

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.

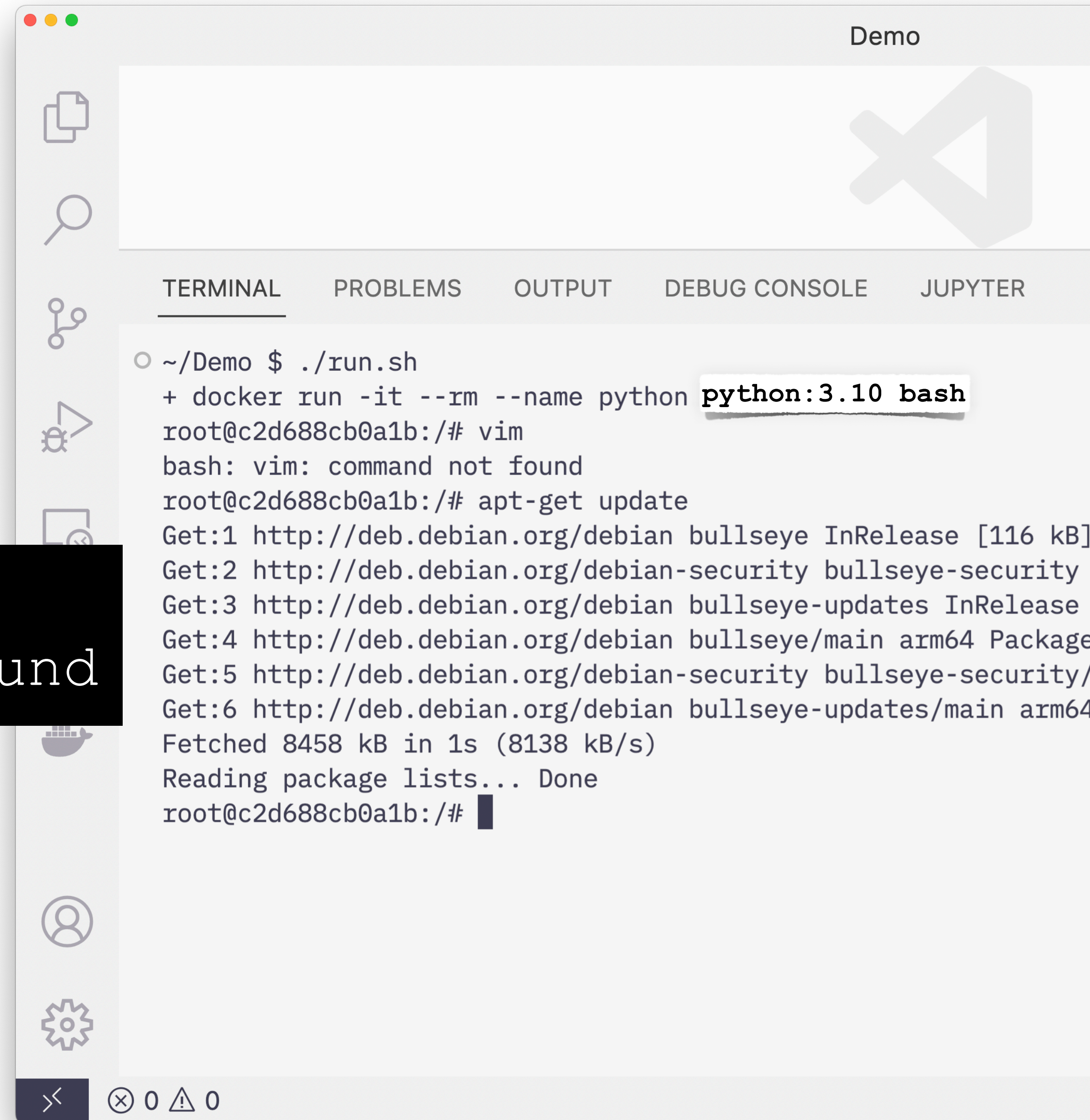
ubuntu

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 titled "Demo" with a sidebar containing icons for file operations, search, and settings. The terminal output is as follows:

```
~/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 InRelease [46.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [46.4 kB]
Get:4 http://deb.debian.org/debian bullseye/main arm64 Packages [90.1 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main arm64 Packages [10.8 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main arm64 Packages [10.8 kB]
Fetched 8458 kB in 1s (8138 kB/s)
Reading package lists... Done
root@c2d688cb0a1b:/#
```

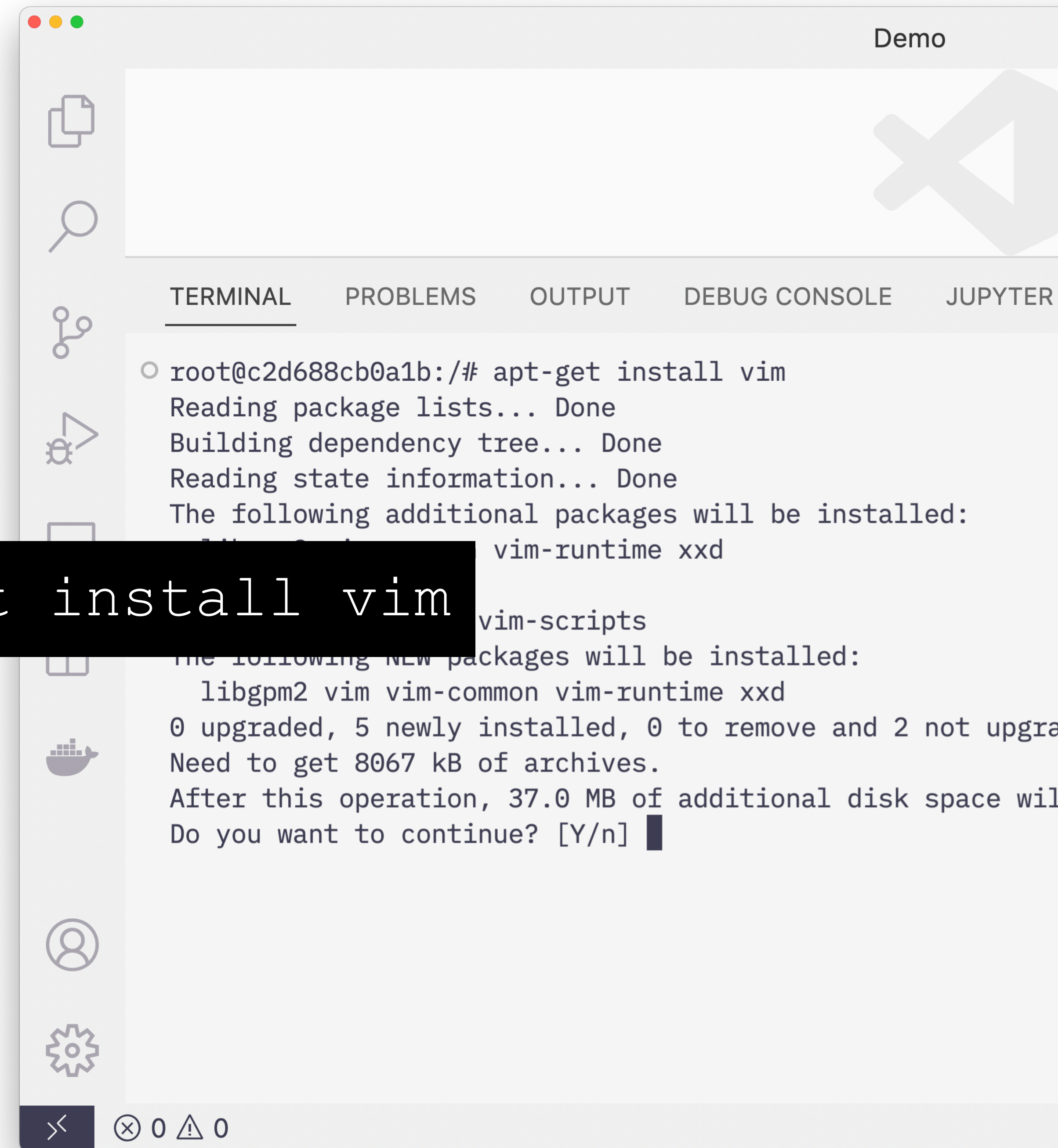

ubuntu

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



The screenshot shows a terminal window titled "Demo" with a sidebar on the left containing icons for file operations, search, and system settings. The terminal output shows the command `apt-get install vim` being executed. The output includes the following text:

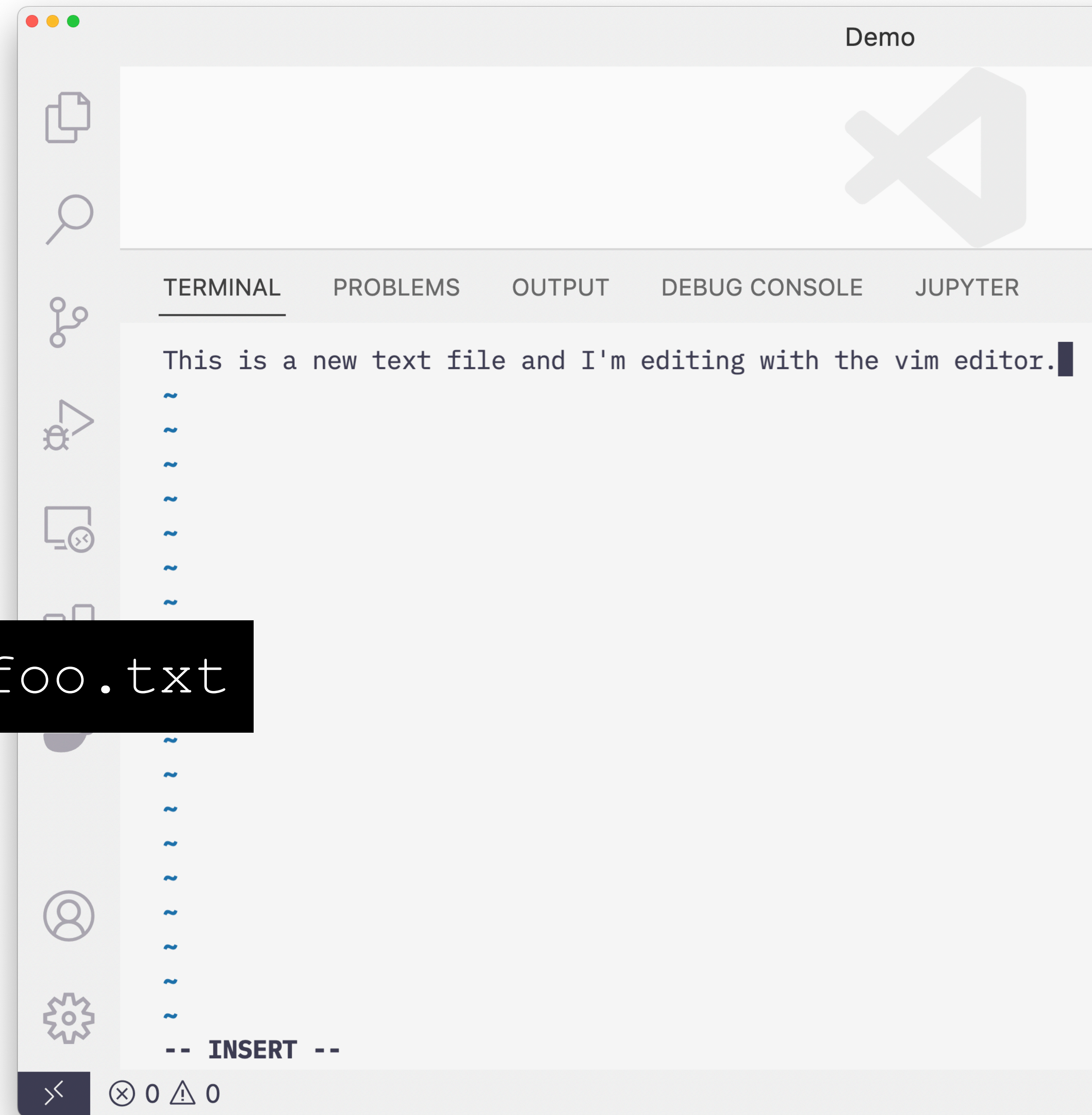
```
root@c2d688cb0a1b:/# apt-get install vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  vim-runtime xxd
The following NEW packages will be installed:
  libgpm2 vim vim-common vim-runtime xxd
0 upgraded, 5 newly installed, 0 to remove and 2 not upgraded.
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.

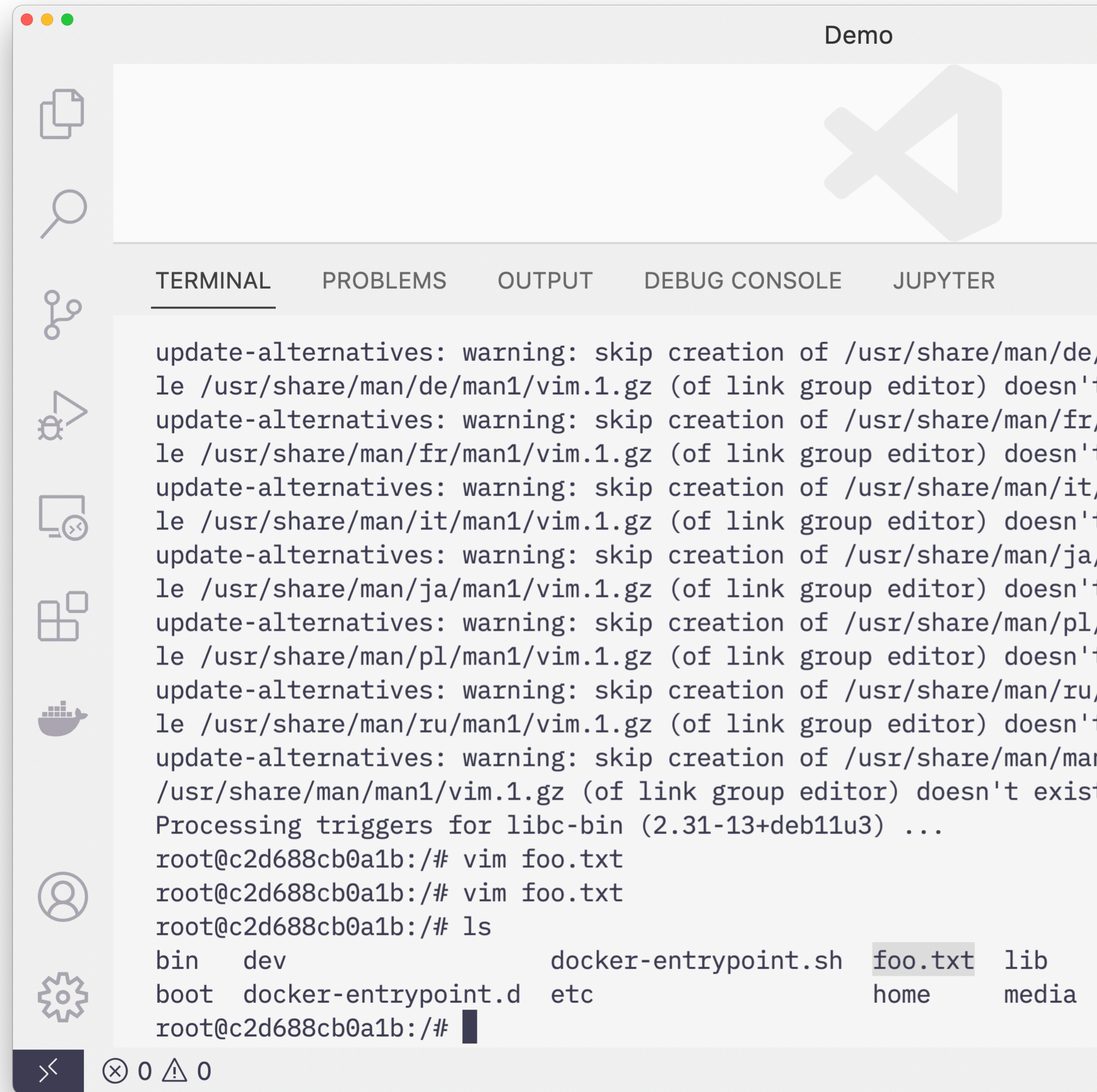
```
root@c2d688cb0a1b:/# vim foo.txt
```



ubuntu

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 titled "Demo" with a search icon in the top right. The terminal output displays the following commands and their results:

