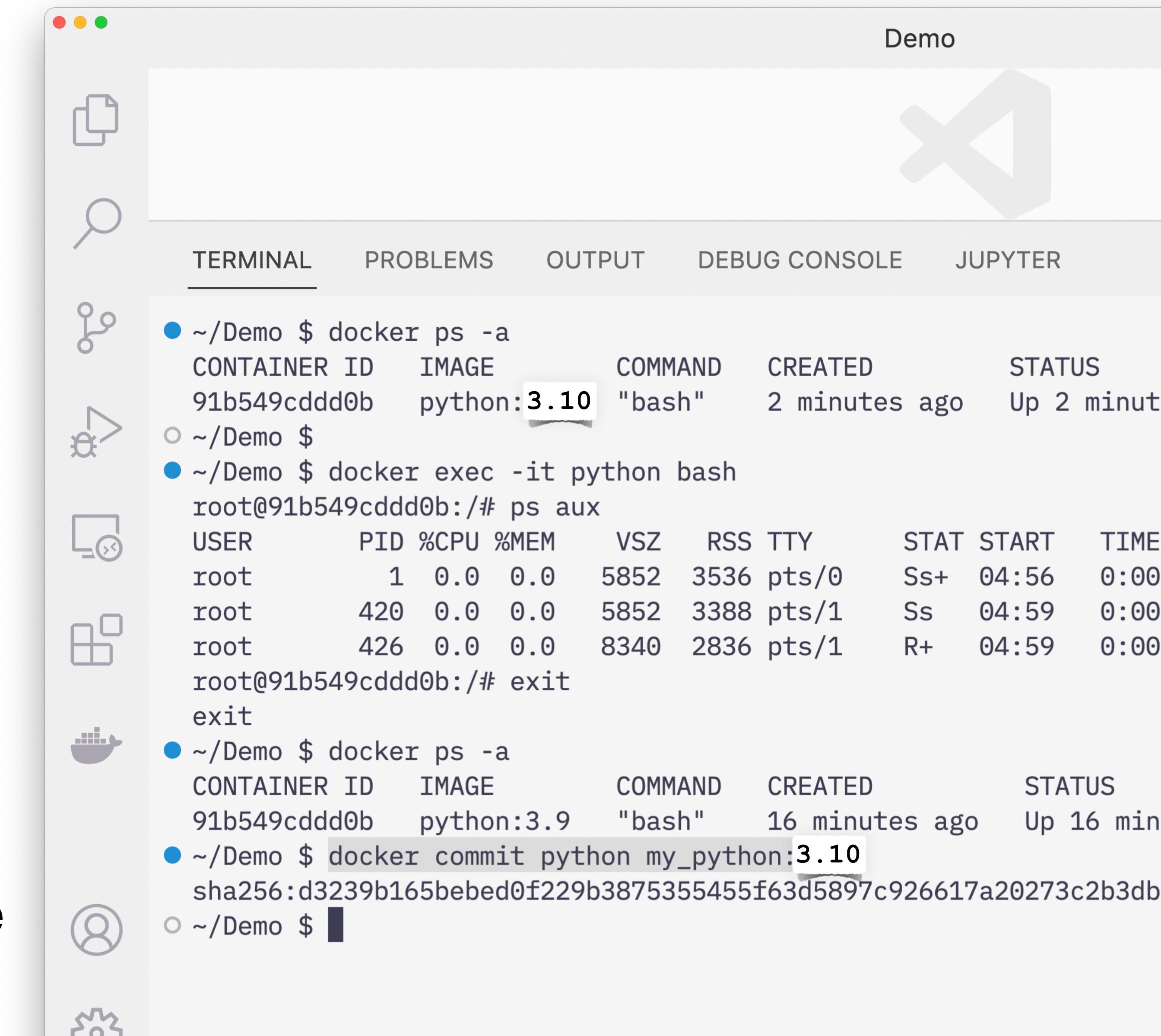
The screenshot shows a Docker terminal session in VS Code. The terminal tab is active, displaying the output of a `docker ps -a` command. A second terminal window is open, showing the result of a `docker exec -it python bash` command. Both windows show the same container ID, `91b549cddd0b`, indicating they are connected to the same container simultaneously.

```
~/Demo $ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS     NAMES
91b549cddd0b        python:3.10        "bash"              2 minutes ago    Up 2 minutes          0.0.0.0:4243->4243/tcp
~/Demo $
~/Demo $ docker exec -it python bash
root@91b549cddd0b:/# 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   0:0
root            420  0.0  0.0      5852    3388 pts/1    Ss    04:59   0:0
root            426  0.0  0.0      8340    2836 pts/1    R+    04:59   0:0
root@91b549cddd0b:/# 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.



The screenshot shows the VS Code interface with the 'Terminal' tab selected. The terminal window displays the following sequence of commands and their output:

```

● ~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
91b549cddd0b python:3.10 "bash" 2 minutes ago Up 2 minutes

○ ~/Demo $

● ~/Demo $ docker exec -it python bash
root@91b549cddd0b:/# 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@91b549cddd0b:/# exit
exit

● ~/Demo $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
91b549cddd0b python:3.9 "bash" 16 minutes ago Up 16 minutes

● ~/Demo $ docker commit python my_python:3.10
sha256:d3239b165bebed0f229b3875355455f63d5897c926617a20273c2b3db

○ ~/Demo $

```

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

Docker

Saving with docker commit

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

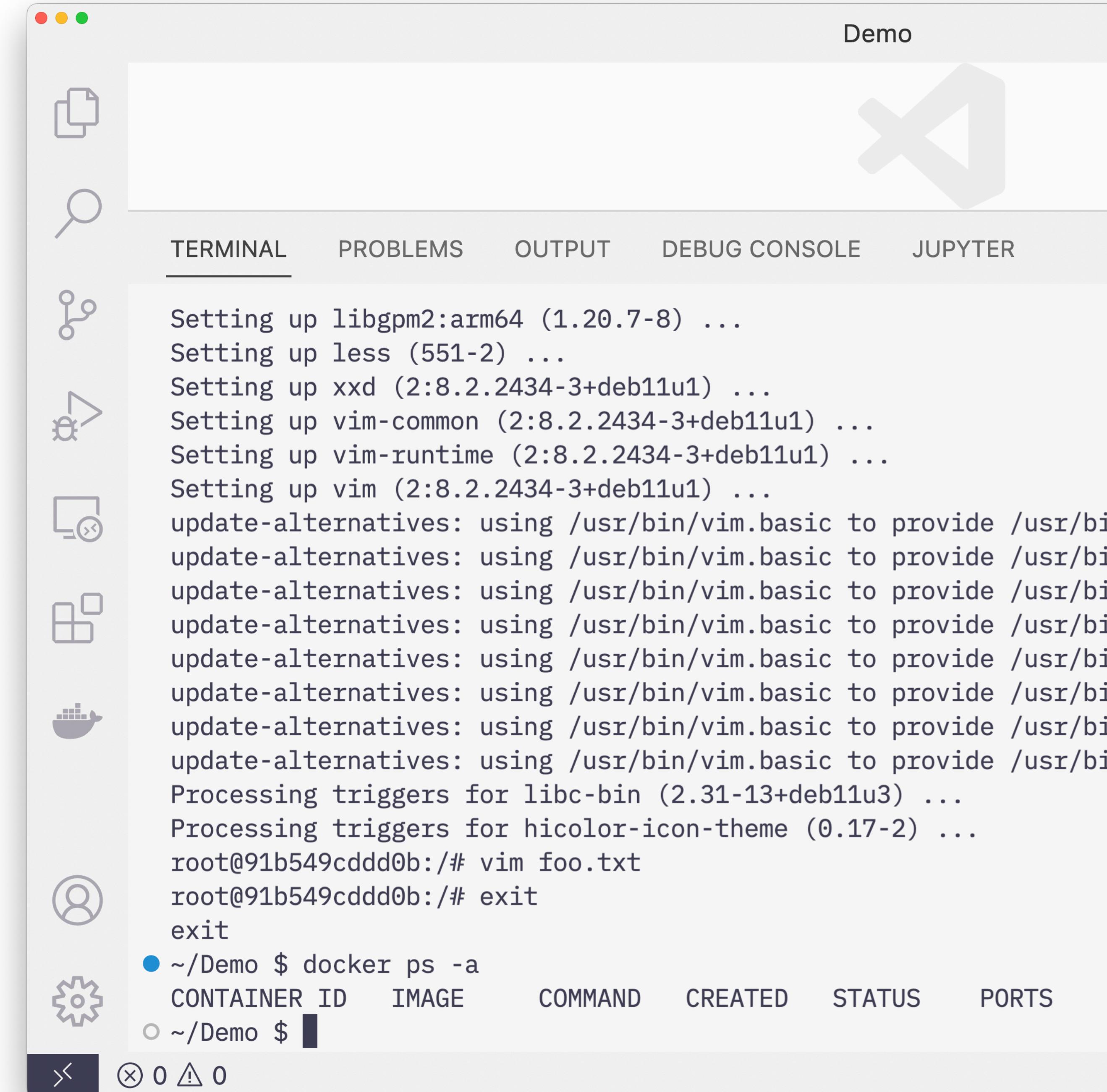
The screenshot shows the VS Code interface with the Docker extension installed. The sidebar on the left has icons for files, search, connections, terminals, and Docker. The main area is a terminal window titled "Demo" showing the output of the "docker images" command.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my_python	3.10	d3239b165beb	2 minutes ago	918MB
python	3.10	580b0402c5a8	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



The screenshot shows the VS Code interface with a terminal tab selected. The terminal window displays the following command-line session:

```
Setting up libgpm2:arm64 (1.20.7-8) ...
Setting up less (551-2) ...
Setting up xxd (2:8.2.2434-3+deb11u1) ...
Setting up vim-common (2:8.2.2434-3+deb11u1) ...
Setting up vim-runtime (2:8.2.2434-3+deb11u1) ...
Setting up vim (2:8.2.2434-3+deb11u1) ...
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@91b549cddd0b:/# vim foo.txt
root@91b549cddd0b:/# exit
exit
● ~/Demo $ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
○ ~/Demo $
```

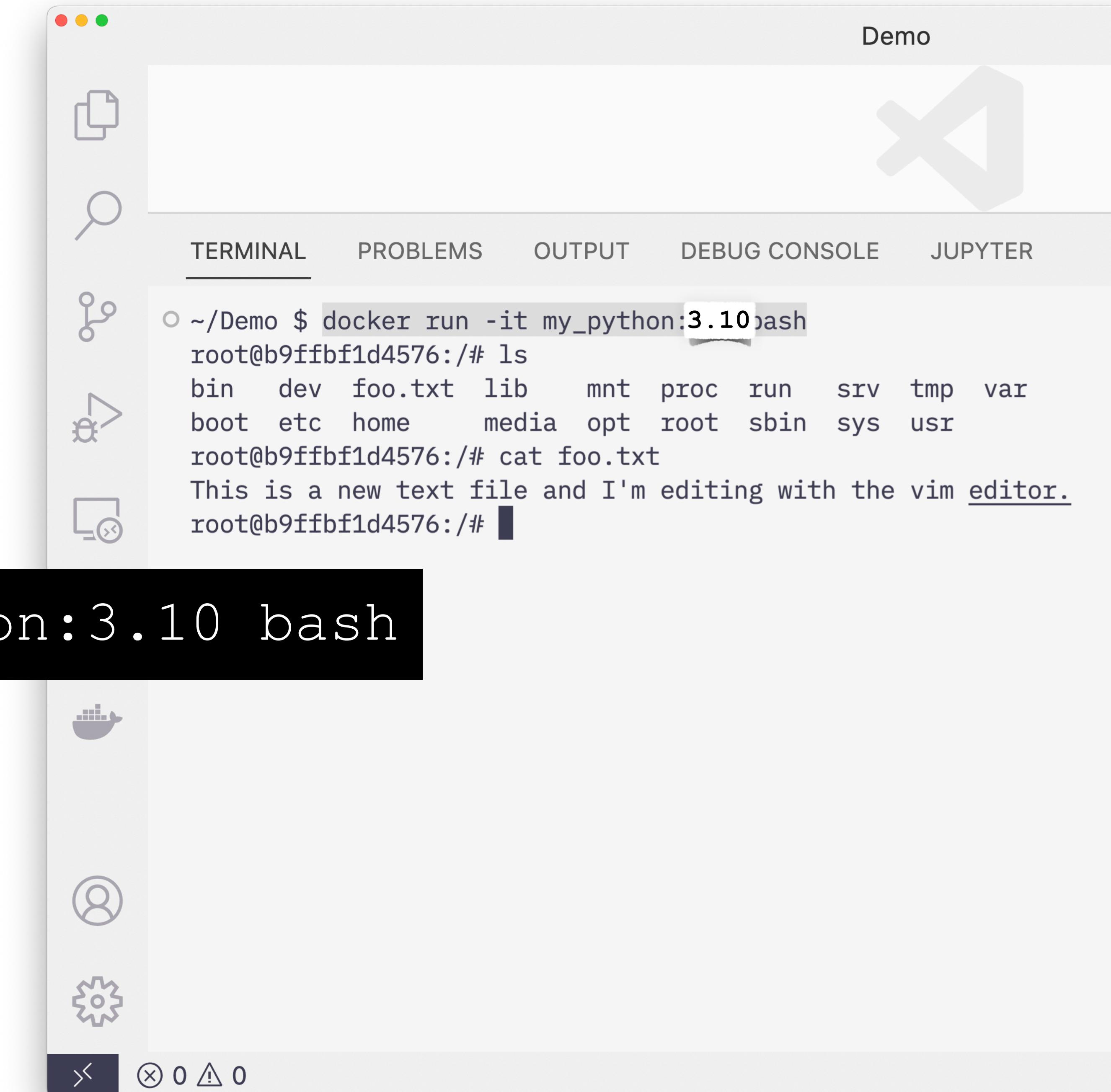
The terminal also shows the status bar at the bottom with icons for file operations, search, and terminal controls.

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
```



The screenshot shows the VS Code interface with the Docker extension installed. The title bar says "Demo". The left sidebar has icons for file, search, connection, and terminal. The main area has tabs for TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. The TERMINAL tab is active, showing a terminal session:

```
~/.Demo $ docker run -it my_python:3.10 bash
root@b9ffbf1d4576:/# ls
bin dev foo.txt lib mnt proc run srv tmp var
boot etc home media opt root sbin sys usr
root@b9ffbf1d4576:/# cat foo.txt
This is a new text file and I'm editing with the vim editor.
root@b9ffbf1d4576:/#
```

Below the terminal, there's a black bar with the command `$ docker run -it my_python:3.10 bash`. The bottom left of the interface shows the Docker icon in the sidebar and some status indicators at the bottom.