```
update-alternatives: warning: skip creation of /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/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /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/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /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/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /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:/#
```

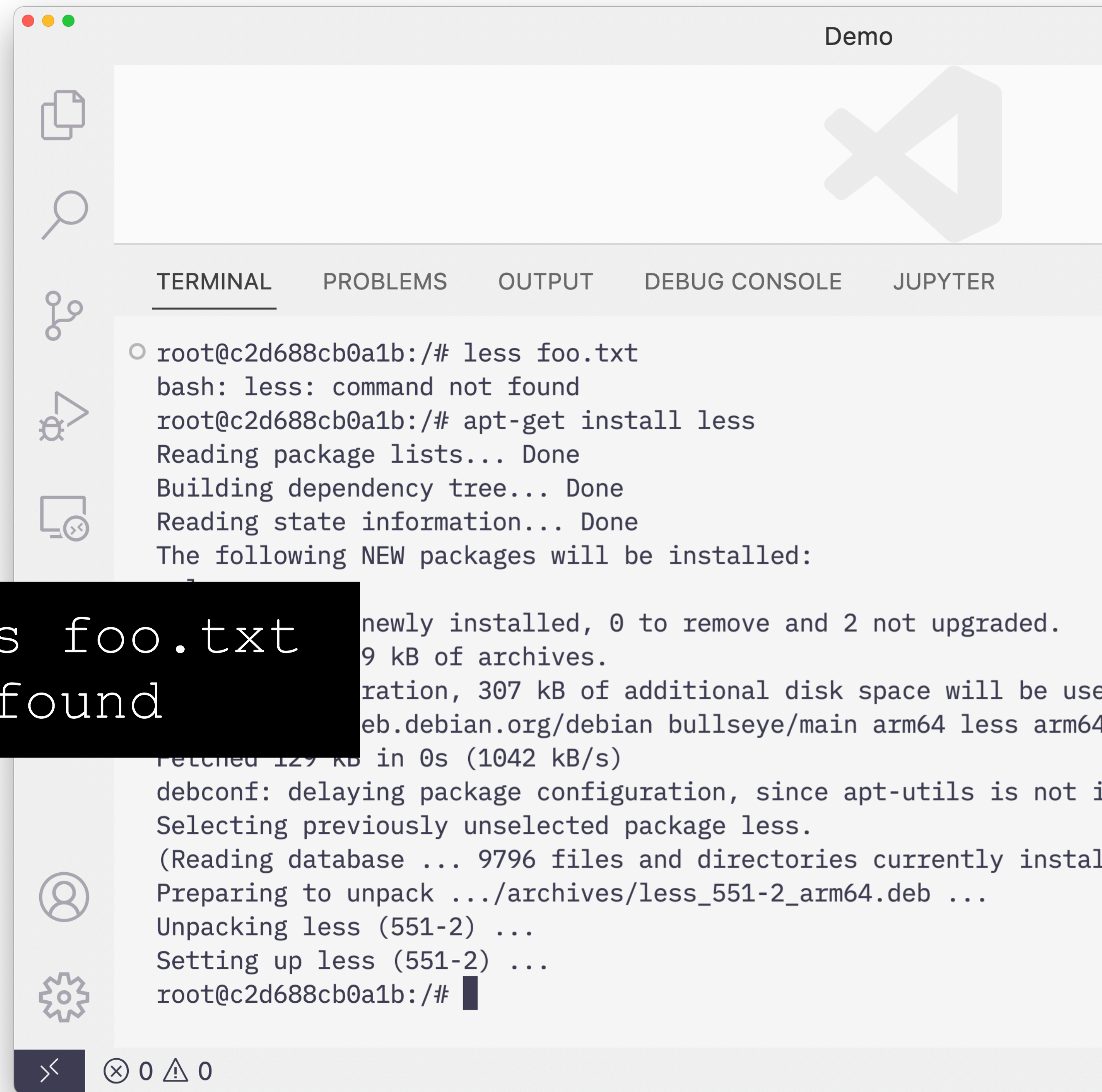
The terminal window includes a sidebar with icons for file operations, search, and other functions. The status bar at the bottom shows navigation arrows and window management icons.

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
```



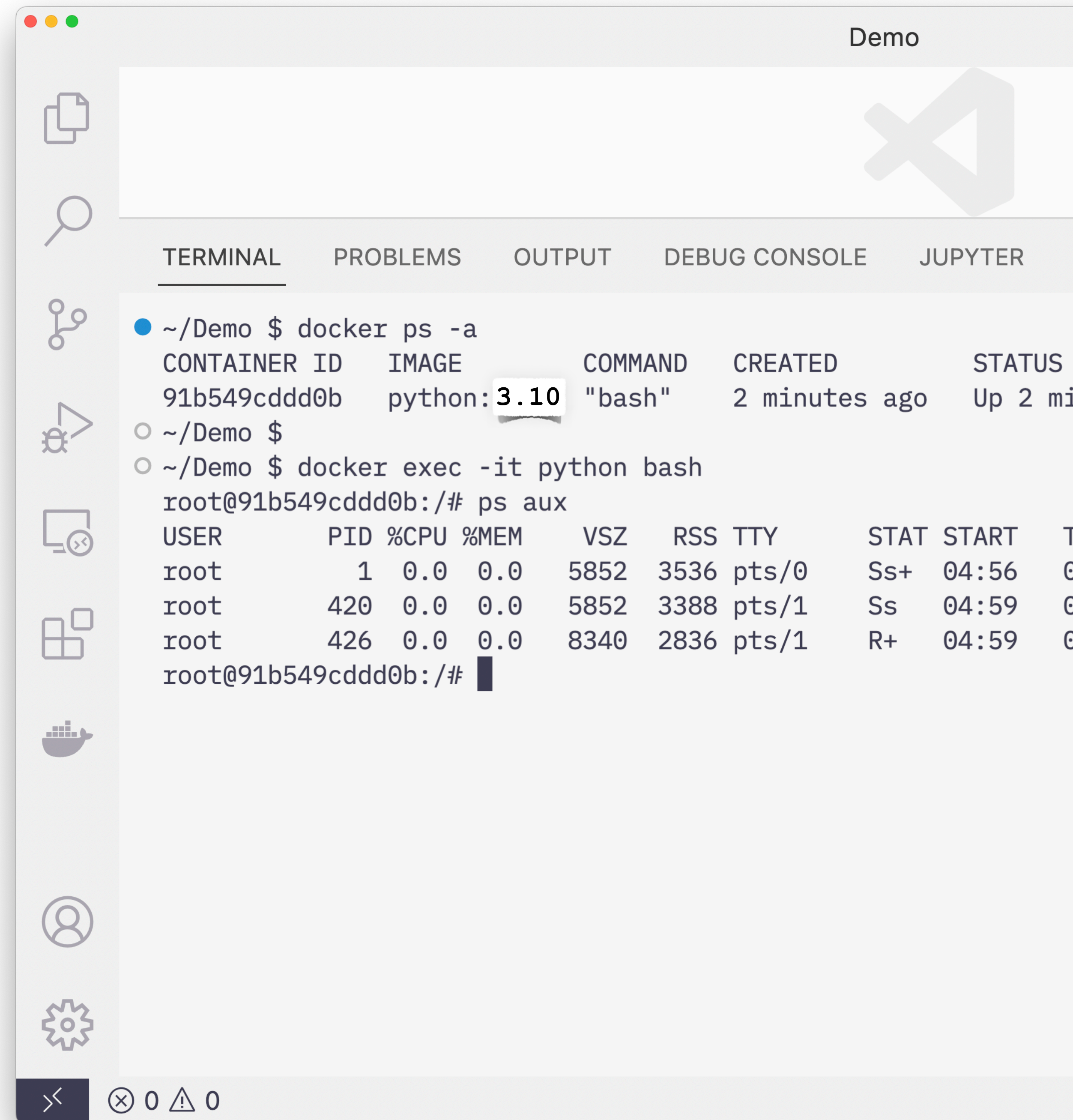
The screenshot shows a terminal window titled "Demo" with a sidebar on the left containing icons for file operations, search, and settings. The terminal output is as follows:

```
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
0 packages newly installed, 0 to remove and 2 not upgraded.
0 kB of archives.
0 kB of additional disk space will be used.
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 a VS Code terminal window titled "Demo" with the Docker logo in the background. The terminal output is as follows:

```
~/Demo $ docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS
91b549cddd0b   python:3.10   "bash"          2 minutes ago   Up 2 mi
~/Demo $
~/Demo $ docker exec -it python bash
root@91b549cddd0b:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   T
root           1  0.0  0.0   5852   3536 pts/0    Ss+   04:56   G
root          420  0.0  0.0   5852   3388 pts/1    Ss    04:59   G
root          426  0.0  0.0   8340   2836 pts/1    R+    04:59   G
root@91b549cddd0b:/#
```


Docker

Multiple C

The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the following commands and output:

```
~/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:/#
```

The terminal output shows a list of running Docker containers. The first container, with ID 91b549cddd0b, is running the python:3.10 image with the command "bash". It was created 2 minutes ago and is currently up. The second part of the terminal shows the output of the `ps aux` command inside the container, listing the processes running within it.

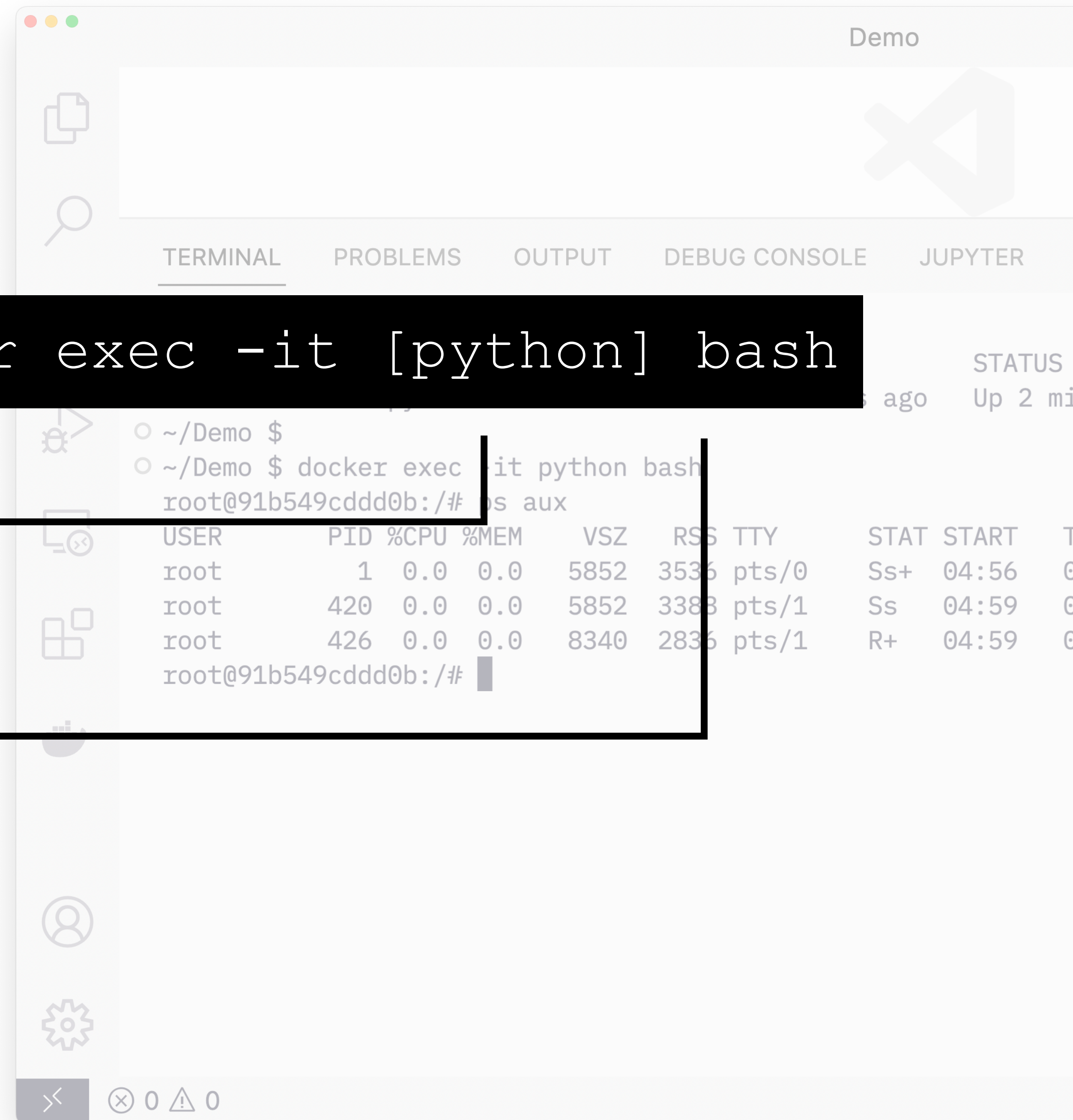
The interface includes a sidebar on the left with various icons, a top bar with the title "Demo" and window controls, and a bottom status bar showing "0" errors and "0" warnings.

Docker

Multiple Container Connections

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

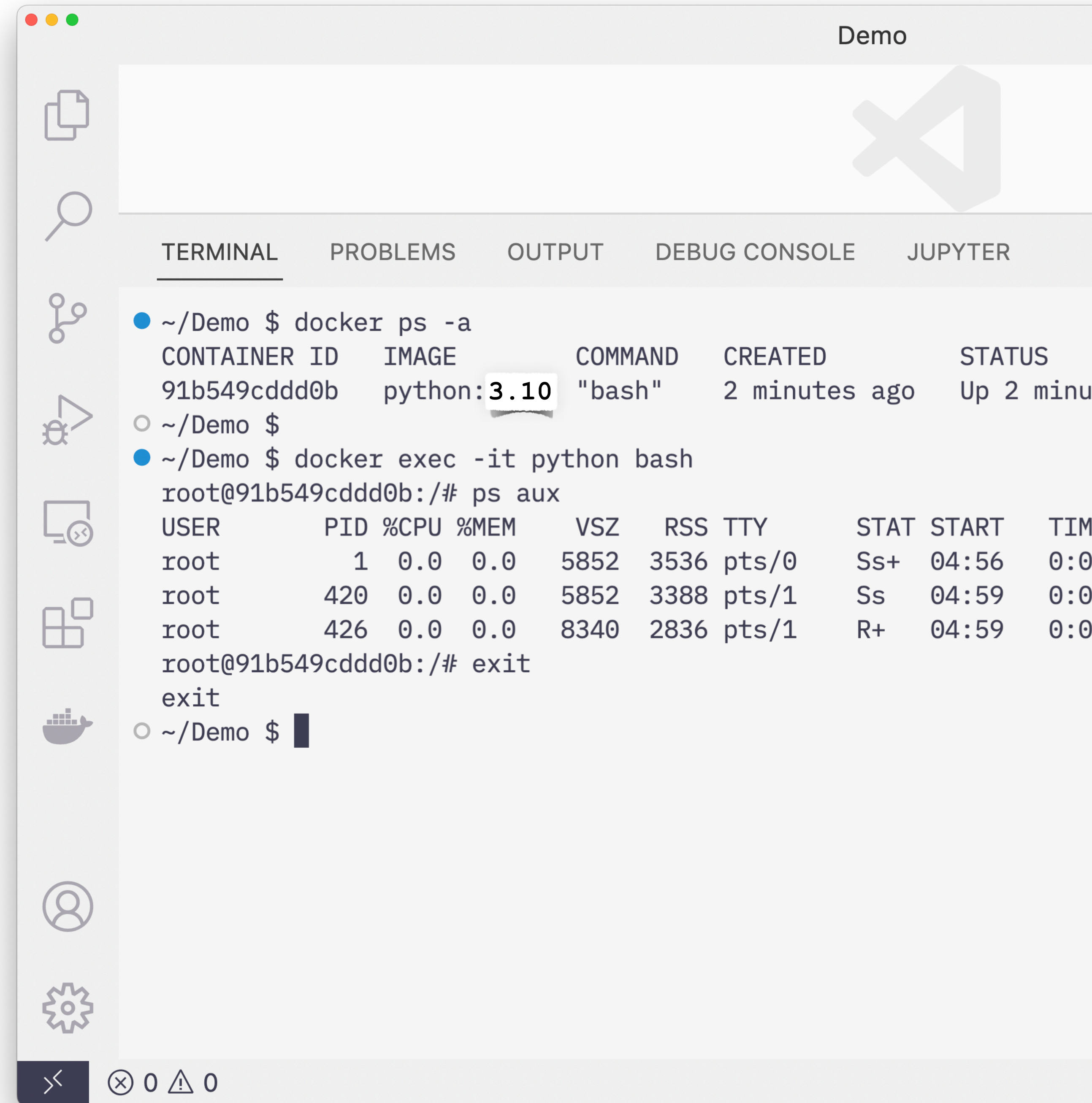
- You need to specify the name or ID of the running container
- You need to specify the command you want to execute in the new container
- In most cases, you want a new bash shell



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 VS Code terminal window titled "Demo" with the Docker logo in the background. The terminal output is as follows:

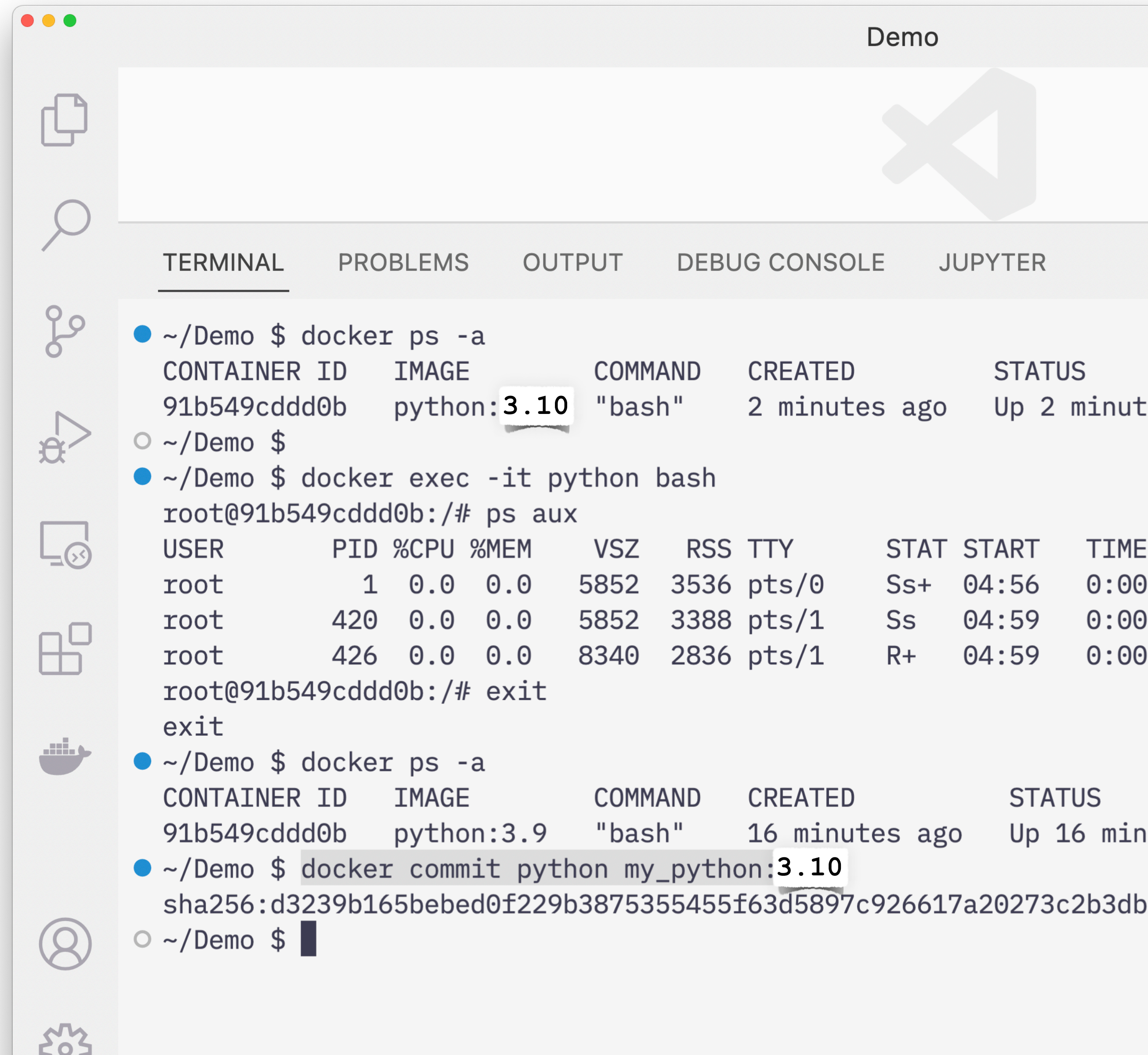
```
~/Demo $ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
91b549cddd0b   python:3.10   "bash"                  2 minutes ago Up 2 minutes
~/Demo $
~/Demo $ docker exec -it python bash
root@91b549cddd0b:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIM
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 $
```

The terminal interface includes tabs for "TERMINAL", "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "JUPYTER". The status bar at the bottom shows a back arrow, a close button, and "0" errors and warnings.

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 a VS Code terminal window titled "Demo" with the Docker logo in the top right. The terminal has tabs for "TERMINAL", "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "JUPYTER". The terminal output shows the following sequence of commands and results:

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

~/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 min

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

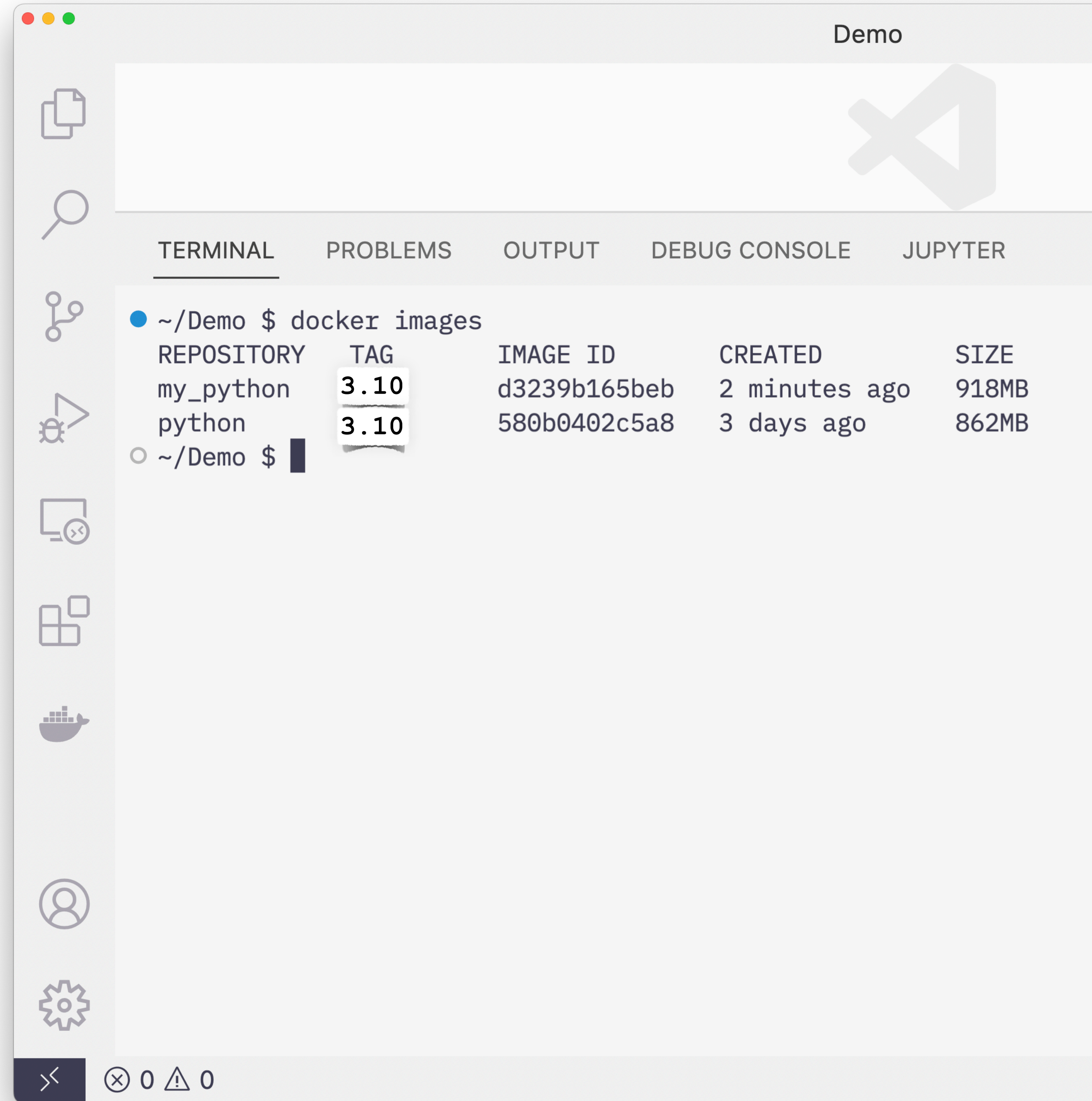
~/Demo $
```

```
$ 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 a terminal window in VS Code with the following content:

```
~/Demo $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my_python	3.10	d3239b165beb	2 minutes ago	918MB
python	3.10	580b0402c5a8	3 days ago	862MB

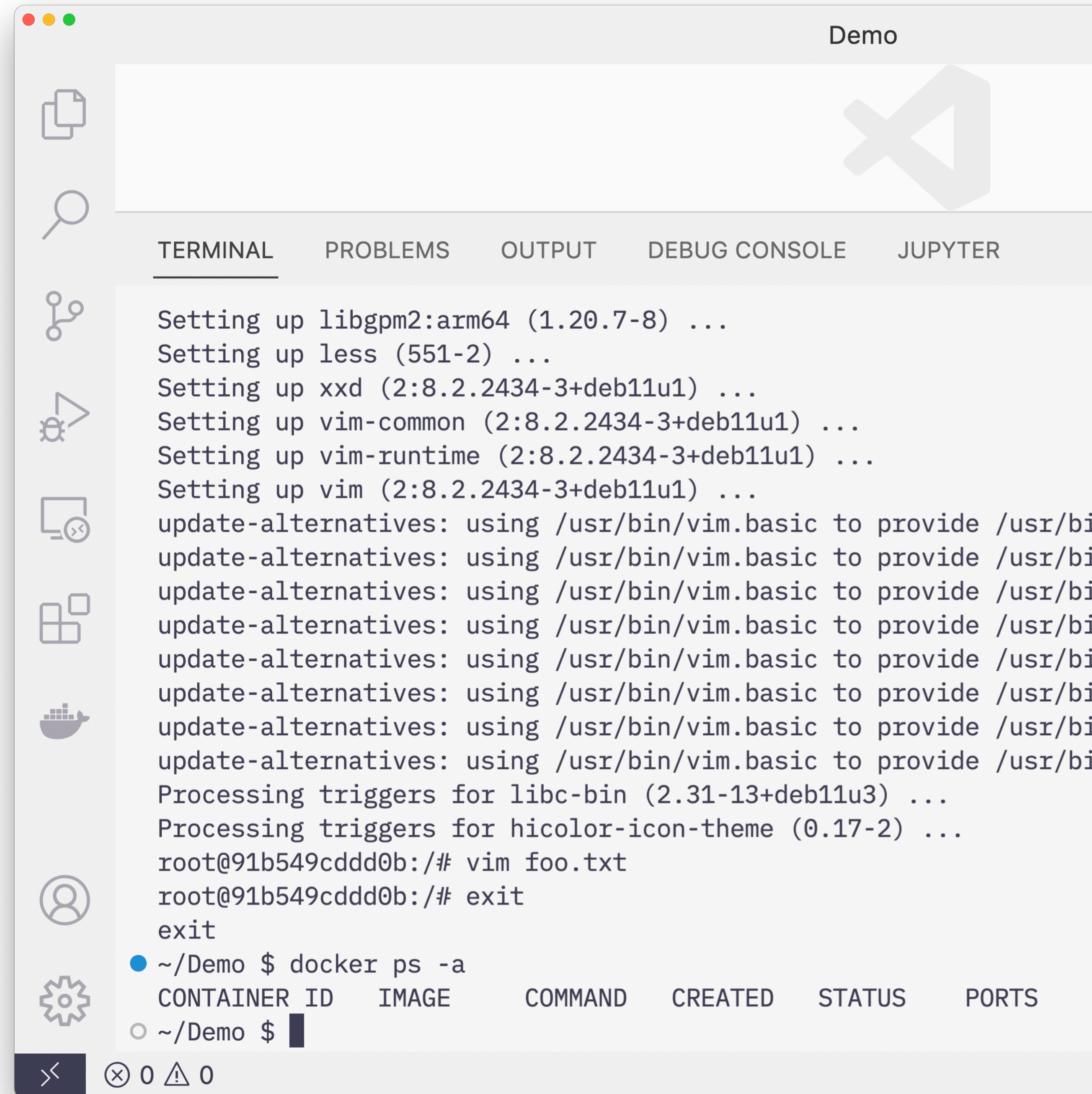
```
~/Demo $
```

The terminal window also shows the VS Code interface with the 'TERMINAL' tab selected and the Docker logo in the background.

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 a terminal window titled "Demo" with the Visual Studio Code logo in the top right. The terminal output shows the installation of various packages in a container, followed by the execution of `vim foo.txt` and `exit`. Below the terminal output, a `docker ps -a` command is shown, which would list the container's details. The terminal prompt is `~/Demo $`.

```
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
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@91b549cddd0b:/# vim foo.txt
root@91b549cddd0b:/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
~/Demo $
```

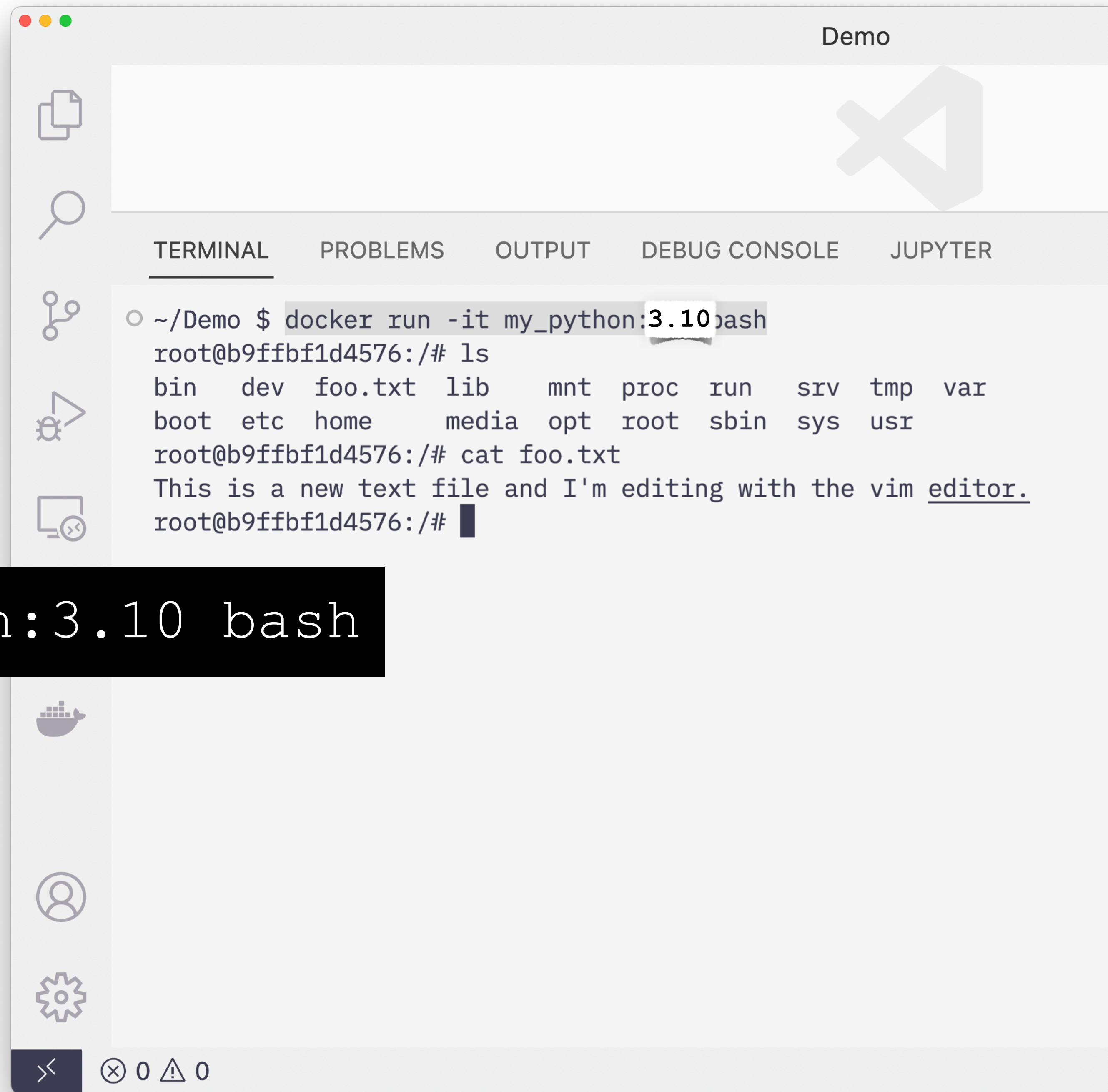

Docker

Saving with `docker commit`

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

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

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



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.



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + v [window icon] [trash icon] [up arrow icon] [close icon]

~/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:/#



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:/#
```




TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

docker + v [icons]

```
~/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:/#
```




TERMINAL

PROBLEMS

OUTPUT

DEBUG CONSOLE

JUPYTER



docker



```
~/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:/#
```




TERMINAL

PROBLEMS

OUTPUT

DEBUG CONSOLE

JUPYTER



docker



```
~/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

```
Demo
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

```
~/Demo $ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
10aa1f5fc064	my_python:3.10	"bash"	6 seconds ago	Exited (0)		1 second ago

```
~/Demo $ docker images
```

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
sha256:52b1fe98191f94da10cce60322bac2e5ad4b32f5320adbc57ec0821beafd1e04
```