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



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + ▾ □ ⚡ ^ X

- ~/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
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

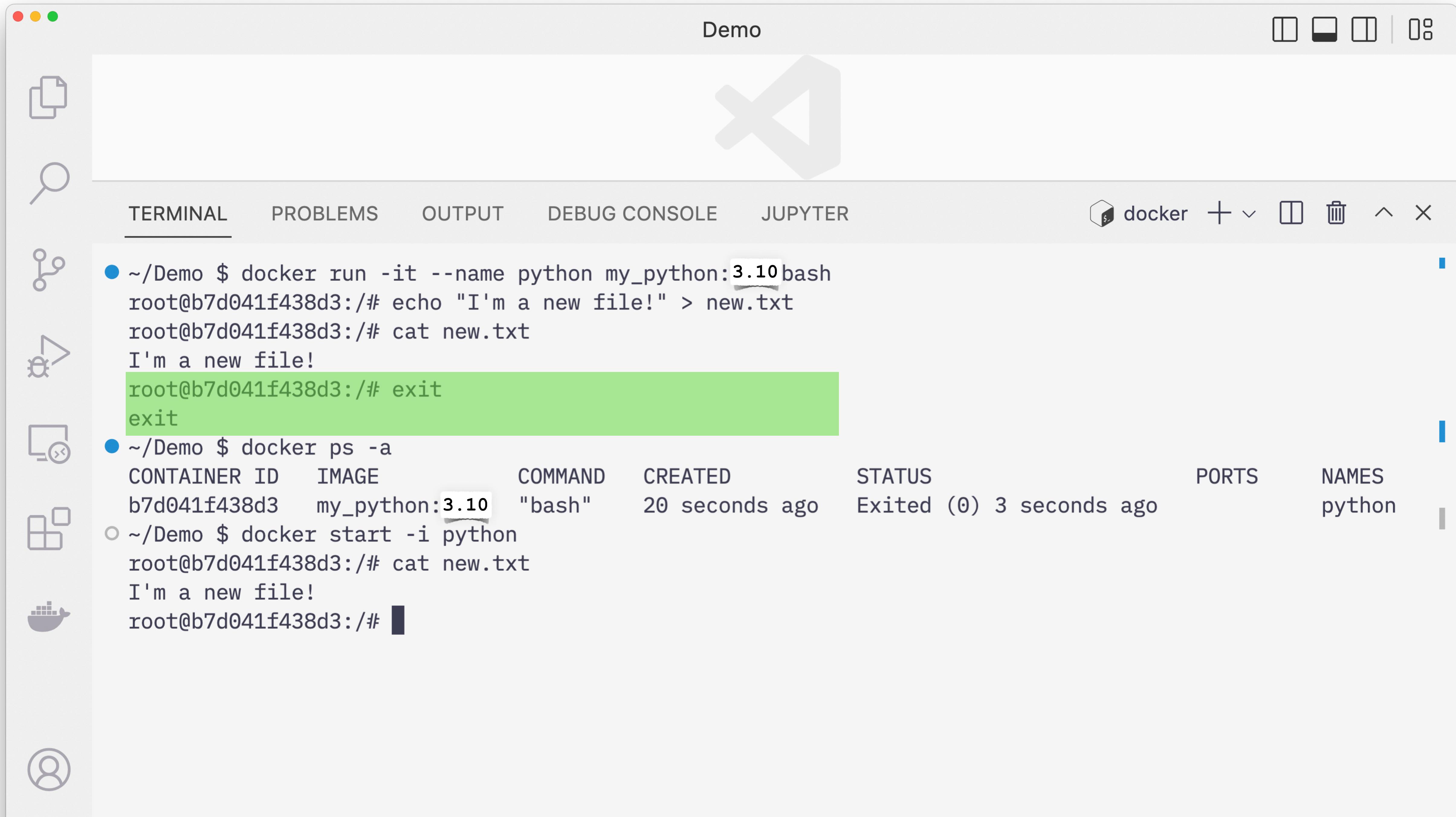
- ~/Demo \$ docker start -i python
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/# █

Demo

The screenshot shows the Visual Studio Code (VS Code) interface with the title bar "Demo". On the left is the sidebar with icons for files, search, connections, terminals, and Docker. The main area has tabs for TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. A Docker extension icon is visible in the top right. The TERMINAL tab is active, displaying a session with the following commands and output:

```
● ~/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
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
○ ~/Demo $ docker start -i python
root@b7d041f438d3:/# cat new.txt
I'm a new file!
root@b7d041f438d3:/#
```

A green highlight covers the first command and its output. The Docker sidebar shows a single container named "python" with the status "Exited (0) 3 seconds ago".



Demo



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + ▾ □ ⚡ ^ X

- ~/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
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



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + ▾ □ ⚡ ^ X

	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						

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

Demo



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + ▾ □ ⚡ ^ X

- ~/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
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

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

The screenshot shows the VS Code interface with a terminal tab active. The terminal window displays the following sequence of commands:

- ~/Demo \$ docker ps -a
- CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
- REPOSITORY TAG IMAGE ID CREATED SIZE
- my_httpd latest 23417707c159 About an hour ago 137MB
- my_python 3.10 d3239b165beb 12 hours ago 918MB
- python 3.10 580b0402c5a8 4 days ago 862MB
- httpd 2.4 b5543eff25e7 4 days ago 137MB
- httpd 2.4-alpine 74dd47829003 2 weeks ago 54.1MB
- ~/Demo \$ docker run -it --name python my_python:3.9 bash
- root@10aa1f5fc064:/# exit
- exit
- ~/Demo \$ docker ps -a
- CONTAINER ID IMAGE COMMAND CREATED STATUS
- 10aa1f5fc064 my_python:3.10 "bash" 6 seconds ago Exited (0) 1 second ago
- ~/Demo \$ docker commit python my_python:3.10
- sha256:52b1fe98191f94da10cce60322bac2e5ad4b32f5320adbc57ec0821beafdf1e04
- ~/Demo \$
- ~/Demo \$
- ~/Demo \$
- ~/Demo \$

The terminal window has a dark theme with light-colored text. The sidebar on the left contains icons for file, search, file tree, and Docker. The title bar says "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



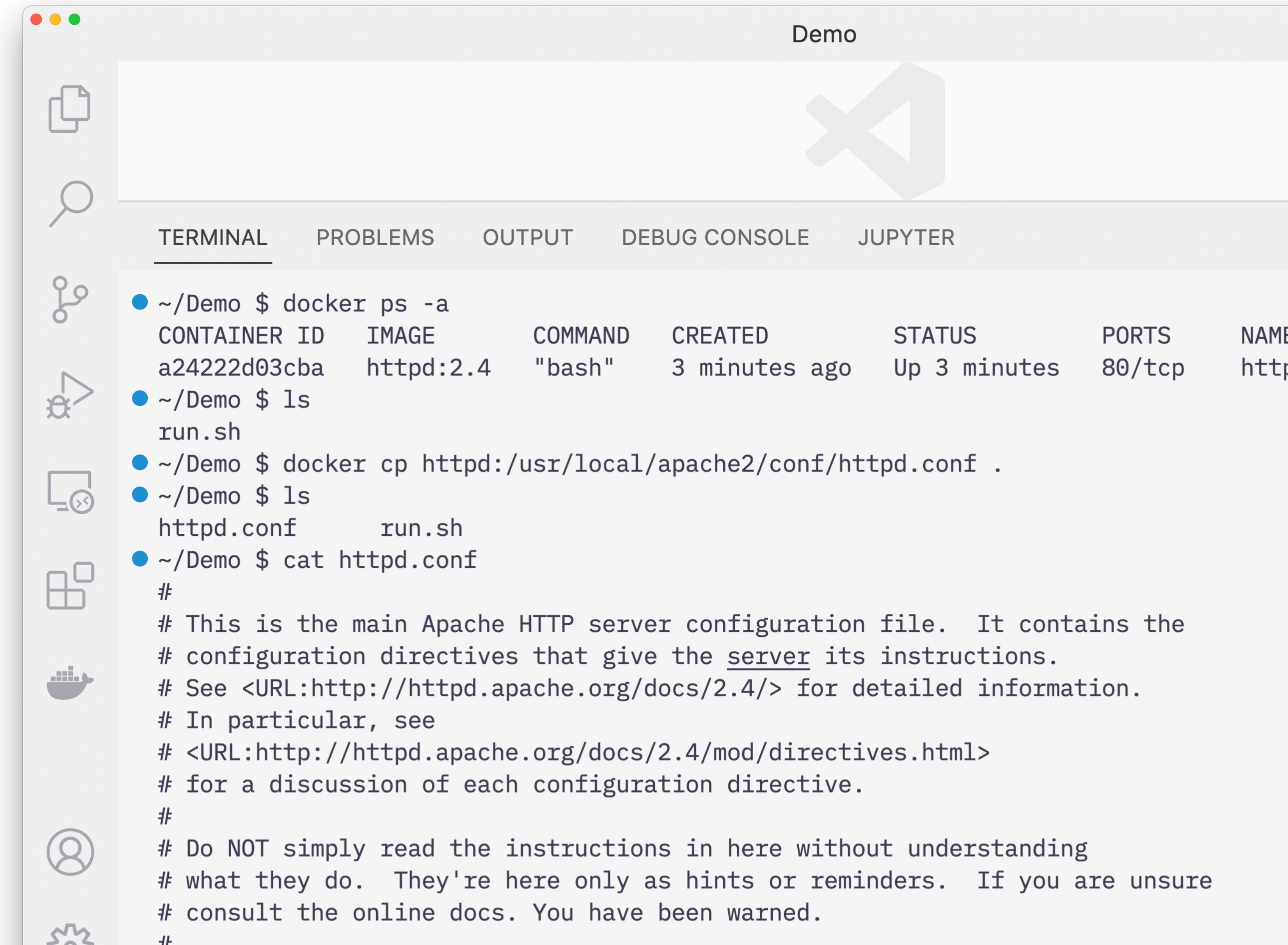
The screenshot shows a macOS Dock with several icons: Finder, Spotlight, Home, and Terminal. The Terminal icon is highlighted with a red circle. To the right of the dock is a screenshot of a terminal window titled "Demo". The terminal window has tabs for TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. The TERMINAL tab is selected. The terminal output is as follows:

```
● ~/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”



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal tab is selected at the top. The output in the terminal is as follows:

```
~/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              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:> for detailed information.
# In particular, see
# <URL:>
# 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

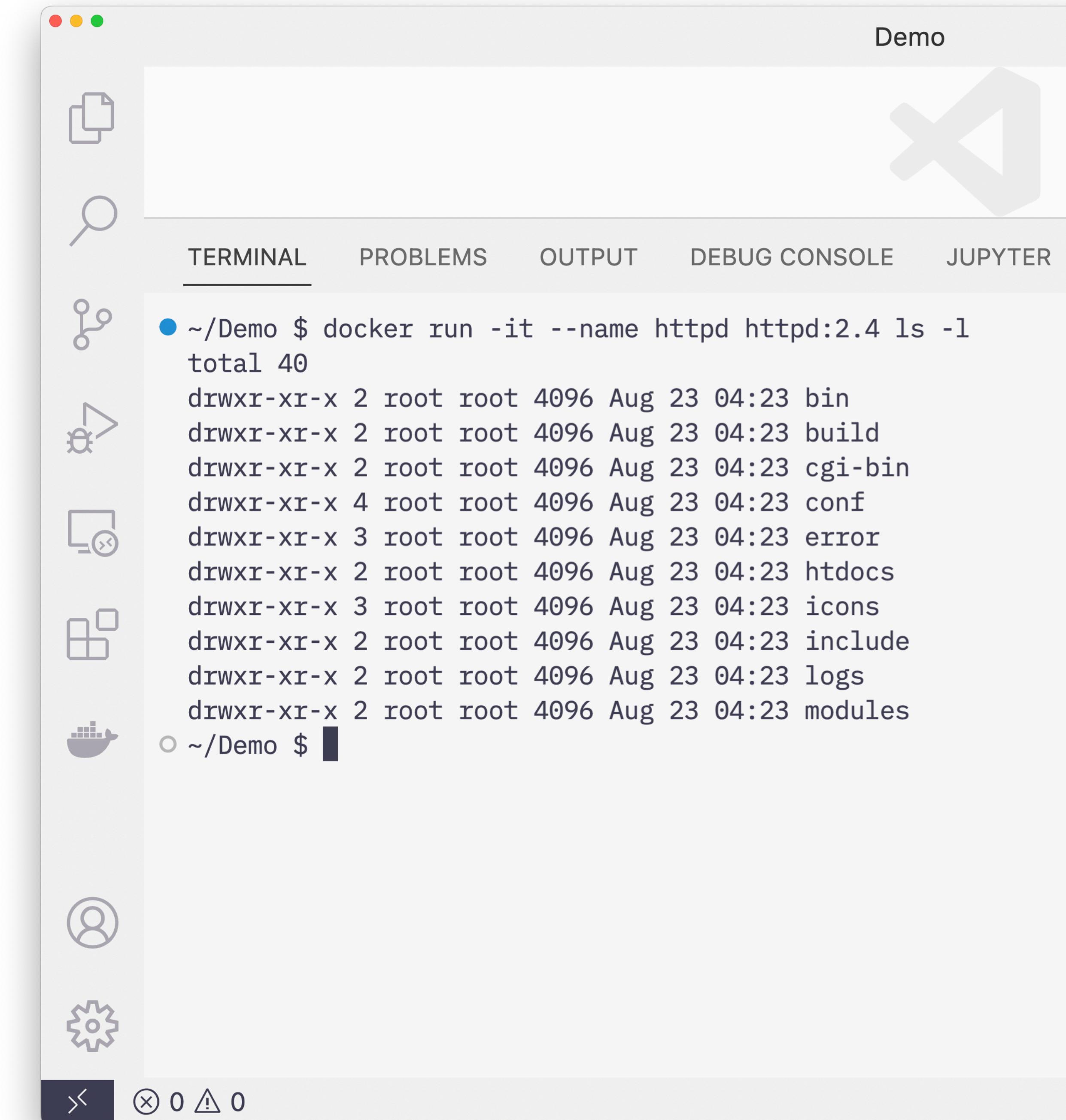
- This works the other way too. You can copy files from your host into a running or stopped container. Just reverse the order of the arguments