```
~/Demo $
~/Demo $
~/Demo $
~/Demo $
```


Docker

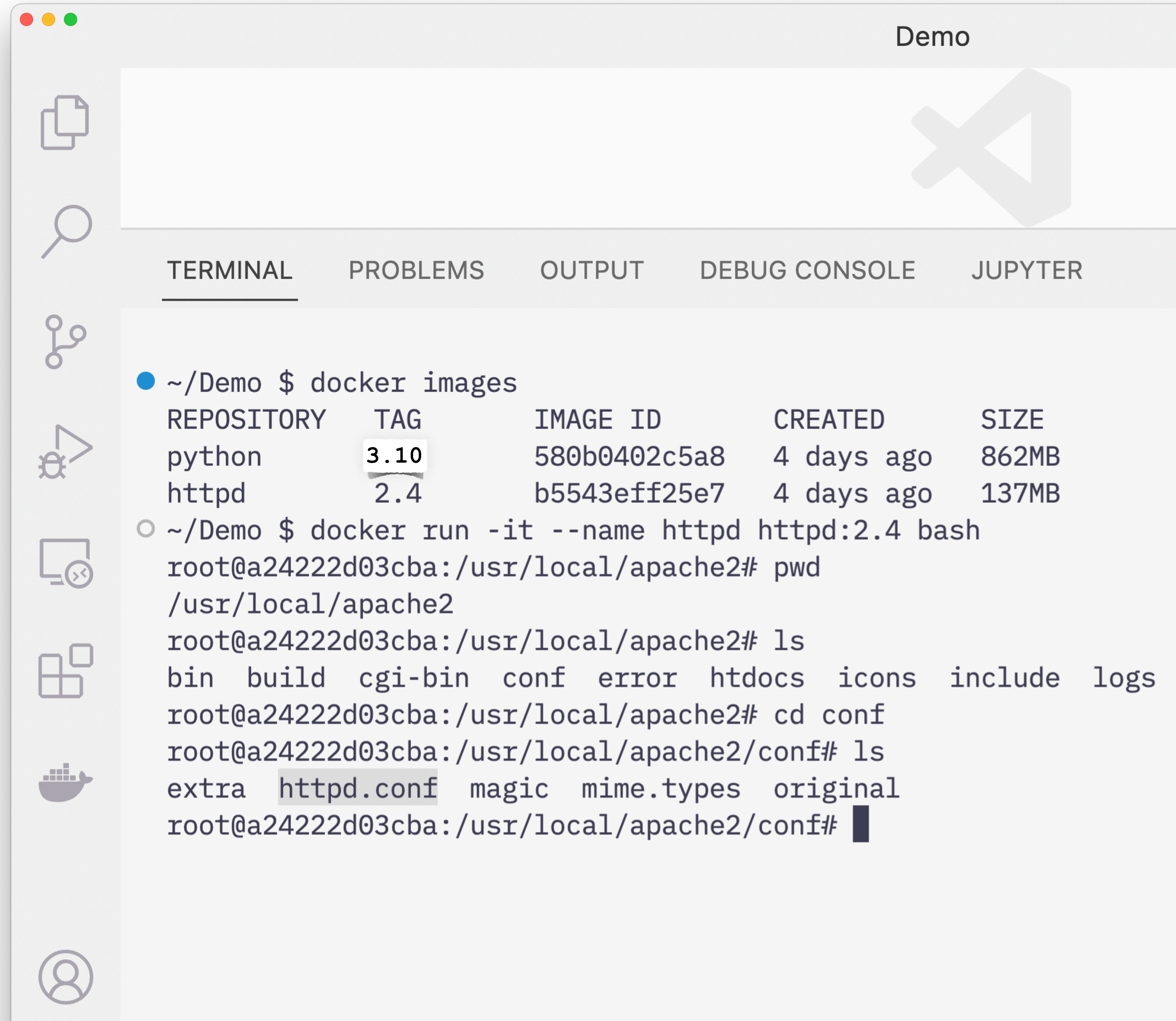
Moving files into and out of a container

- 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 Docker IDE window titled "Demo" with a sidebar on the left containing icons for file explorer, search, Docker Hub, Docker Desktop, Docker Compose, Docker Swarm, and Docker Desktop. The main area has tabs for "TERMINAL", "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "JUPYTER". The terminal output is as follows:

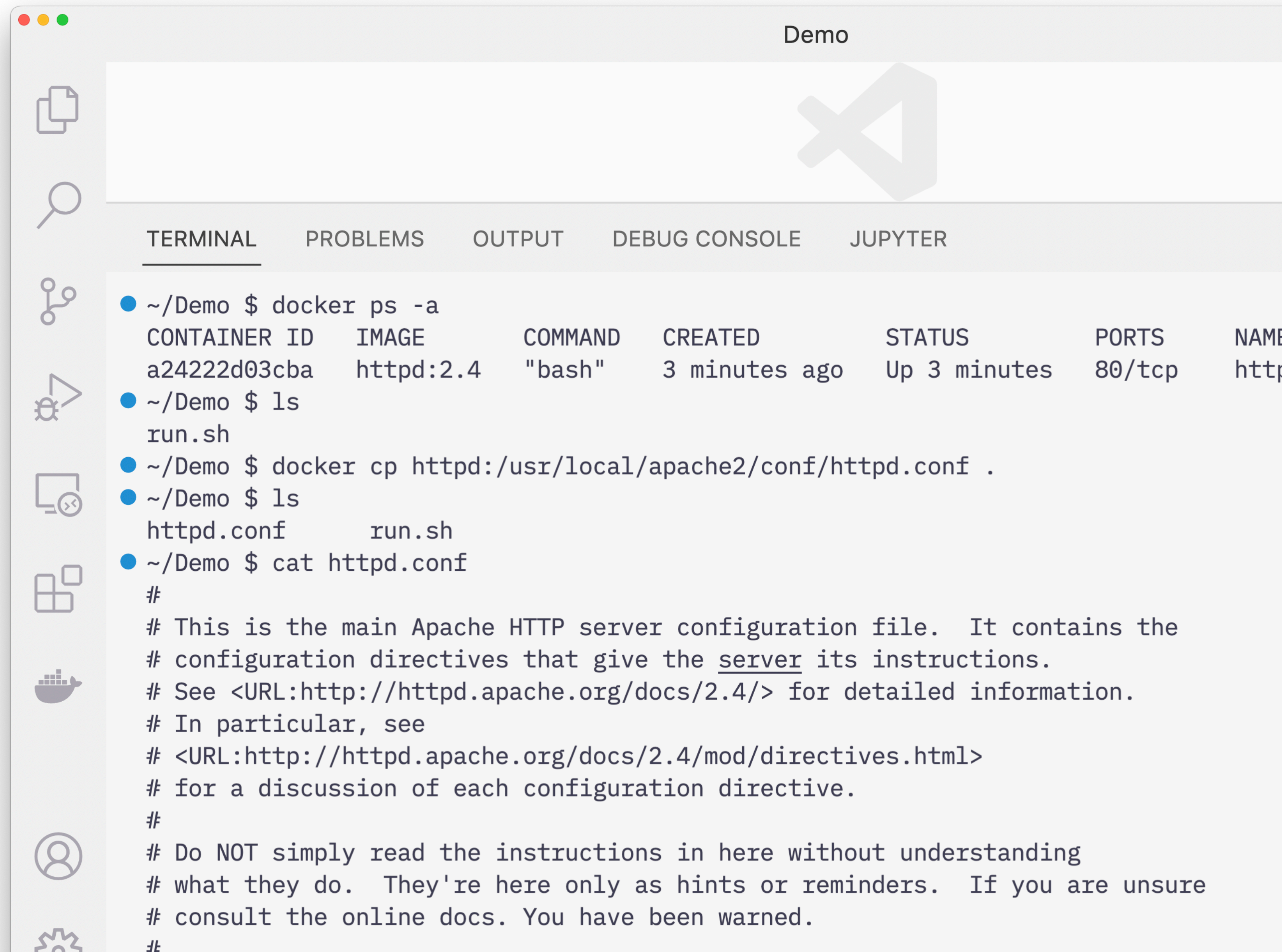
```
~/Demo $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
python        3.10     580b0402c5a8   4 days ago    862MB
httpd         2.4     b5543eff25e7   4 days ago    137MB

~/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 a terminal window titled "Demo" with the Visual Studio Code logo in the top right. The terminal output 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: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 unsure
# consult the online docs. You have been warned.
#
```

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

Docker

Copying Files

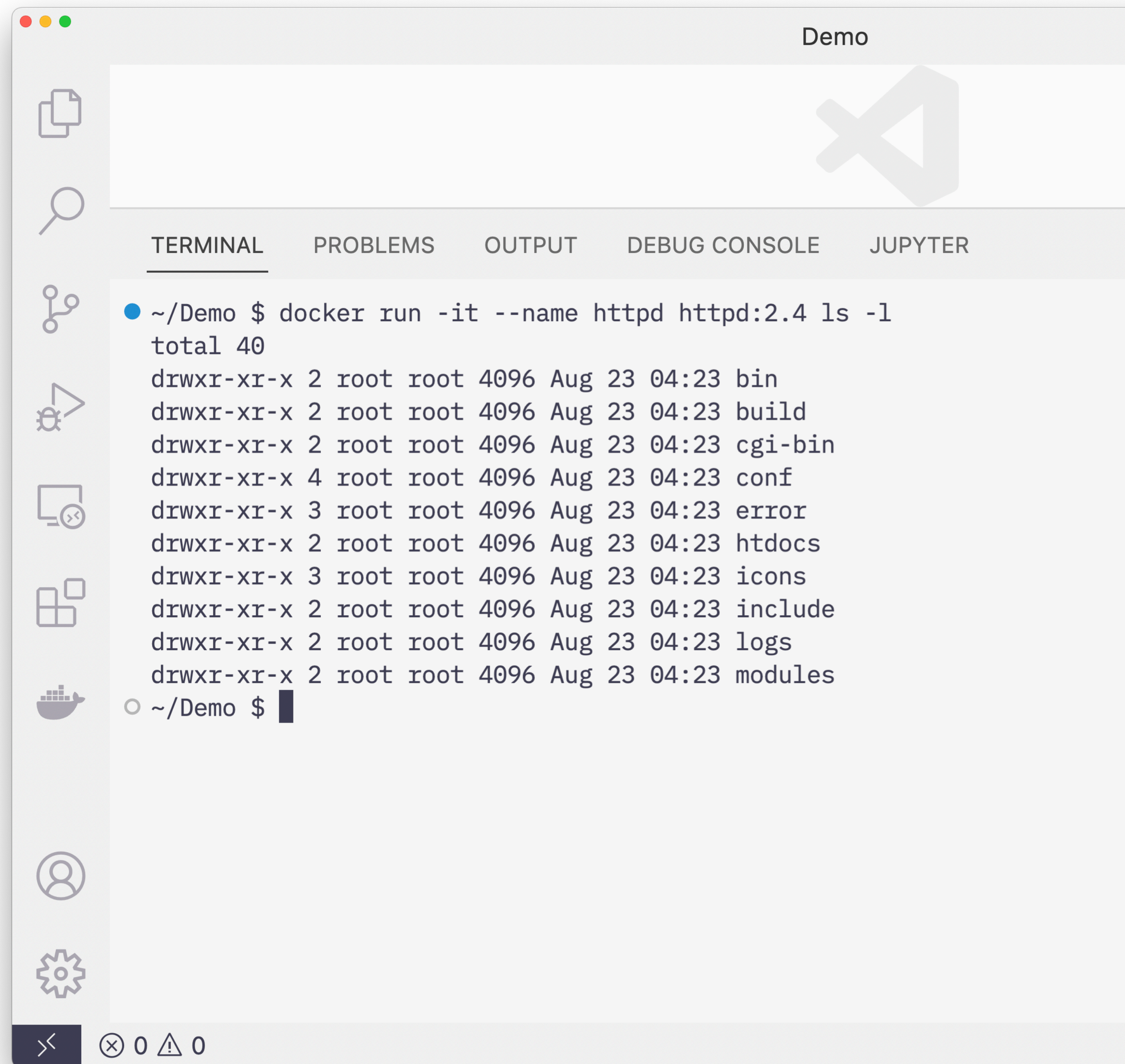
- This works the other way too. You can copy files from your host into a running or stopped container. Just reverse the order of the arguments

```
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 VS Code terminal window titled "Demo". The terminal output is as follows:

```
~/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

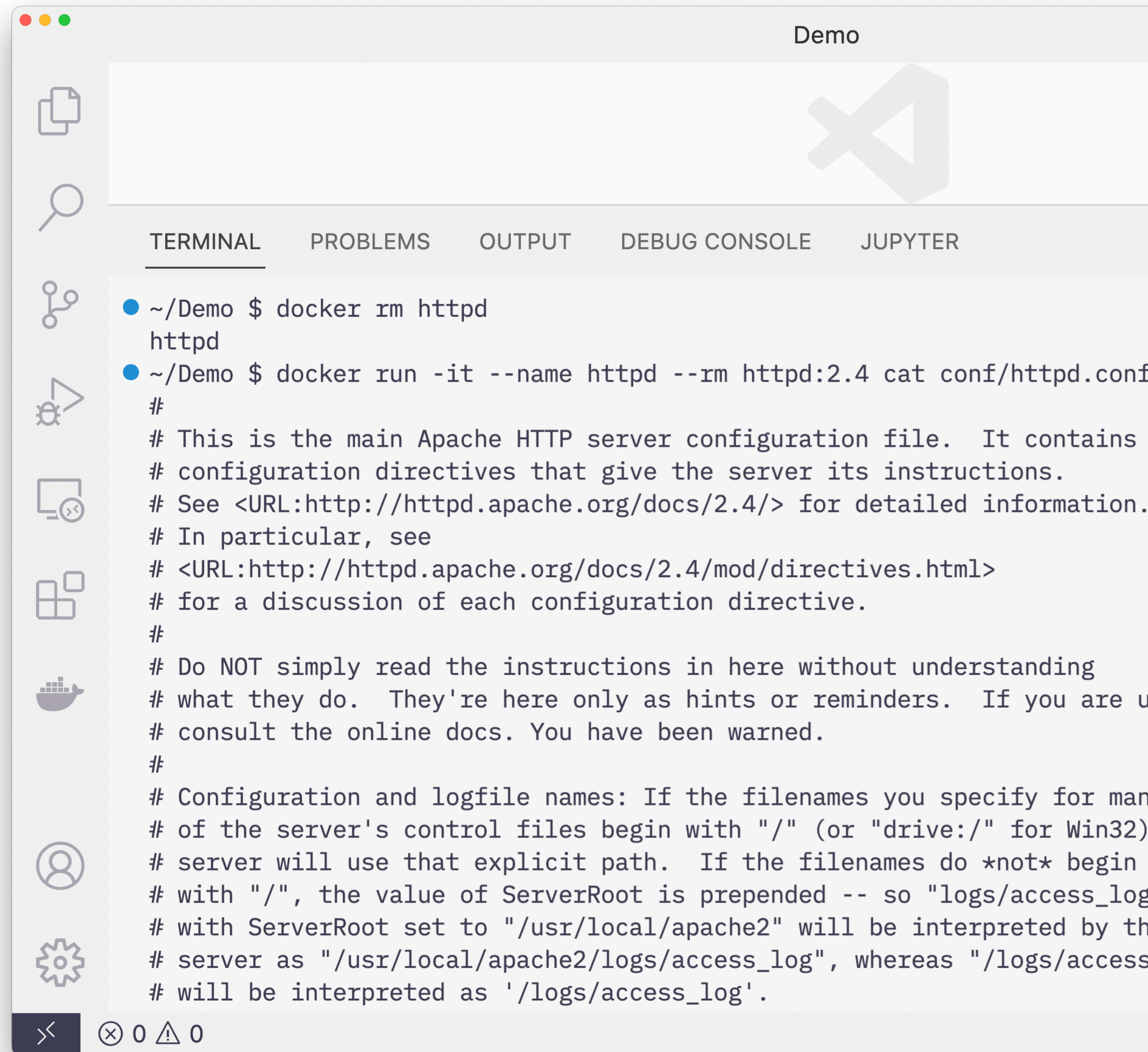
```
drwxr-xr-x 4 root root 4096 Aug 23 04:23 conf
drwxr-xr-x 3 root root 4096 Aug 23 04:23 error
drwxr-xr-x 2 root root 4096 Aug 23 04:23 htdocs
drwxr-xr-x 3 root root 4096 Aug 23 04:23 icons
drwxr-xr-x 2 root root 4096 Aug 23 04:23 include
drwxr-xr-x 2 root root 4096 Aug 23 04:23 logs
drwxr-xr-x 2 root 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 "9eed573c10acb8596335c8c8551bb05fa2c2f79868aaf9a19d717d3fd04491b". 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 $
```

- 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 titled "Demo" with a sidebar on the left containing various icons. The terminal output is as follows:

```
~/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
# consult the online docs. You have been warned.
#
# 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
```

- 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

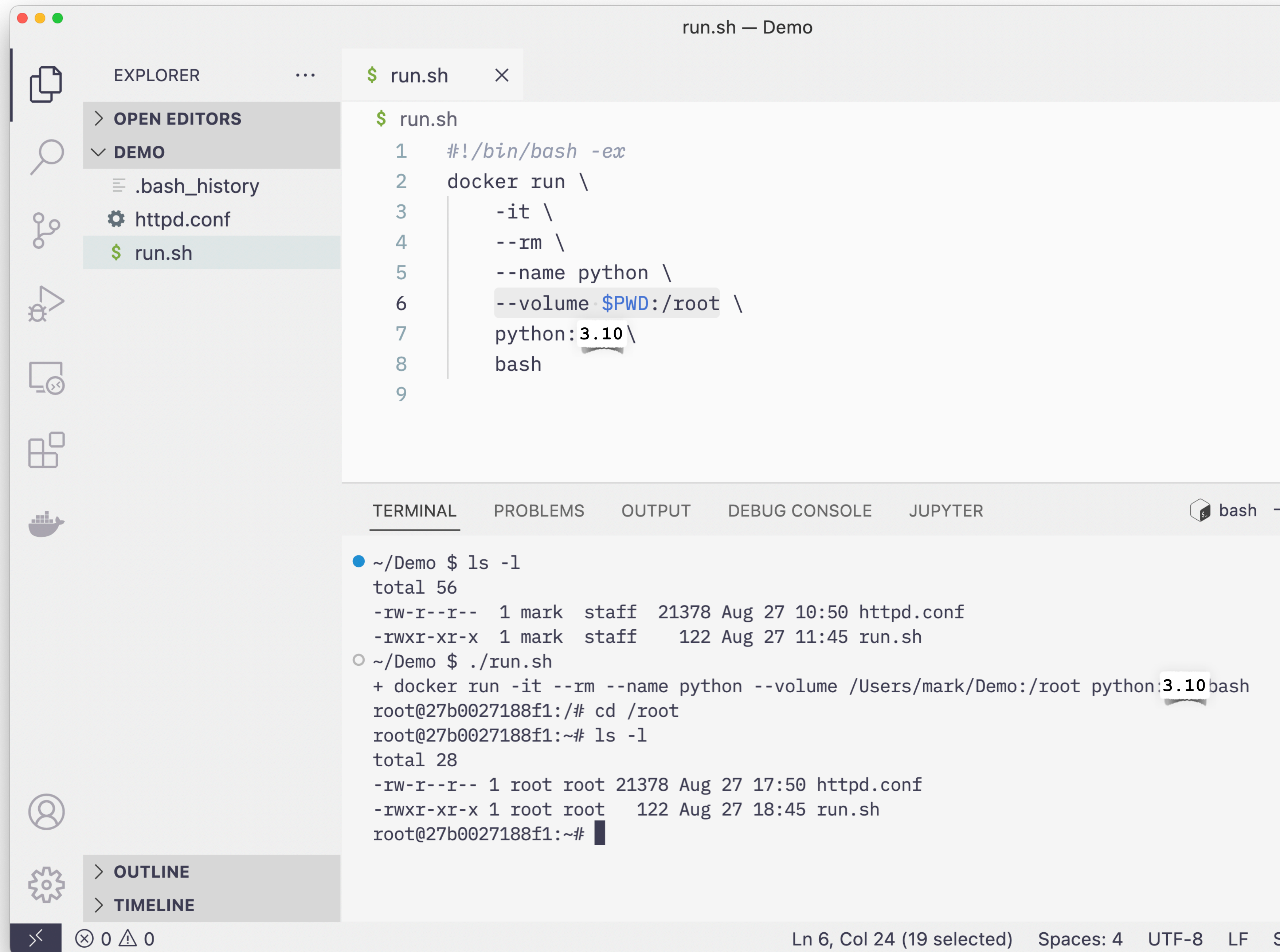
```
docker run --volume $PWD:/root python:3.10
```

```
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
v DEMO
  .bash_history
  httpd.conf
  $ run.sh
  $ 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
PROBLEMS
OUTPUT
DEBUG CONSOLE
JUPYTER
~/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
~/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:~#
```

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



EXPLORER



\$ run.sh

hello.py ×



> OPEN EDITORS

▼ DEMO

▼ work

hello.py

\$ run.sh



> OUTLINE

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/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

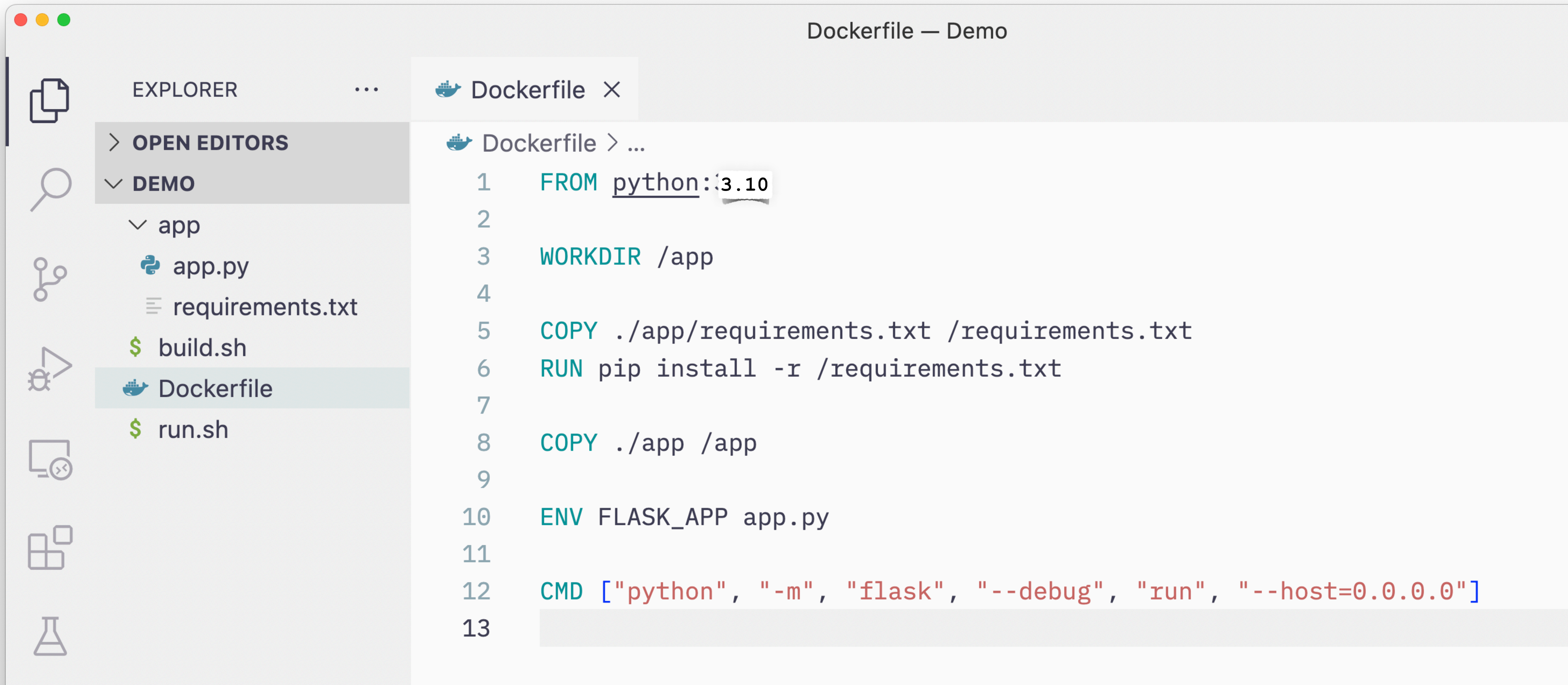
The Dockerfile

- 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

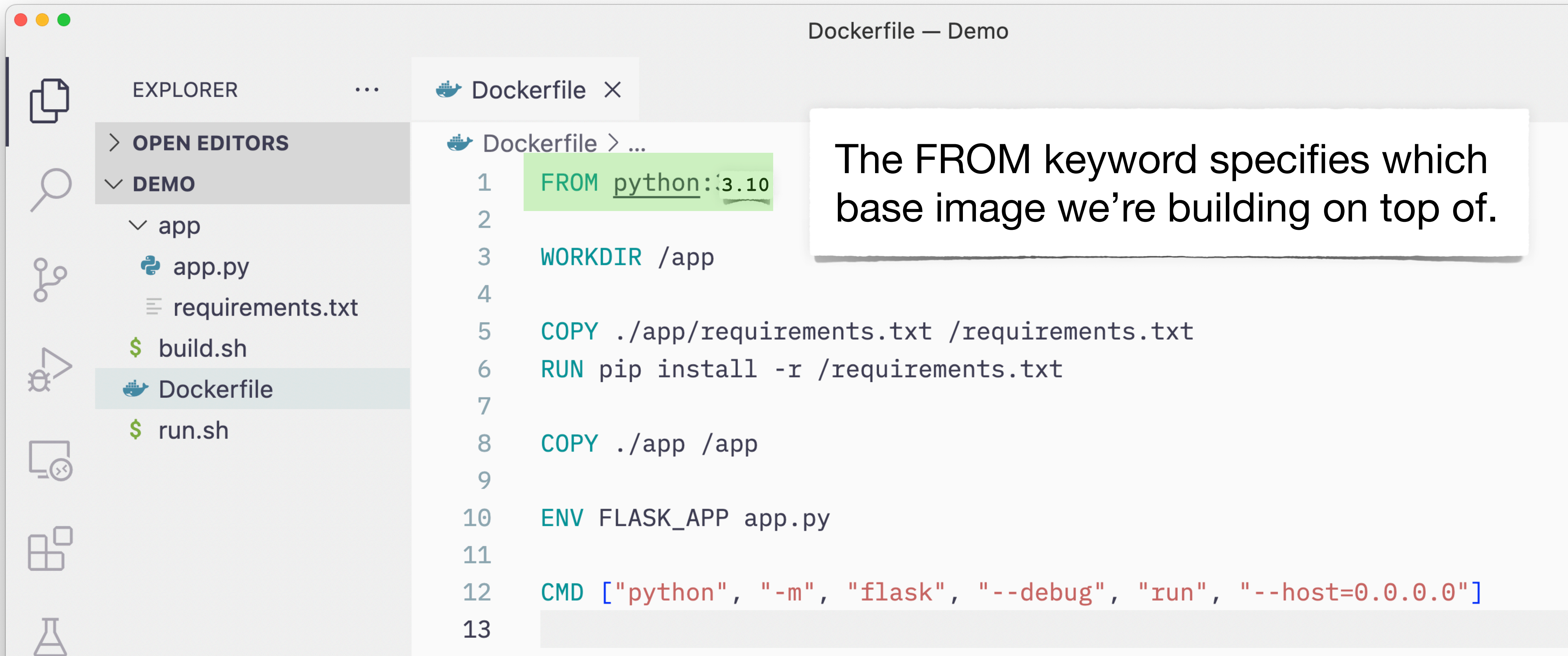


The image shows a screenshot of a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a folder named "DEMO" containing files like "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The "Dockerfile" file is selected and open in the main editor. 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
```