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

Docker

Other Container Commands

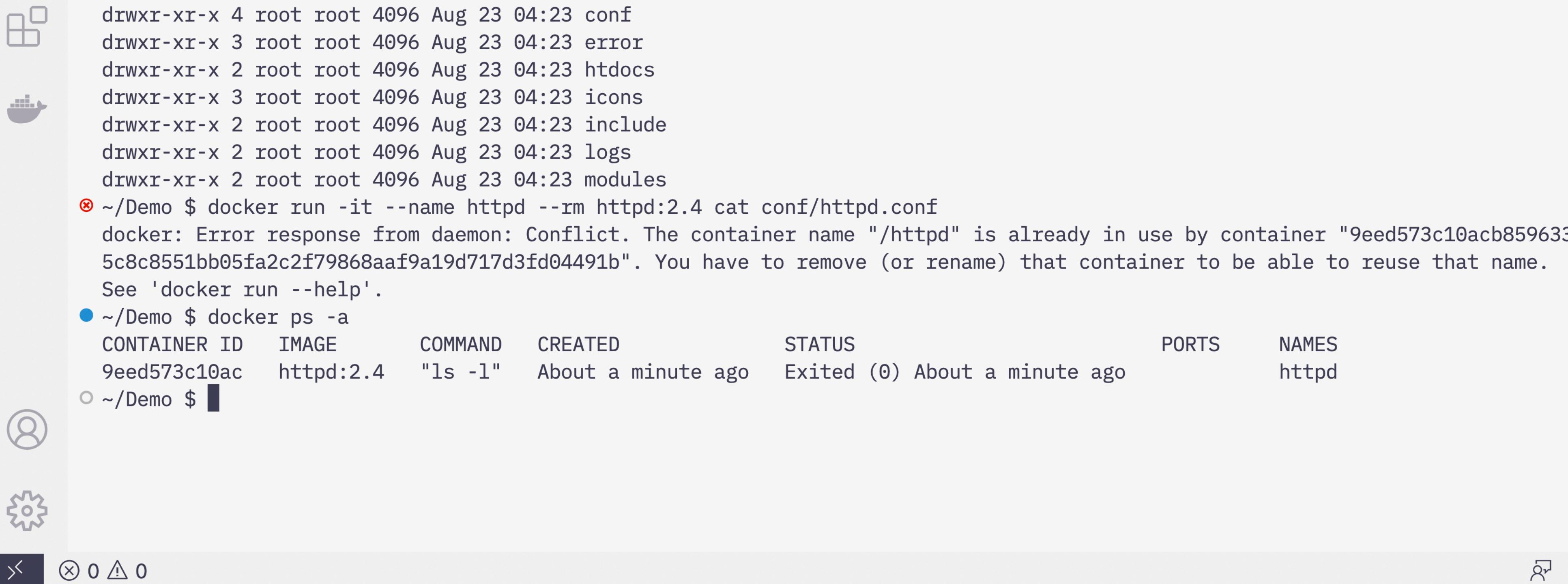
- You don't have to just run a new bash shell inside of a container.
- We can just run the `ls` command
- Or just a `cat` command.



The screenshot shows a terminal window within a code editor interface. The title bar says "Demo". The terminal tab is selected. The output of the command `docker run -it --name httpd httpd:2.4 ls -l` is displayed, showing the directory structure of the Apache web server container. The terminal prompt ends with a vertical bar and a cursor.

```
~/Demo $ docker run -it --name httpd 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



The screenshot shows a terminal window with the following content:

```
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 -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 "9eed573c10acb859633
5c8c8551bb05fa2c2f79868aaf9a19d717d3fd04491b". 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
9eed573c10ac httpd:2.4 "ls -l" About a minute ago Exited (0) About a minute ago httpd

○ ~/Demo $ █
```

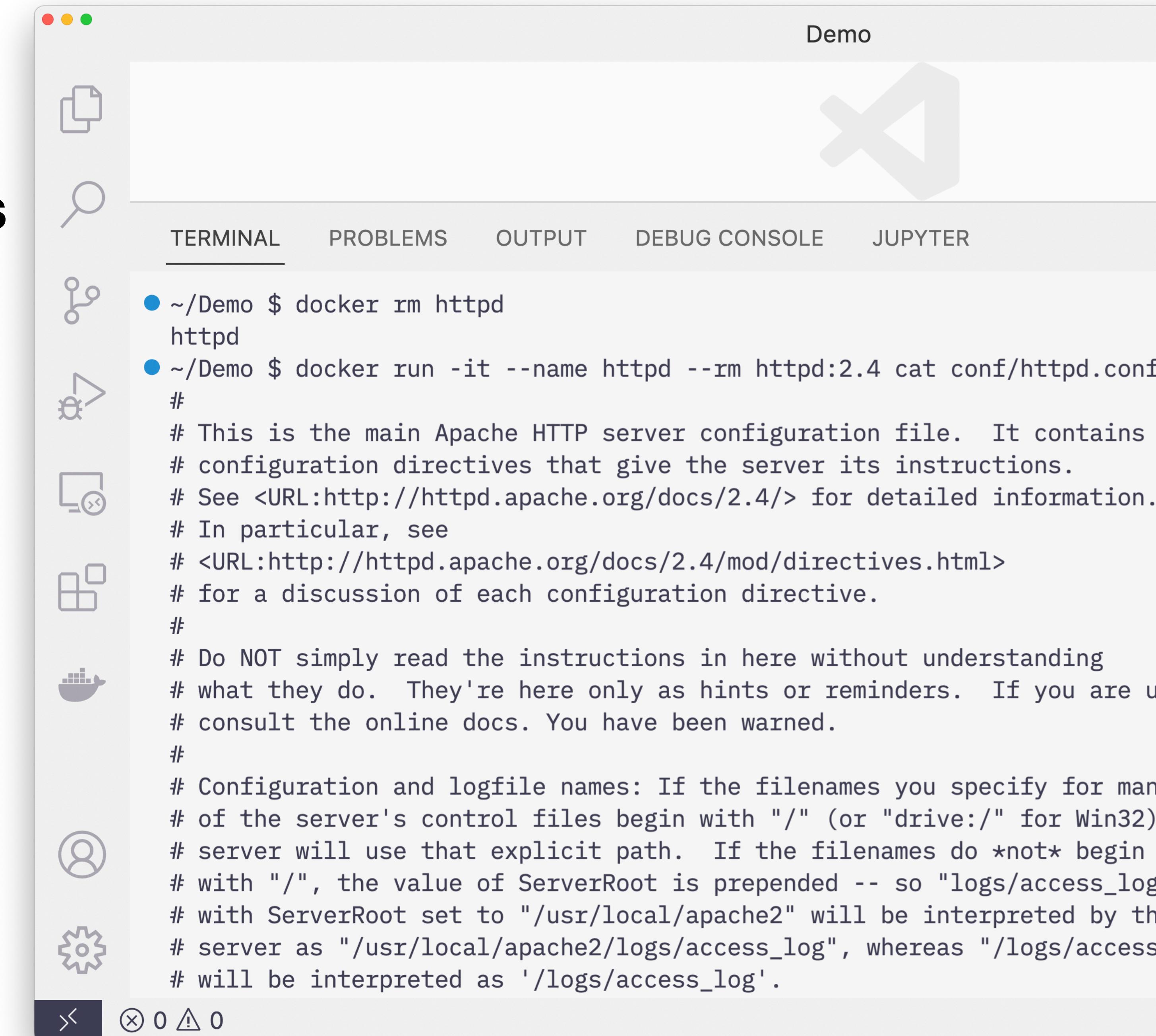
The terminal shows the user's directory (~/.Demo), attempting to run a new container named 'httpd' based on the 'httpd:2.4' image. The command fails because a container with the name 'httpd' already exists. The user then lists all containers ('docker ps -a') and sees the existing container 'httpd' listed.

- 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

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



The screenshot shows a terminal window in VS Code with the title "Demo". The terminal tab is selected at the top. On the left, there is a vertical sidebar with icons for file operations, search, connection, terminal, workspace, Docker, users, and settings. The terminal content displays the following text:

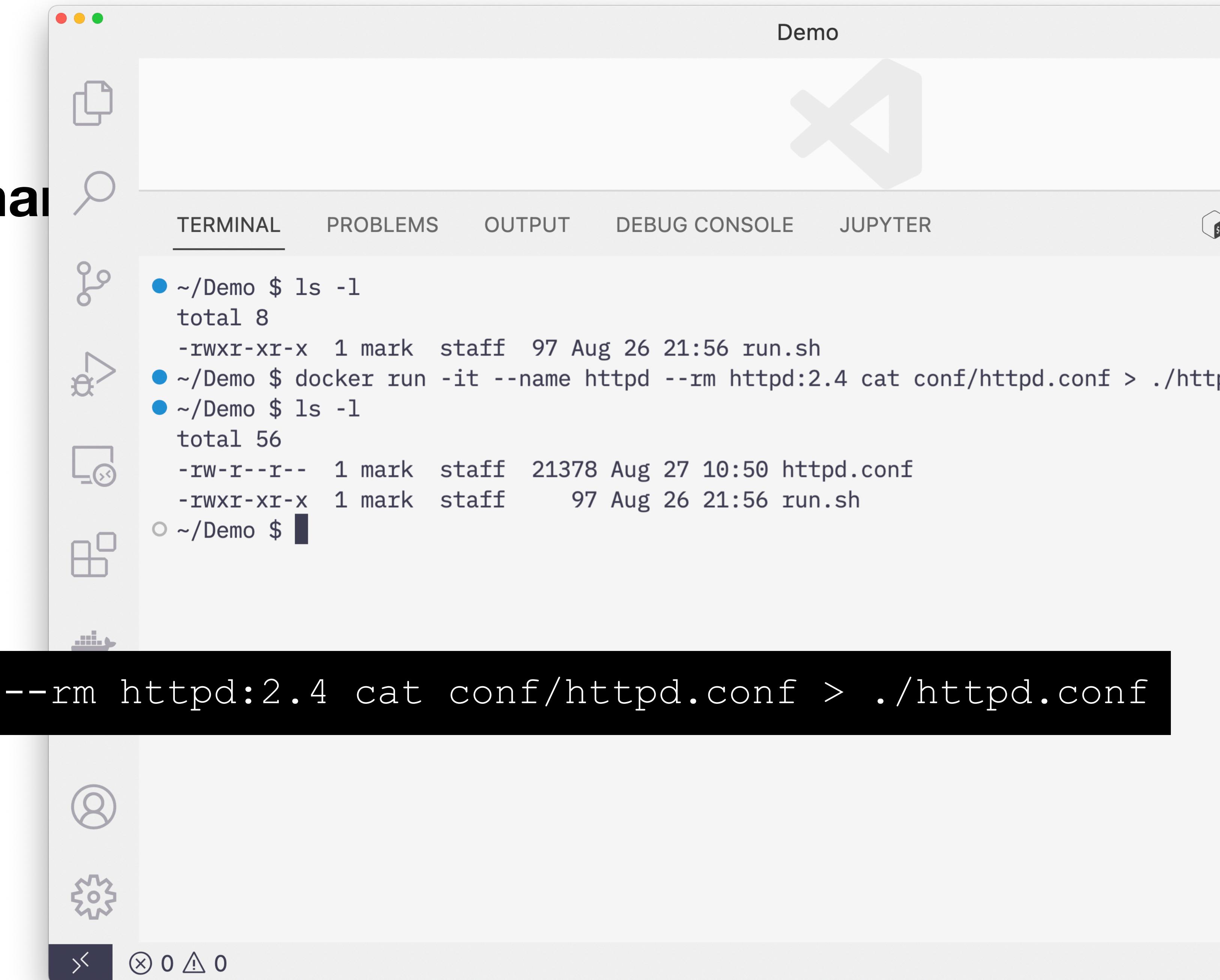
```
● ~/Demo $ docker rm httpd
httpd
● ~/Demo $ docker run -it --name httpd --rm httpd:2.4 cat conf/httpd.conf
#
# This is the main Apache HTTP server configuration file. It contains
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed information.
# In particular, see
# <URL: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 u
#
# Configuration and logfile names: If the filenames you specify for man
# of the server's control files begin with "/" (or "drive:/" for Win32)
# 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/apache2" will be interpreted by th
# server as "/usr/local/apache2/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
```



The screenshot shows the VS Code interface with the title bar "Demo". The left sidebar has icons for File, Search, Connect, Share, and Docker. The top navigation bar includes TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. The TERMINAL tab is active, displaying the following command history:

```
~/Demo $ ls -l
total 8
-rwxr-xr-x 1 mark staff 97 Aug 26 21:56 run.sh
~/Demo $ docker run -it --name httpd --rm httpd:2.4 cat conf/httpd.conf > ./httpd.conf
~/Demo $ ls -l
total 56
-rw-r--r-- 1 mark staff 21378 Aug 27 10:50 httpd.conf
-rwxr-xr-x 1 mark staff 97 Aug 26 21:56 run.sh
~/Demo $
```

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.

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

- EXPLORER View:** Shows a folder named "DEMO" containing ".bash_history", "httpd.conf", and "run.sh".
- Editor View:** A file named "run.sh" is open, displaying the following script content:

```
$ run.sh
1 #!/bin/bash -ex
2 docker run \
3   -it \
4   --rm \
5   --name python \
6   --volume $PWD:/root \
7   python:3.10 \
8   bash
9
```

- Terminal View:** Shows two sessions:
 - Session 1 (blue dot):

```
~/Demo $ ls -l
total 56
-rw-r--r-- 1 mark staff 21378 Aug 27 10:50 httpd.conf
-rwxr-xr-x 1 mark staff 122 Aug 27 11:45 run.sh
```
 - Session 2 (orange dot):

```
~/Demo $ ./run.sh
+ docker run -it --rm --name python --volume /Users/mark/Demo:/root python:3.10 bash
root@27b0027188f1:/# cd /root
root@27b0027188f1:~/# ls -l
total 28
-rw-r--r-- 1 root root 21378 Aug 27 17:50 httpd.conf
-rwxr-xr-x 1 root root 122 Aug 27 18:45 run.sh
root@27b0027188f1:~/#
```
- Status Bar:** Shows "Ln 6, Col 24 (19 selected) Spaces: 4 UTF-8 LF S".

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 ... \$ run.sh hello.py X

> OPEN EDITORS

DEMO

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) + ".")
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

bash + ×

~ /Demo \$./run.sh
+ docker run -it --rm --name python --volume /Users/mark/Demo/work:/root/work python:3.10 bash
root@6e78993d35d3:/# cd /root/
root@6e78993d35d3:~/# ls -l
total 0
drwxr-xr-x 3 root root 96 Aug 27 18:54 work
root@6e78993d35d3:~/# cd work/
root@6e78993d35d3:~/work# ls -l
total 4
-rw-r--r-- 1 root root 173 Aug 27 19:03 hello.py
root@6e78993d35d3:~/work# python hello.py
Hello! It is currently Saturday, August 27 at 07:04 PM.
root@6e78993d35d3:~/work#

Demo

Creating Images

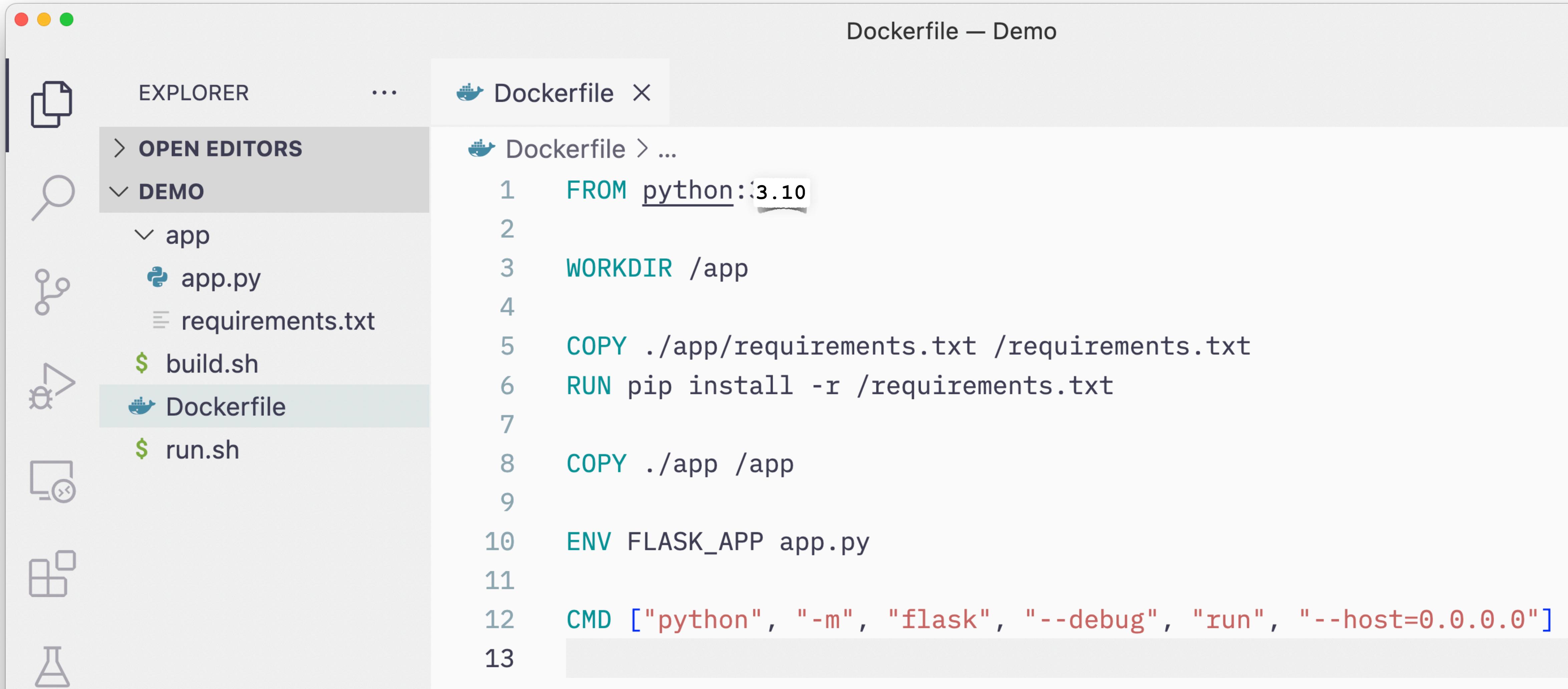
The Dockerfile

- 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

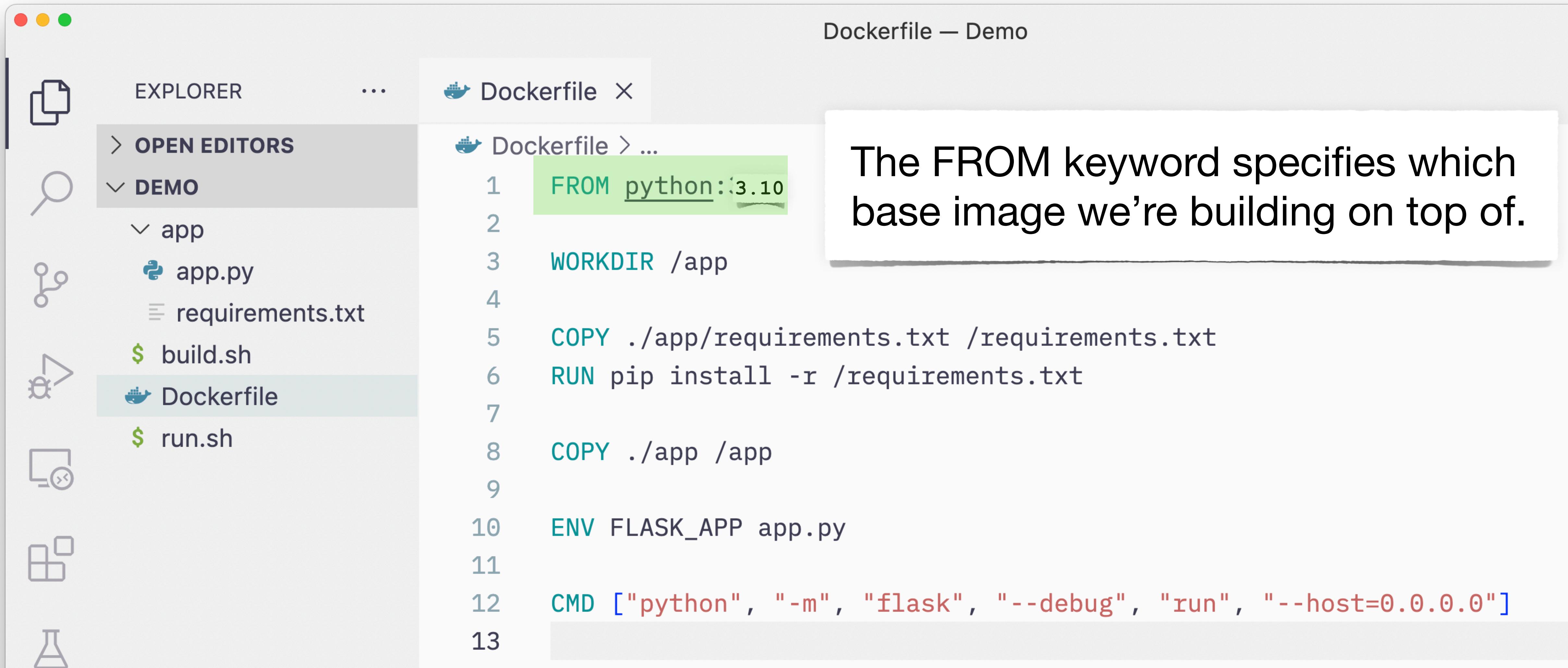


The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer, displaying a file tree with a project named 'DEMO'. Inside 'DEMO' are subfolders 'app' containing 'app.py' and 'requirements.txt', and a script 'build.sh'. The 'Dockerfile' is selected in the Explorer and is also open in the main editor area. The main editor window title is 'Dockerfile — Demo'. The code in the editor is:

```
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



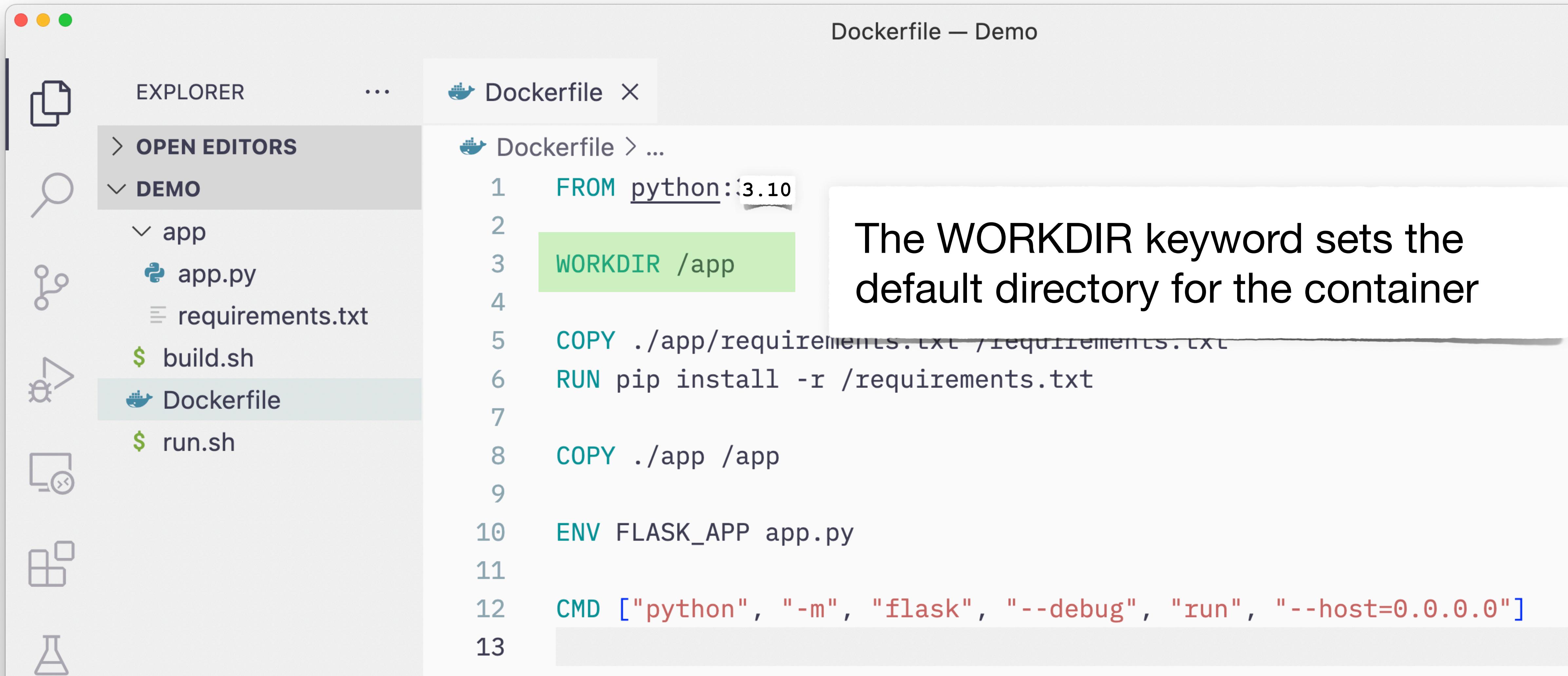
The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer, displaying a project structure for a 'DEMO' folder. Inside 'DEMO' are 'app', 'app.py', 'requirements.txt', 'build.sh', 'Dockerfile' (which is selected), and 'run.sh'. The main area is the 'Dockerfile' editor, titled 'Dockerfile — Demo'. The code is:

```
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 the 'FROM' line with the text: 'The FROM keyword specifies which base image we're building on top of.'

Creating Images

The Dockerfile



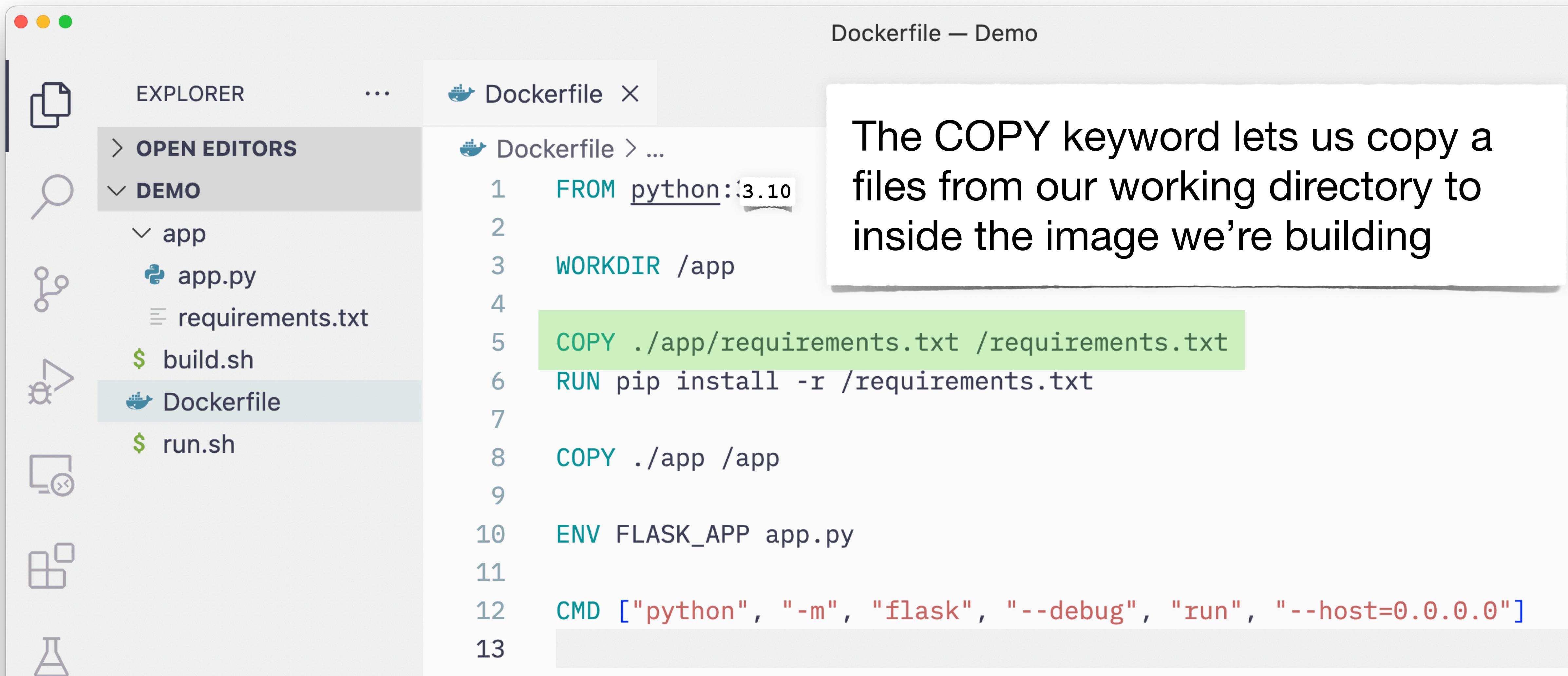
The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer, displaying a project structure with a folder named 'DEMO'. Inside 'DEMO' are subfolders 'app' containing 'app.py' and 'requirements.txt', and files 'build.sh', 'Dockerfile', and 'run.sh'. The 'Dockerfile' file is currently selected and open in the main editor area. The code in the editor is:

```
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"]
```

A callout box points to the 'WORKDIR' command, which is highlighted with a green background. The text inside the callout box reads: 'The WORKDIR keyword sets the default directory for the container'.

Creating Images

The Dockerfile



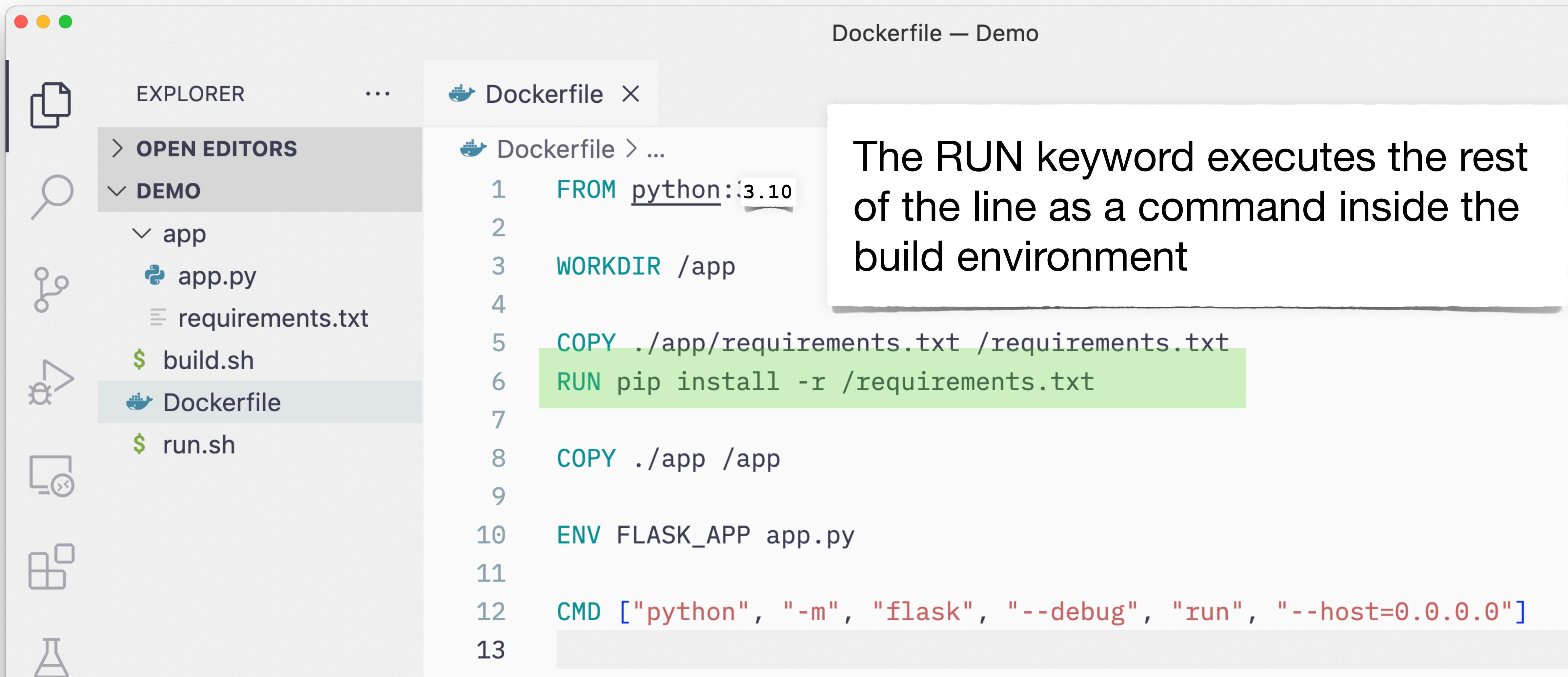
Dockerfile — Demo

```
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"]
```

The COPY keyword lets us copy files from our working directory to inside the image we're building

Creating Images

The Dockerfile



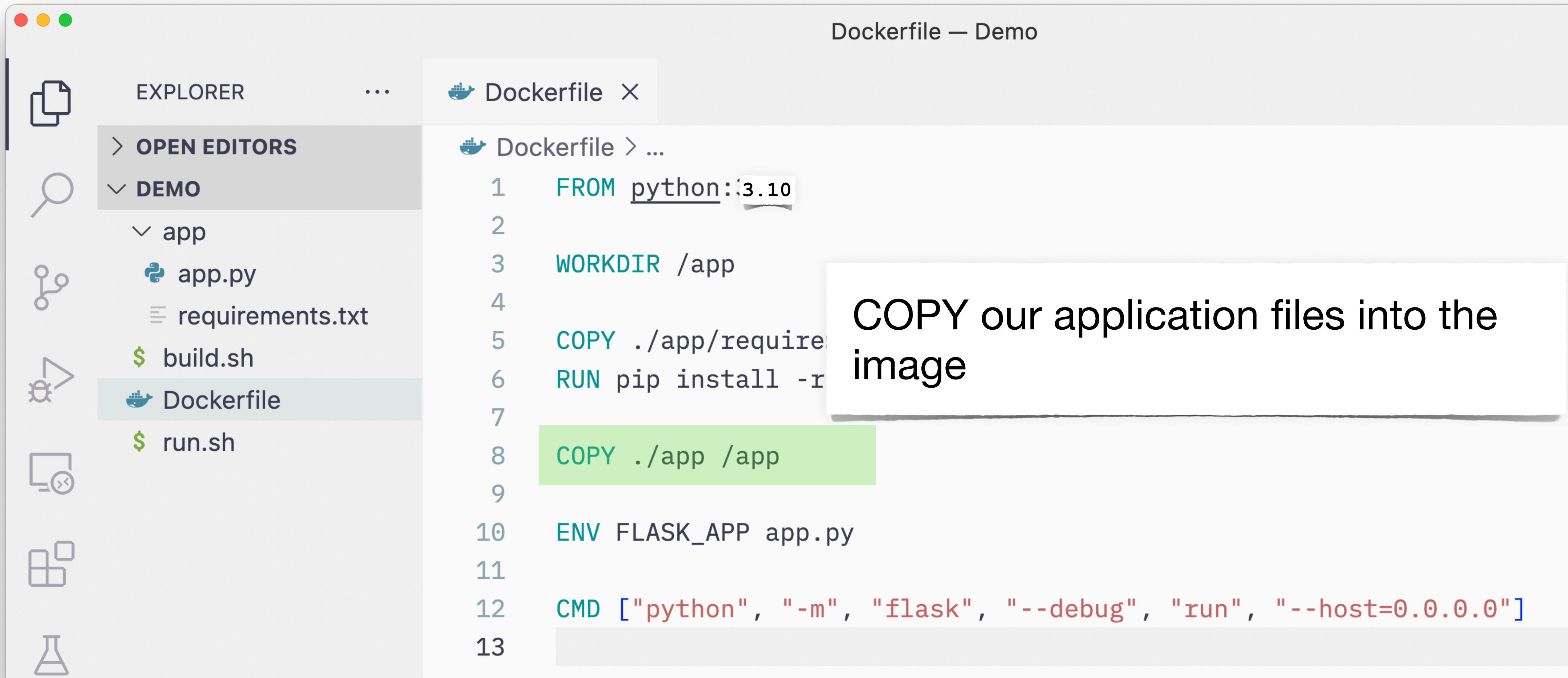
The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer view, which lists a project named "DEMO". Inside "DEMO", there is an "app" folder containing "app.py" and "requirements.txt", a "build.sh" file, and a "Dockerfile". The "Dockerfile" is currently selected and open in the main editor area. The code in the editor is:

```
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 the "RUN" command in line 6, highlighting it with a green background. The text inside the callout box reads: "The RUN keyword executes the rest of the line as a command inside the build environment".

Creating Images

The Dockerfile



The screenshot shows the Visual Studio Code interface with the following details:

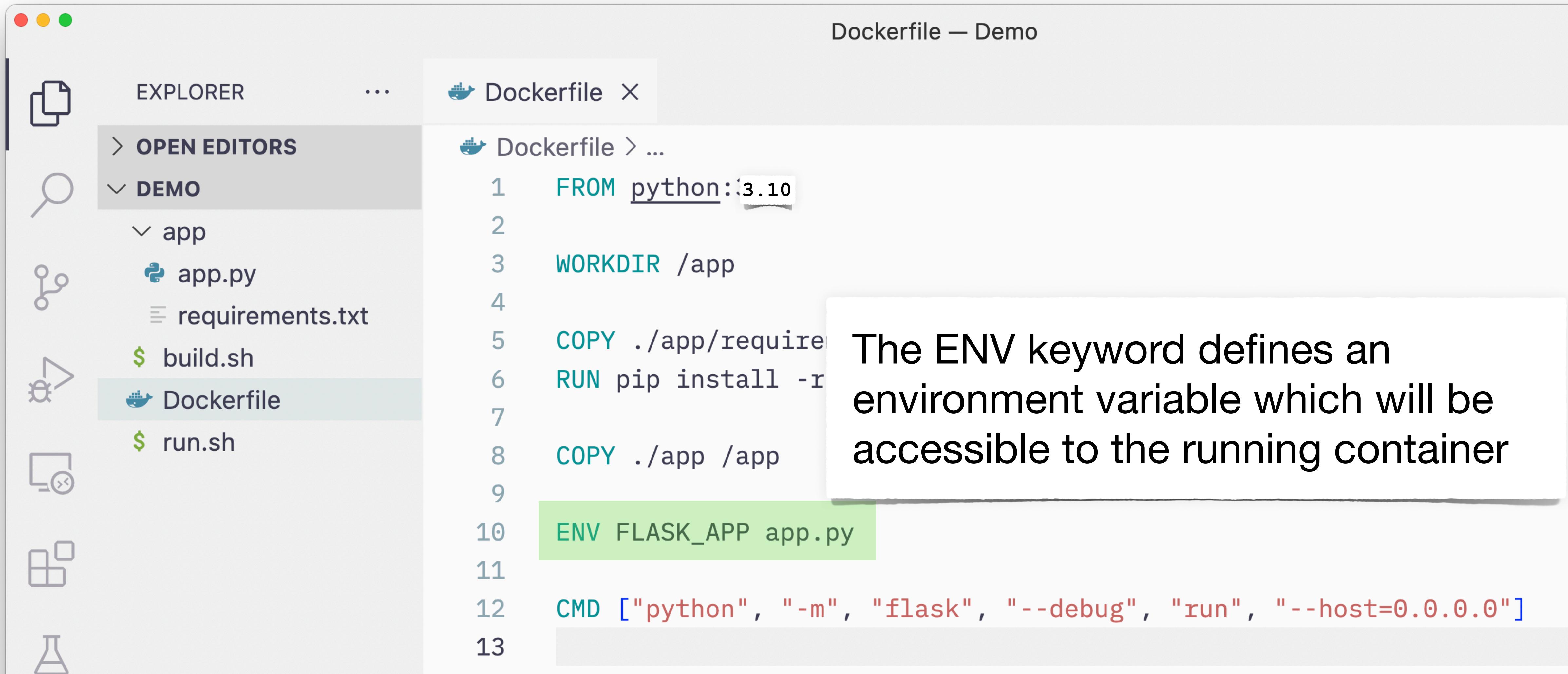
- Explorer View:** Shows a project structure with a folder named "DEMO". Inside "DEMO" are subfolders "app", "requirements.txt", "build.sh", and files "Dockerfile" and "run.sh".
- Dockerfile Editor:** The file "Dockerfile" is open in the main editor area. The code is as follows:

```
FROM python:3.10
WORKDIR /app
COPY ./app/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"]
```

A callout box highlights the line `COPY ./app /app` with the text: "COPY our application files into the image".