Creating Images

The Dockerfile



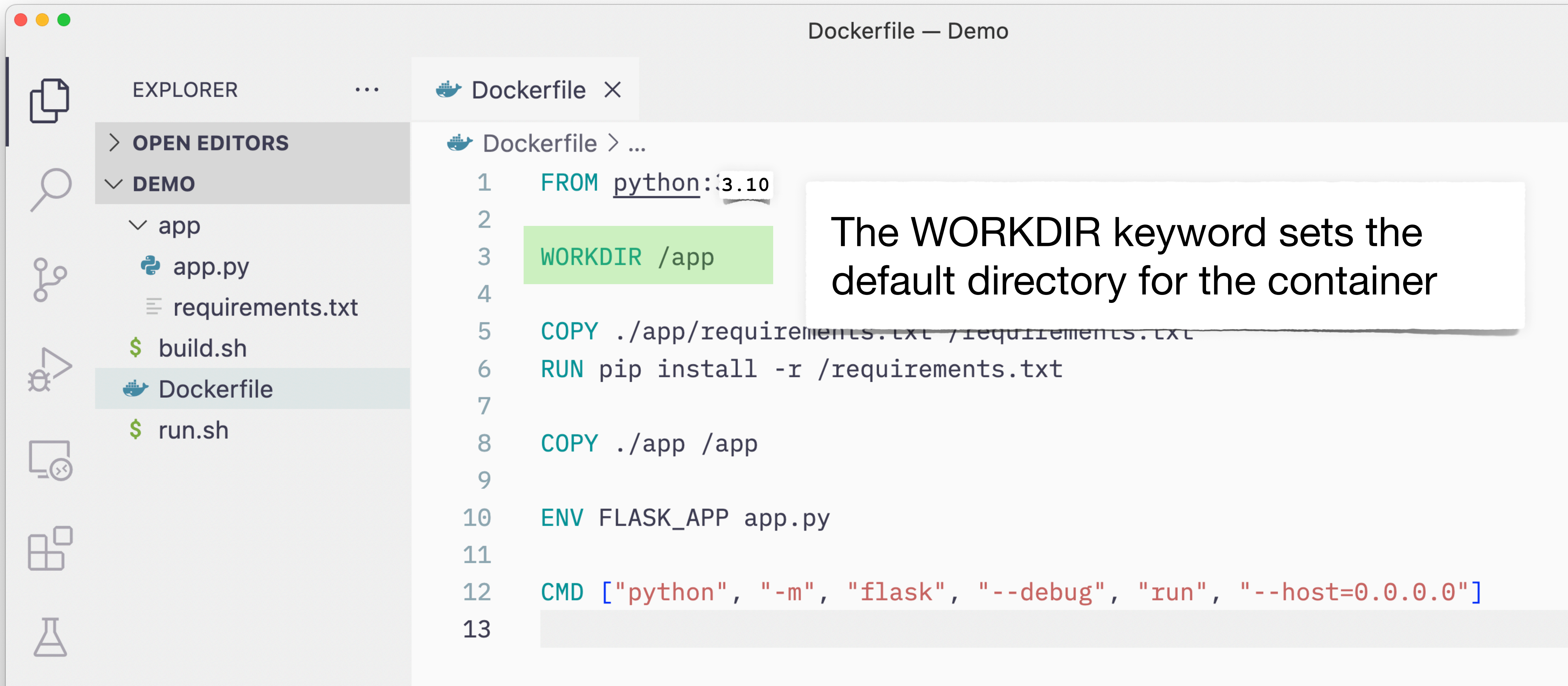
The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a "DEMO" folder containing an "app" subfolder with files "app.py" and "requirements.txt", and other files "build.sh", "Dockerfile", and "run.sh". The "Dockerfile" is selected and open in the editor. The code in the editor 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 on the right side of the editor highlights the first line of code, `FROM python:3.10`, with the text: "The FROM keyword specifies which base image we're building on top of."

Creating Images

The Dockerfile



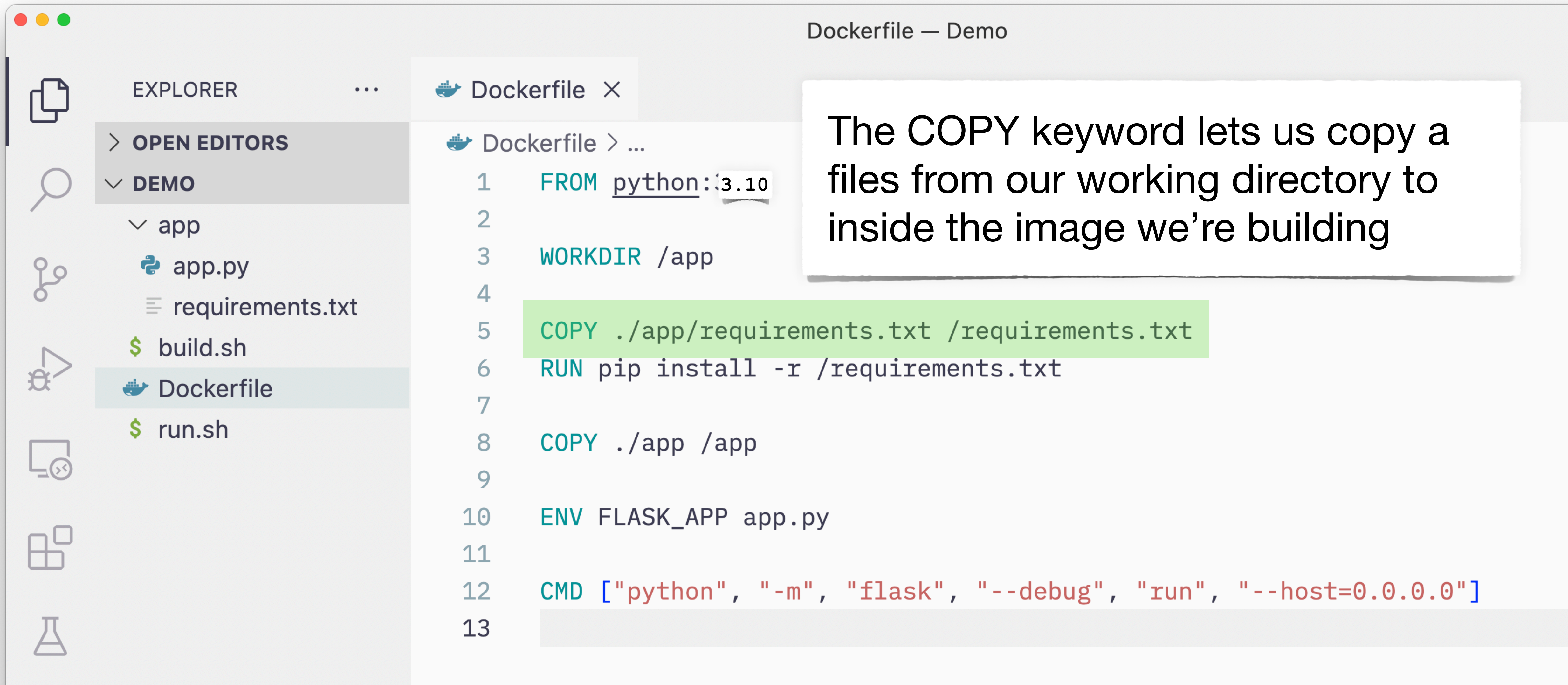
The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a "DEMO" folder containing "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The main editor area displays the content of the Dockerfile:

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

A callout box highlights the `WORKDIR /app` line, stating: "The WORKDIR keyword sets the default directory for the container".

Creating Images

The Dockerfile



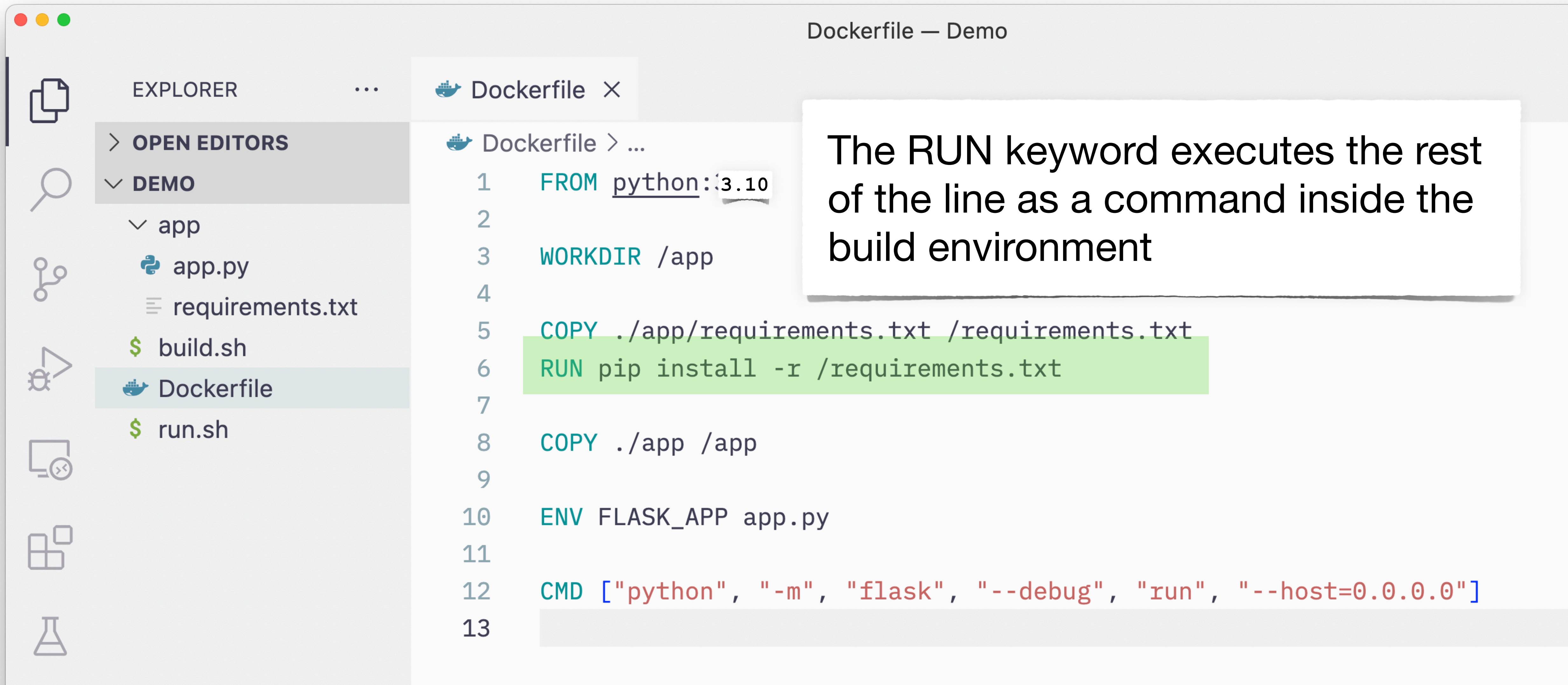
The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a "DEMO" folder containing "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The main editor area displays the content of the Dockerfile:

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

A callout box on the right contains the text: "The COPY keyword lets us copy a files from our working directory to inside the image we're building". The line `COPY ./app/requirements.txt /requirements.txt` in the Dockerfile is highlighted in green.