Creating Images

The Dockerfile



The screenshot shows a code editor interface with a sidebar and a main editor area. The sidebar on the left has icons for file operations, search, and navigation. The main area shows a file tree under 'EXPLORER' with a project named 'DEMO'. Inside 'DEMO', there is a folder 'app' containing 'app.py' and 'requirements.txt', a file 'build.sh', and a Dockerfile. The Dockerfile is open in the editor.

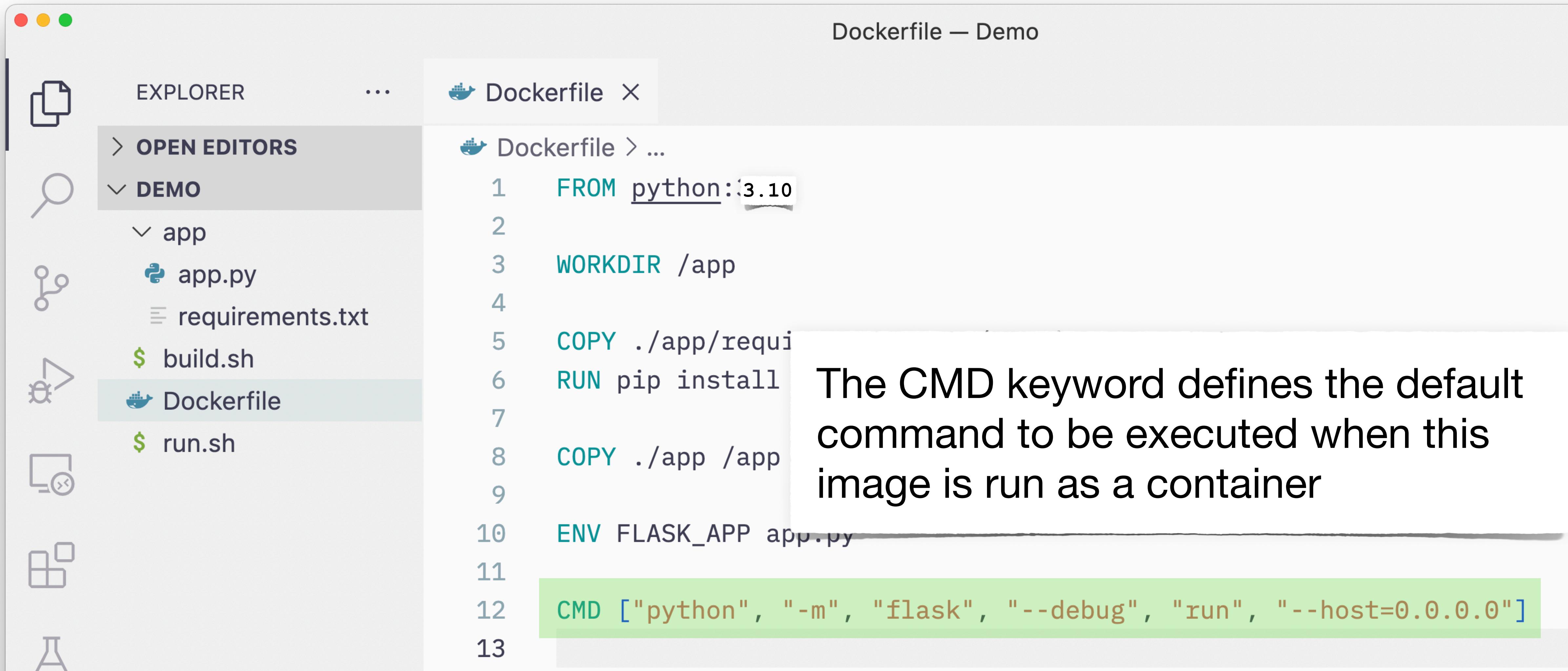
Dockerfile — Demo

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/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



The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer, displaying a project structure for a 'DEMO' folder. Inside 'DEMO' are subfolders 'app', files 'app.py' and 'requirements.txt', and scripts 'build.sh', 'Dockerfile', and 'run.sh'. The 'Dockerfile' file is currently selected in the Explorer and is also open in the main code editor area. The code editor shows a Dockerfile with the following content:

```
FROM python:3.10
WORKDIR /app
COPY ./app/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"]
```

A callout box points to the 'CMD' command at the bottom of the Dockerfile, highlighting it with a green background. The text inside the callout box reads: '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]
```

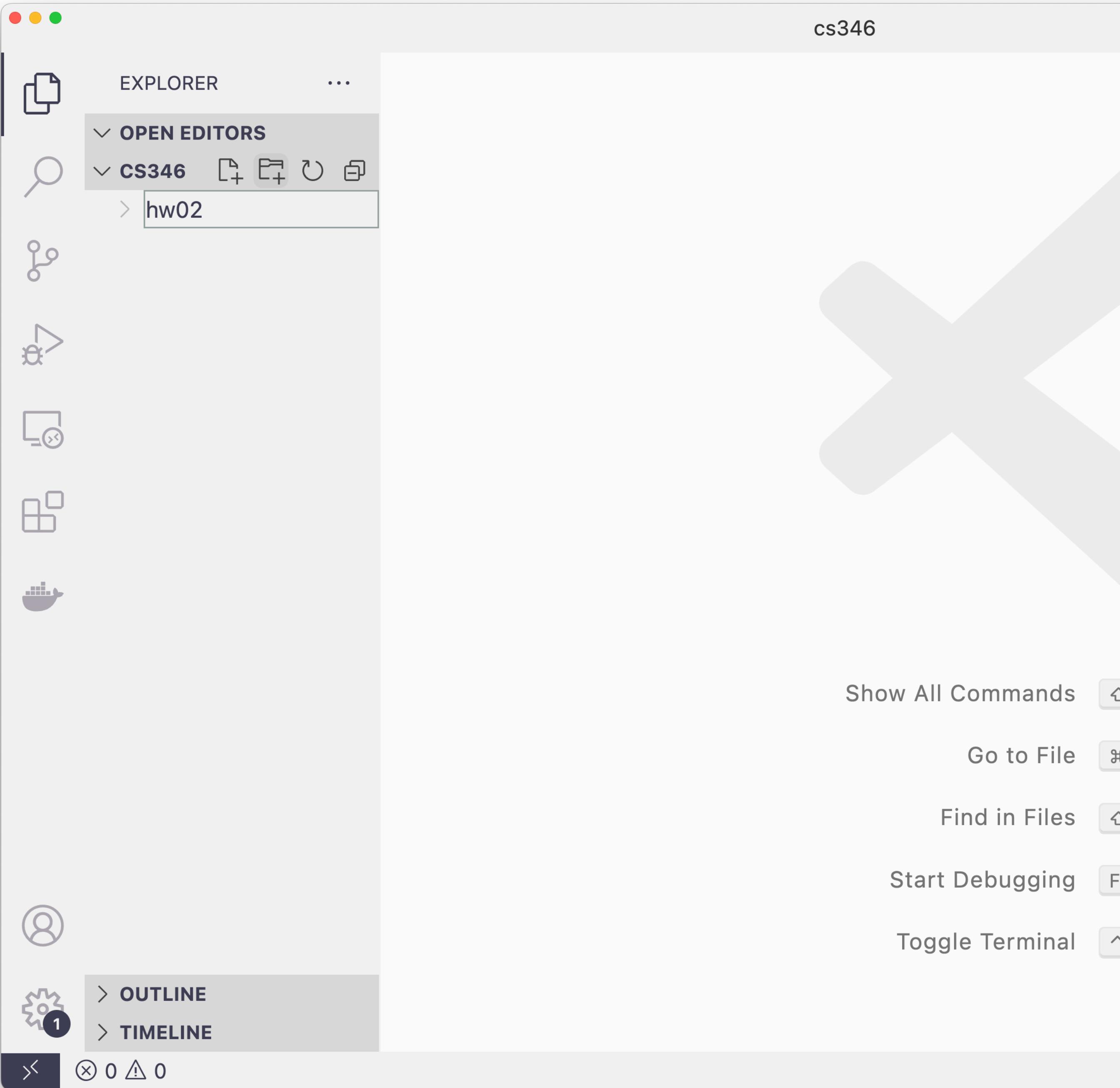
```
docker build --tag my_app:latest .
```

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

Creating Images

Getting Started Basics

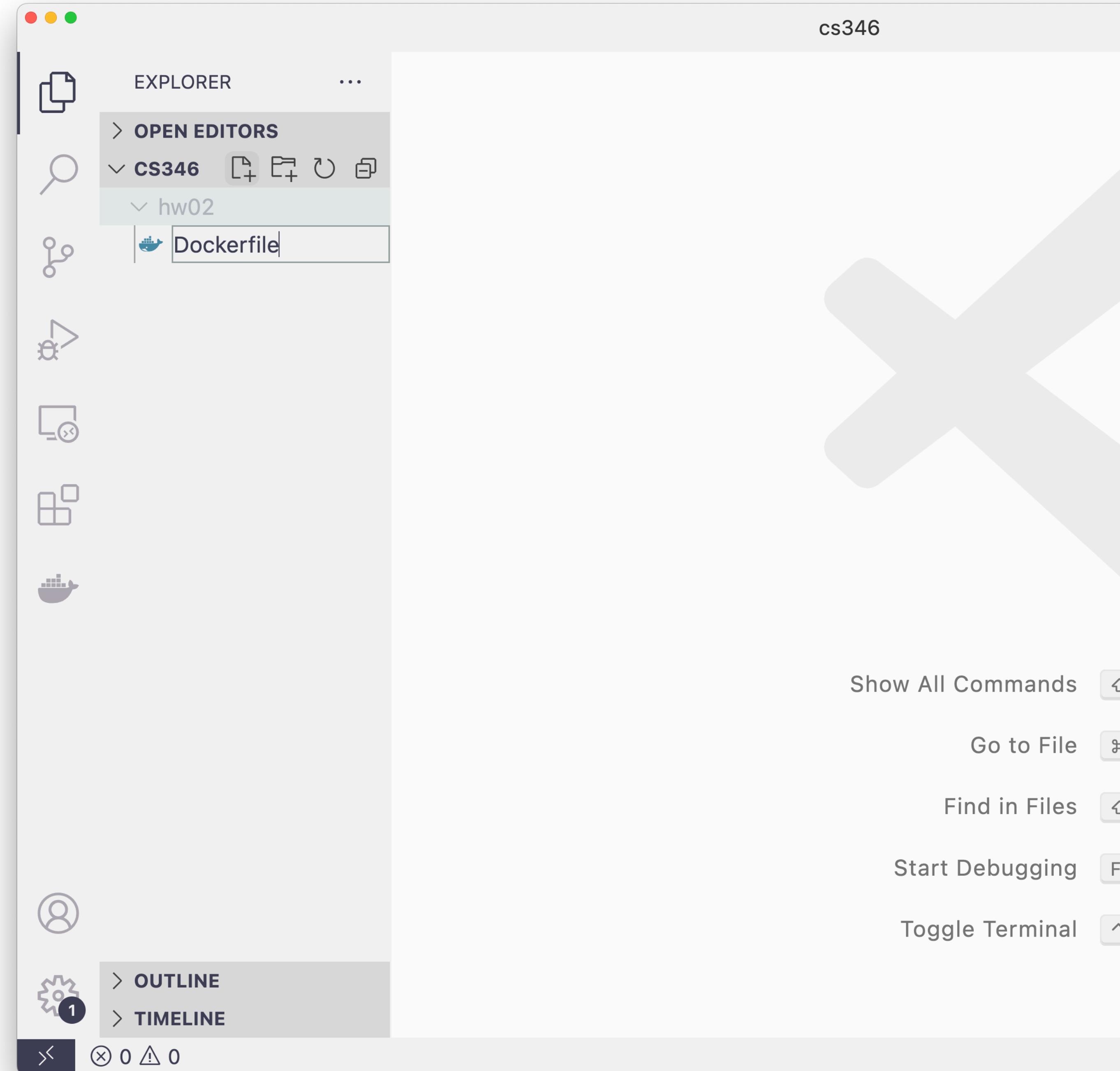
- When working on any project, such as an application or homework assignment, the first step is often to create a new directory to hold all the stuff relating to the project.
- So to start with, figure out where on your laptop you want to keep all your work for this class, and make a new folder in there.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



Creating Images

Getting Started Basics

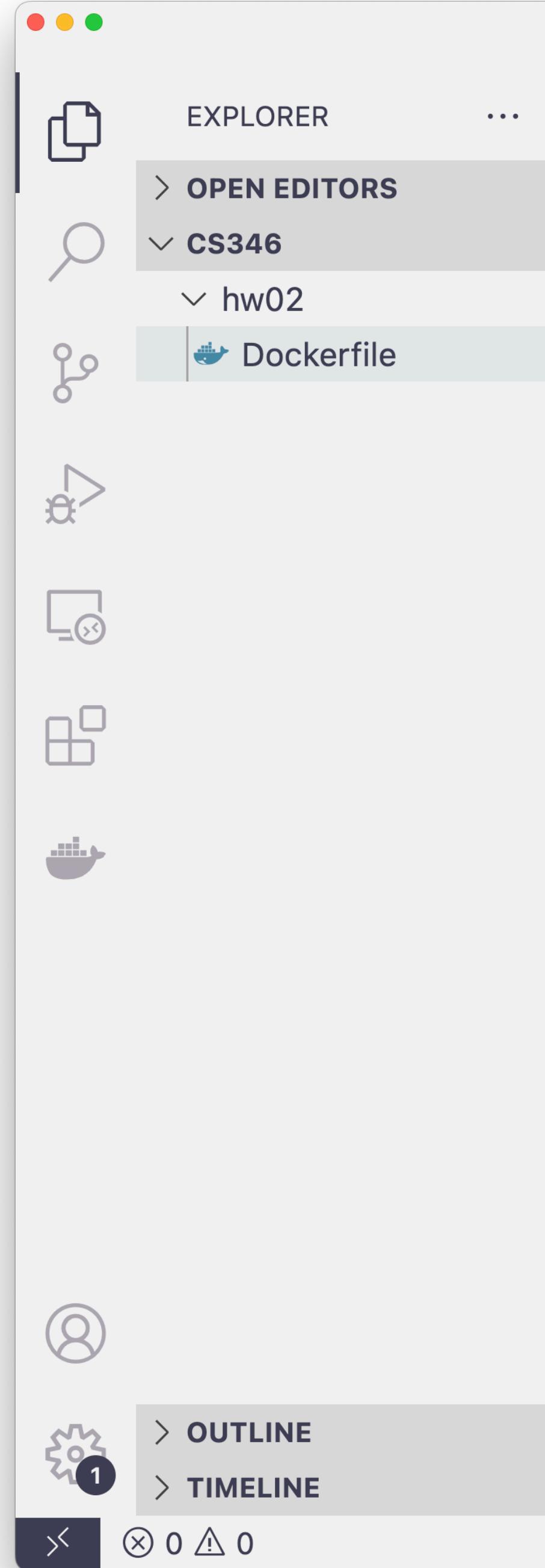
- Next we need to create a new empty text file inside our project folder, and name it **Dockerfile**



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.



The screenshot shows a code editor interface with a sidebar labeled "EXPLORER". The sidebar lists a project structure: "CS346" expanded to show "hw02" which contains a "Dockerfile". The main editor area is titled "Dockerfile" and shows the path "hw02 > Dockerfile > ...". The Dockerfile content is as follows:

```
1 FROM httpd:2.4
2 
```

The first line "FROM httpd:2.4" is highlighted in blue, indicating it is selected or being edited. The status bar at the bottom right of the editor shows "Ln 2, 0".

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 .
```

The screenshot shows the Visual Studio Code (VS Code) interface. In the top right corner, the title bar reads "Dockerfile – cs346". The left sidebar is the Explorer view, showing a tree structure with "CS346" expanded, "hw02" expanded, and "Dockerfile" selected. The main editor area is titled "Dockerfile" and contains the following code:

```
1 FROM httpd:2.4
2
```

Below the editor are tabs for TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. The TERMINAL tab is active, showing two entries:

- ~/cs346 \$ cd hw02/
- ~/cs346/hw02 \$ docker build -t hw02:latest .

At the bottom of the interface, there is a status bar with icons for outline, timeline, and error counts (0 errors, 0 warnings).

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 VS Code interface with the Dockerfile editor open. The Explorer sidebar on the left shows a project structure with 'CS346' expanded, 'hw02' selected, and 'Dockerfile' highlighted. The Dockerfile content is as follows:

```
1 FROM httpd:2.4
2
```

The Terminal tab at the bottom is active, displaying the build logs:

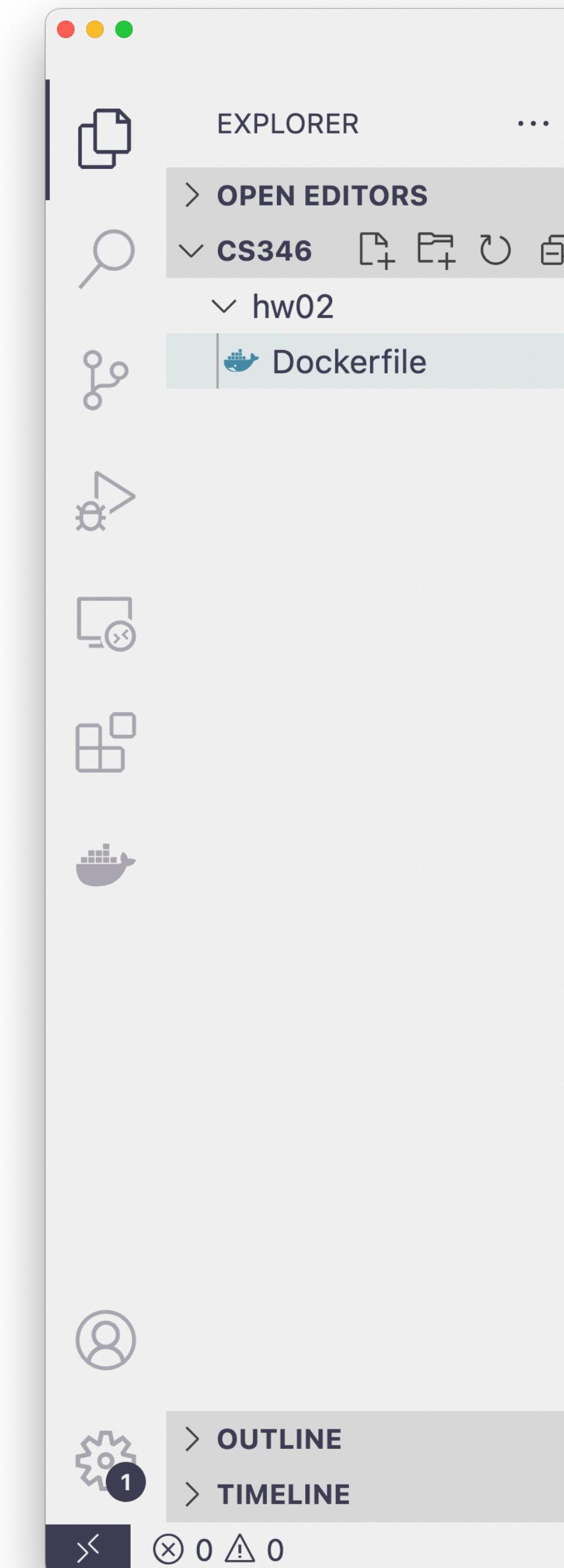
```
=> => sha256:ece99f7f2dd23c291f8aecd79db22e0bc5f64dc58417
=> => sha256:b5543eff25e7fecee76d2c94e7ec9b7172583a39c3a0
=> => sha256:5b142346550416c75ea412d21741de5eaf3e76857aff
=> => sha256:ceb4a75630f5375a81b5da051775265a449cdbd3ec73
=> => sha256:f2056621057792c8761cdf5ee802edad84c28a7e5462
=> => sha256:ee03e037f8b64b93846e433c846d6a019856878ad5c7
=> => sha256:8220bbf1aee7c576d5e9d1ae415171dfe7aeedfd97e3
=> => extracting sha256:5b142346550416c75ea412d21741de5eaf3e76857aff
=> => extracting sha256:ceb4a75630f5375a81b5da051775265a449cdbd3ec73
=> => extracting sha256:f2056621057792c8761cdf5ee802edad84c28a7e5462
=> => extracting sha256:ee03e037f8b64b93846e433c846d6a019856878ad5c7
=> => extracting sha256:8220bbf1aee7c576d5e9d1ae415171dfe7aeedfd97e3
=> => exporting to image
=> => exporting layers
=> => writing image sha256:7f8dc7bf63e7e0d8be16bfd564a816
=> => naming to docker.io/library/hw02:latest
```

The terminal also shows a command prompt: `~/cs346/hw02 $`. A note at the bottom of the terminal says: "Use 'docker scan' to run Snyk tests against images to find".

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 tab in VS Code with the following content:

```
hw02 > Dockerfile > ...
1 FROM httpd:2.4
2
```

The Explorer sidebar shows a folder structure under CS346/hw02, with a Dockerfile selected. The Terminal tab shows the command `~/cs346/hw02 $ docker images` and its output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
httpd	2.4	b5543eff25e7	13 days ago	137MB
hw02	latest	7f8dc7bf63e7	13 days ago	137MB

The status bar at the bottom right indicates "Ln 2, 0".

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 VS Code interface with the Docker extension. The Explorer sidebar shows a project structure with 'CS346' expanded, 'hw02' selected, and 'Dockerfile' highlighted. The Dockerfile tab shows the contents:

```
hw02 > Dockerfile > ...
1 FROM httpd:2.4
2
```

The Terminal tab shows the output of running the Docker command:

```
~/cs346/hw02 $ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
httpd           2.4      b5543eff25e7  13 days ago  137MB
hw02            latest   7f8dc7bf63c7  13 days ago  137MB
~/cs346/hw02 $ docker run -it --rm -p 8080:80 hw02:latest
AH00558: httpd: could not reliably determine the server's
2. Set the 'ServerName' directive globally to suppress thi
AH00558: httpd: Could not reliably determine the server's
2. Set the 'ServerName' directive globally to suppress thi
[Mon Sep 05 16:21:11.029126 2022] [mpm_event:notice] [pid
4 (Unix) configured -- resuming normal operations
[Mon Sep 05 16:21:11.029719 2022] [core:notice] [pid 1:tid
tpd -D FOREGROUND'
```

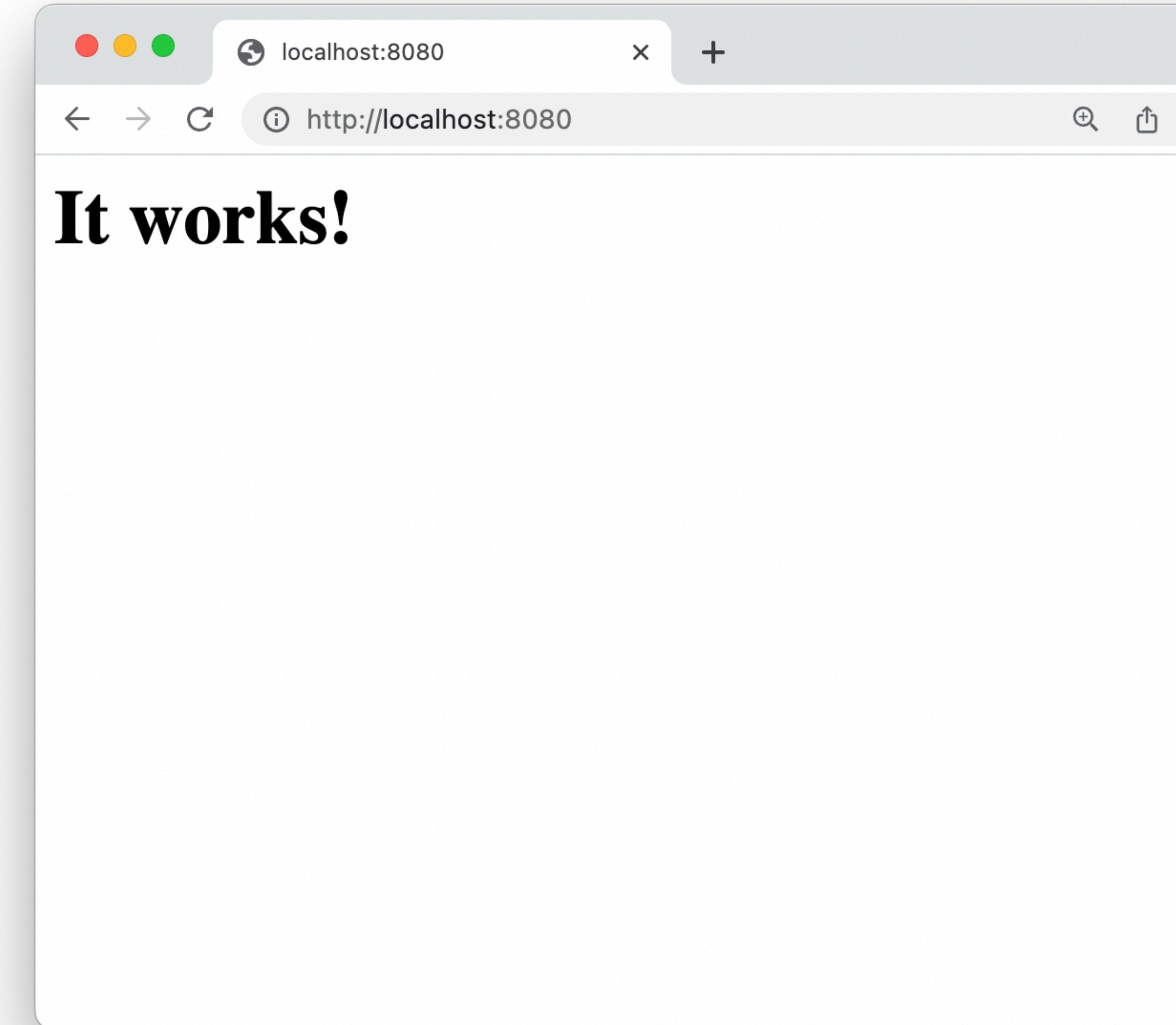
A blue oval highlights the command `docker run -it --rm -p 8080:80 hw02:latest` in the terminal output.

```
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

- 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

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar, which displays a tree view of a project structure under 'CS346/hw02'. The 'Dockerfile' file is selected in the tree. The main workspace shows a 'Dockerfile' editor tab with the following content:

```
FROM httpd:2.4
```

Below the editor is a tab bar with 'TERMINAL' selected. The terminal pane displays the following log output:

```
~/cs346/hw02 $ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
httpd           2.4      b5543eff25e7  13 days ago  137MB
hw02            latest   7f8dc7bf63e7  13 days ago  137MB
~/cs346/hw02 $ docker run -it --rm -p 8080:80 hw02:latest
AH00558: httpd: Could not reliably determine the server's fully qualified domain name
2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name
2. Set the 'ServerName' directive globally to suppress this message
[Mon Sep 05 16:21:11.029126 2022] [mpm_event:notice] [pid 1:tid 281473735553040] AH00094: apd -D FOREGROUND
172.17.0.1 - - [05/Sep/2022:16:25:32 +0000] "GET / HTTP/1.1" 200 45
172.17.0.1 - - [05/Sep/2022:16:25:32 +0000] "GET /favicon.ico HTTP/1.1" 404 196
172.17.0.1 - - [05/Sep/2022:16:26:23 +0000] "-" 408 -
```

A blue rectangular box highlights the last three log entries at the bottom of the terminal output.

At the bottom right of the terminal pane, status information is displayed: 'Ln 2, Col 1' (line 2, column 1), 'Spaces: 4', 'UTF-8', and a small icon.

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 a Microsoft Visual Studio Code interface with a Dockerfile open in the editor. The file contains the following code:

```
FROM httpd:2.4
```

The terminal tab shows the following output from a docker run command:

```
~/cs346/hw02 $ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
httpd           2.4      b5543eff25e7  13 days ago  137MB
hw02            latest    7f8dc7bf63e7  13 days ago  137MB

~/cs346/hw02 $ docker run -it --rm -p 8080:80 hw02:latest
AH00558: httpd: Could not reliably determine the server's fully qualified domain name
2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name
2. Set the 'ServerName' directive globally to suppress this message
[Mon Sep 05 16:21:11.029126 2022] [mpm_event:notice] [pid 1:tid 281473735553040] AH00094: 
4 (Unix) configured -- resuming normal operations
[Mon Sep 05 16:21:11.029719 2022] [core:notice] [pid 1:tid 281473735553040] AH00094: 
tpd -D FOREGROUND'
172.17.0.1 - - [05/Sep/2022:16:25:32 +0000] "GET / HTTP/1.1" 200 45
172.17.0.1 - - [05/Sep/2022:16:25:32 +0000] "GET /favicon.ico HTTP/1.1" 404 196
172.17.0.1 - - [05/Sep/2022:16:26:23 +0000] "-" 408 -
^C[Mon Sep 05 16:38:51.383564 2022] [mpm_event:notice] [pid 1:tid 281473735553040] TERM, shutting down
~/cs346/hw02 $
```

The terminal window has tabs for TERMINAL, PROBLEMS, OUTPUT, DEBUG CONSOLE, and JUPYTER. The status bar at the bottom right shows "Ln 2, Col 1 Spaces: 4 UTF-8".

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]
```

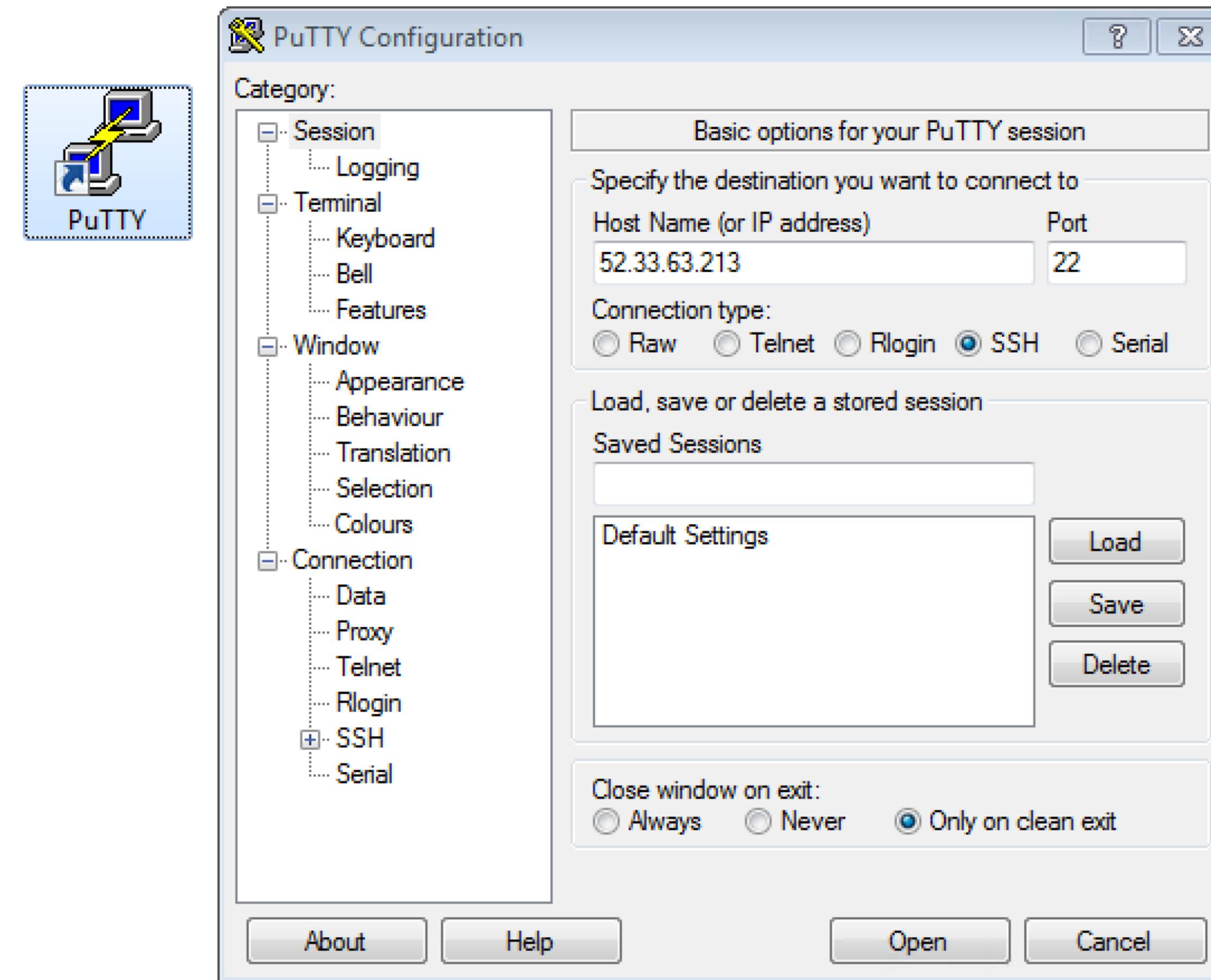
Connecting to Remote Hosts

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

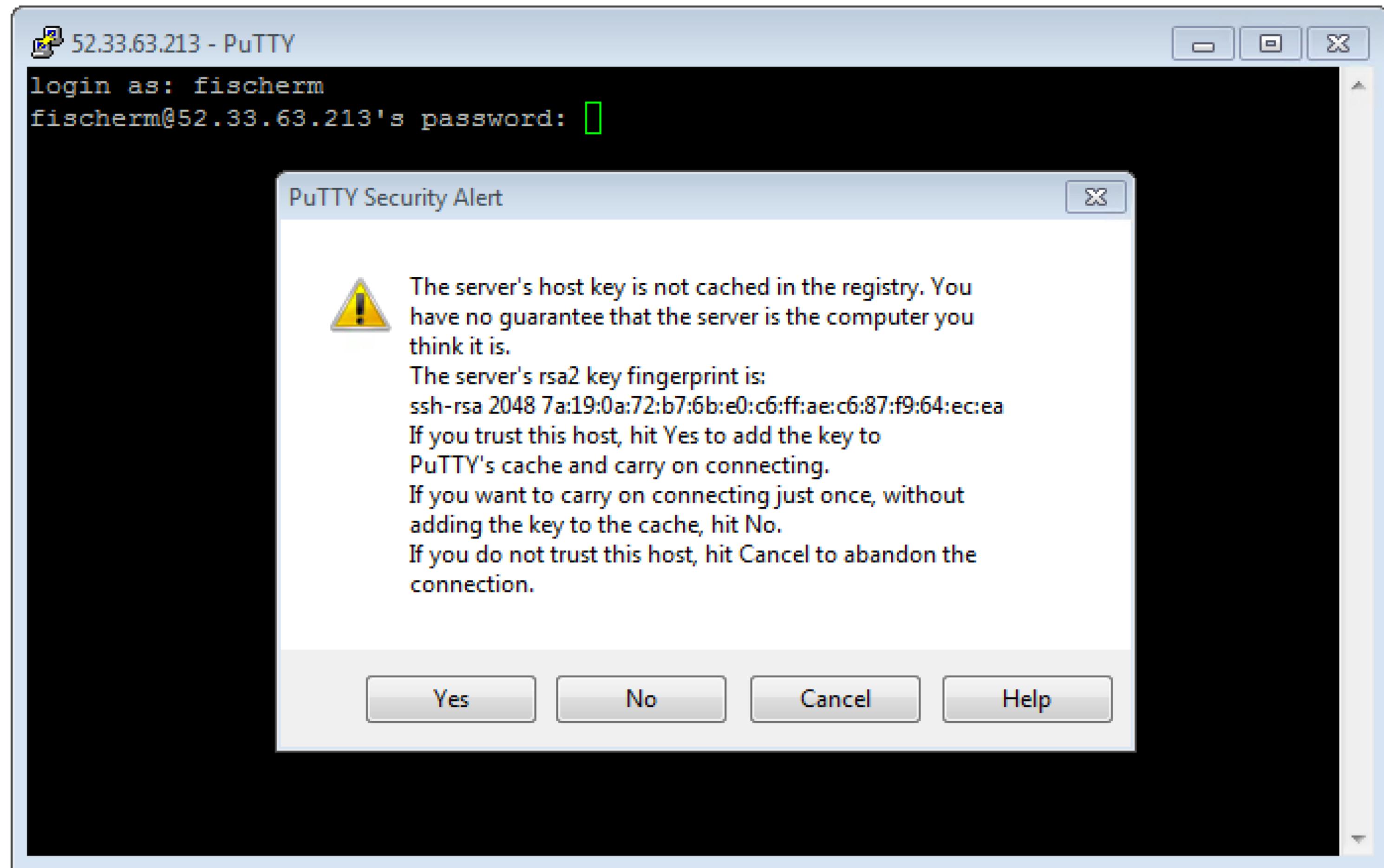
Putty



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

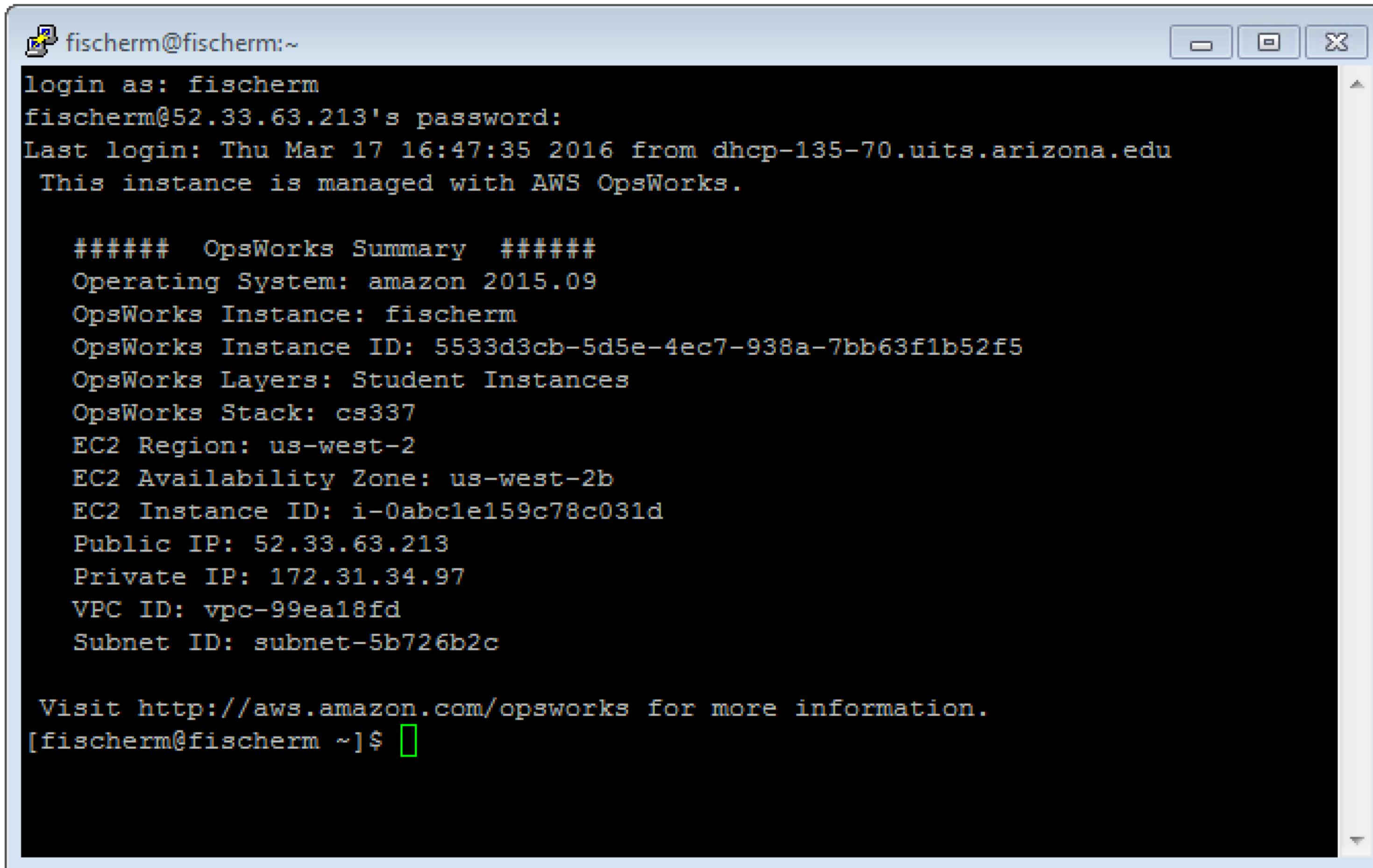
Connecting to Remote Hosts

Putty



Connecting to Remote Hosts

Putty



fischerm@fischerm:~

```
login as: fischerm
fischerm@52.33.63.213's password:
Last login: Thu Mar 17 16:47:35 2016 from dhcp-135-70.uits.arizona.edu
This instance is managed with AWS OpsWorks.

##### OpsWorks Summary #####
Operating System: amazon 2015.09
OpsWorks Instance: fischerm
OpsWorks Instance ID: 5533d3cb-5d5e-4ec7-938a-7bb63f1b52f5
OpsWorks Layers: Student Instances
OpsWorks Stack: cs337
EC2 Region: us-west-2
EC2 Availability Zone: us-west-2b
EC2 Instance ID: i-0abc1e159c78c031d
Public IP: 52.33.63.213
Private IP: 172.31.34.97
VPC ID: vpc-99ea18fd
Subnet ID: subnet-5b726b2c

Visit http://aws.amazon.com/opsworks for more information.
[fischerm@fischerm ~] $
```



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
```

next up: The HTTP Protocol and Networking