Creating Images

The Dockerfile

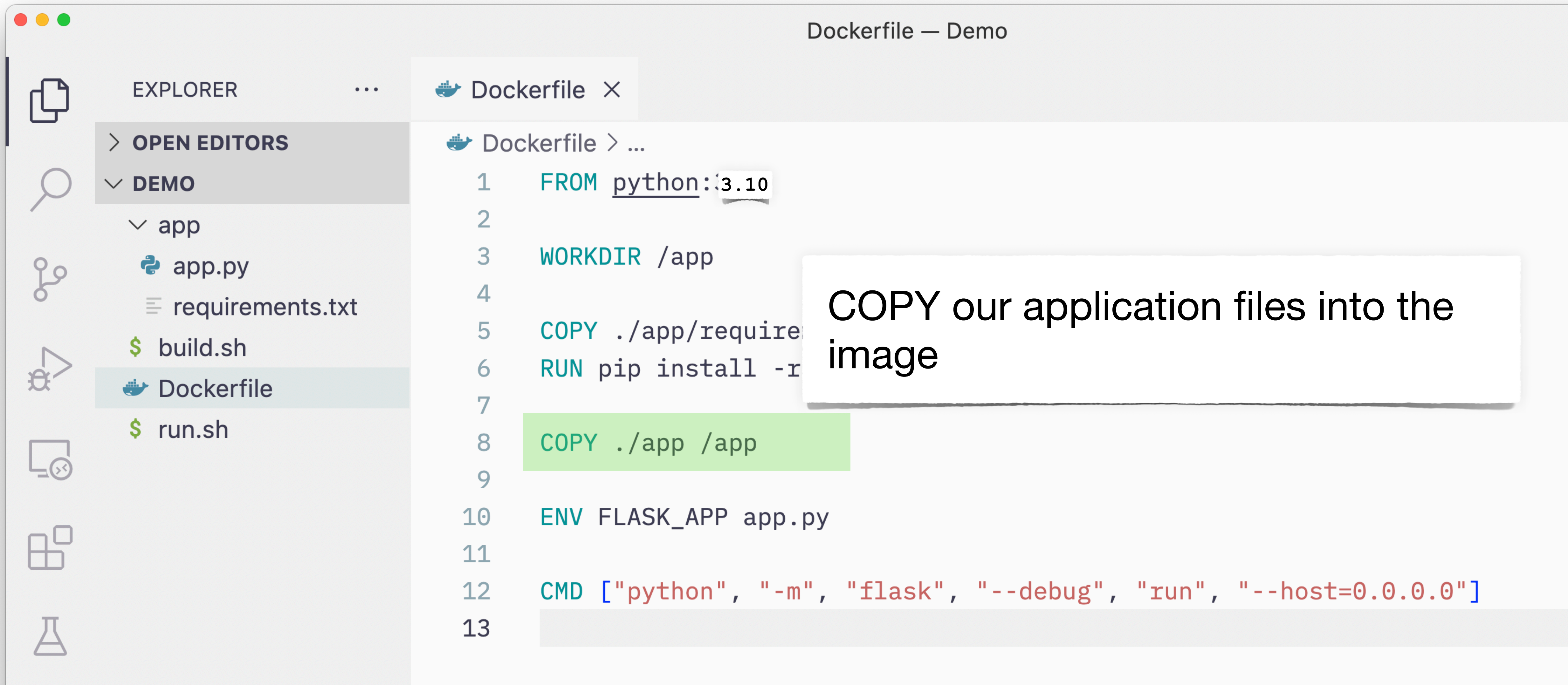


The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a folder named "DEMO" containing files like "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The main editor area displays the content of the "Dockerfile" with line numbers 1 through 13. A white callout box with a drop shadow is positioned over the editor, containing the text: "The RUN keyword executes the rest of the line as a command inside the build environment". The text in the callout box is black. 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
```


Creating Images

The Dockerfile



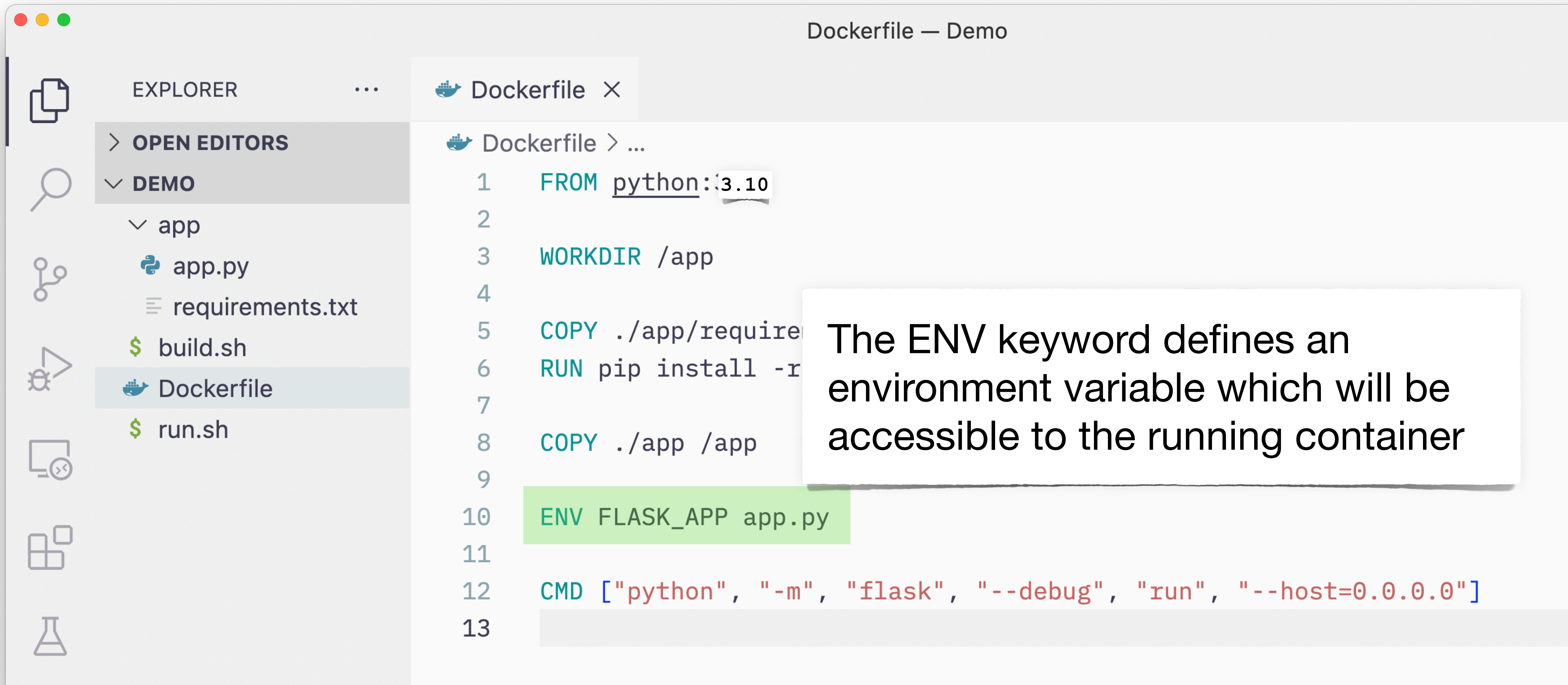
The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a "DEMO" folder containing an "app" subfolder with files "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The main editor area shows the content of the "Dockerfile" file, which includes the following lines:

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /app
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 with a white background and a dark border is positioned over the editor, containing the text: "COPY our application files into the image". The line "COPY ./app /app" in the code is highlighted with a light green background.

Creating Images

The Dockerfile



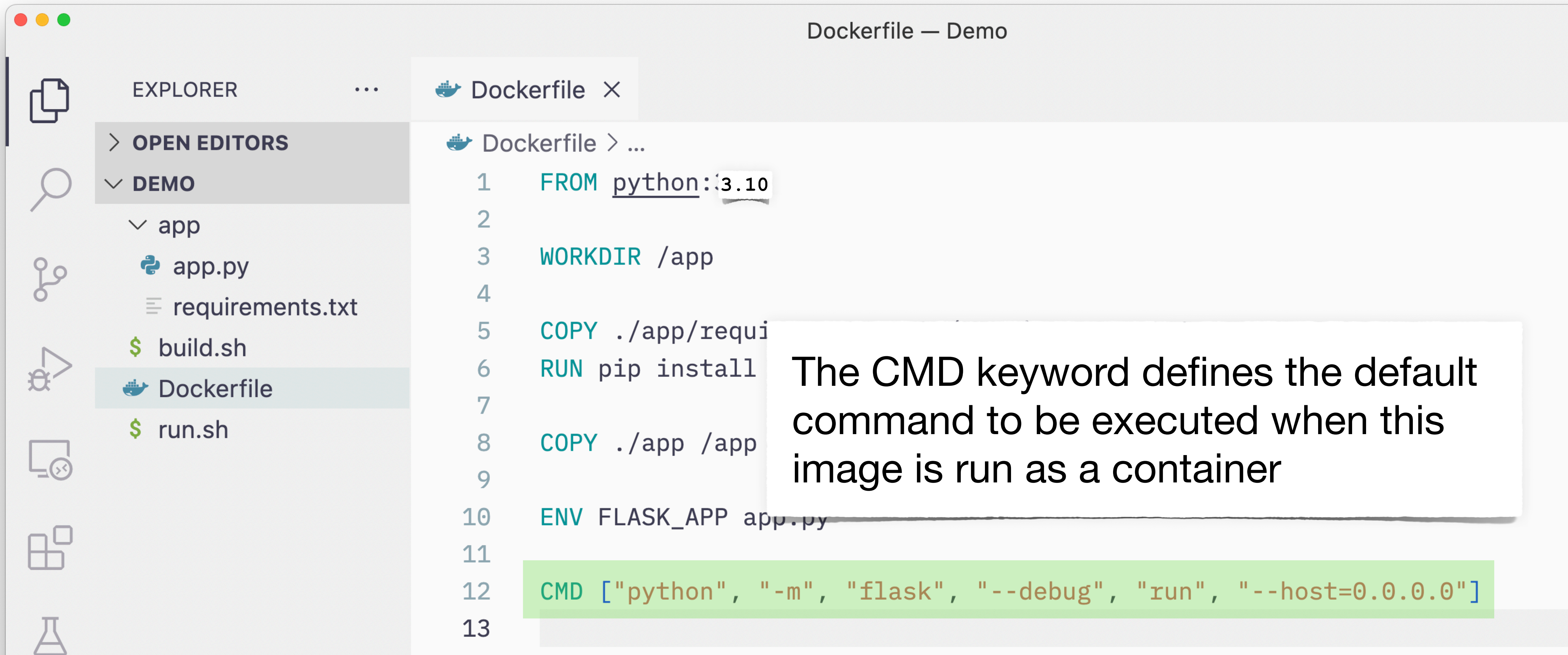
The image shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a folder named "DEMO" containing subfolders "app" and "build.sh", and files "app.py", "requirements.txt", "Dockerfile", and "run.sh". The "Dockerfile" file is selected and open in the editor. The code in the Dockerfile is as follows:

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /app
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 on the right side of the editor contains the text: "The ENV keyword defines an environment variable which will be accessible to the running container". The line "ENV FLASK_APP app.py" in the code is highlighted with a green background.

Creating Images

The Dockerfile



The screenshot shows a code editor window titled "Dockerfile — Demo". On the left, the Explorer sidebar shows a project structure with a folder named "DEMO" containing files like "app.py", "requirements.txt", "build.sh", "Dockerfile", and "run.sh". The "Dockerfile" file is selected and open in the editor. The code in the Dockerfile is as follows:

```
1 FROM python:3.10
2
3 WORKDIR /app
4
5 COPY ./app/requirements.txt /app
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 line `CMD ["python", "-m", "flask", "--debug", "run", "--host=0.0.0.0"]` is highlighted in green. A white text box with a black border is overlaid on the right side of the editor, containing the text: "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]
```

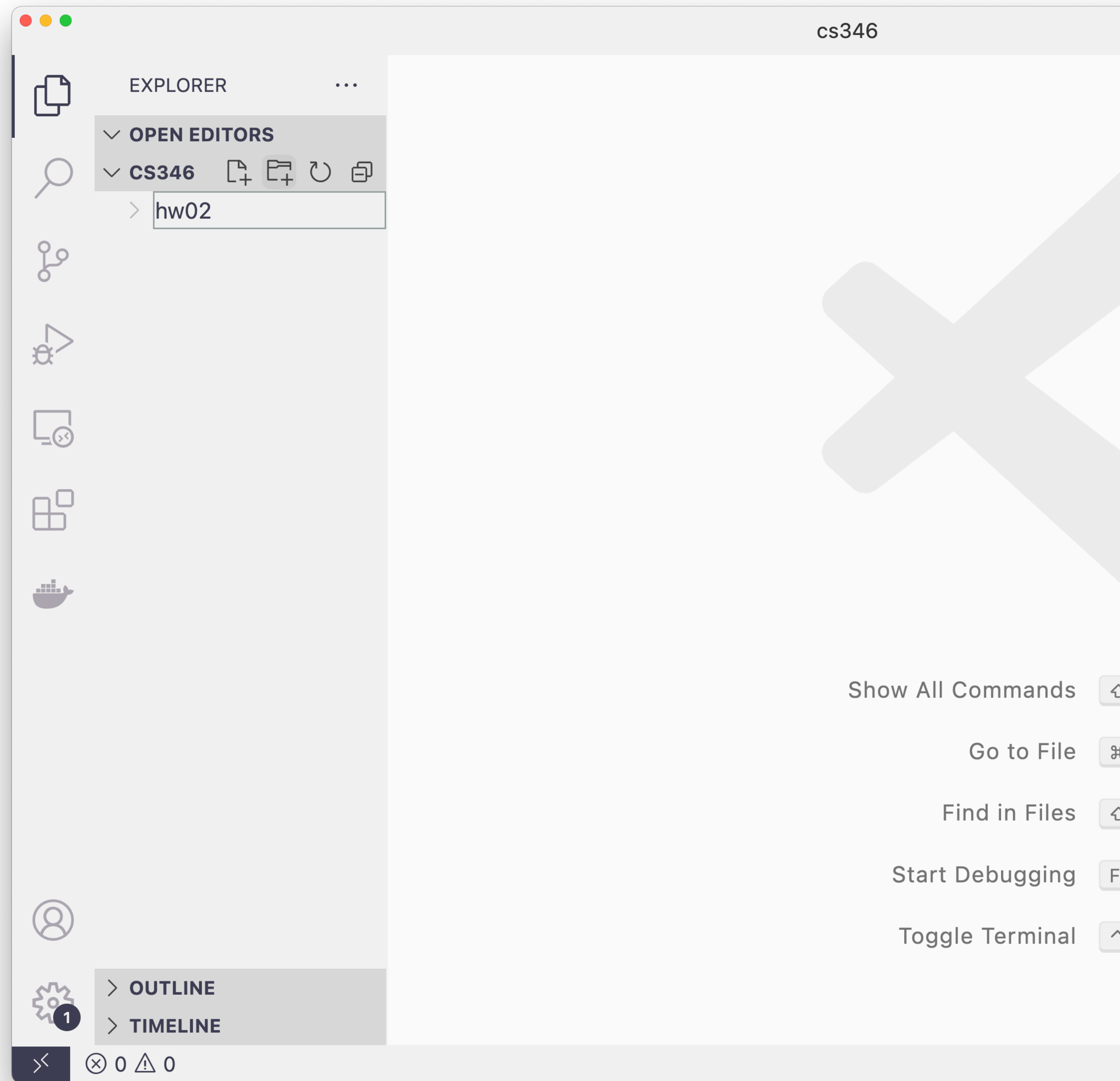
```
docker build --tag my_app:latest .
```

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

Creating Images

Getting Started Basics

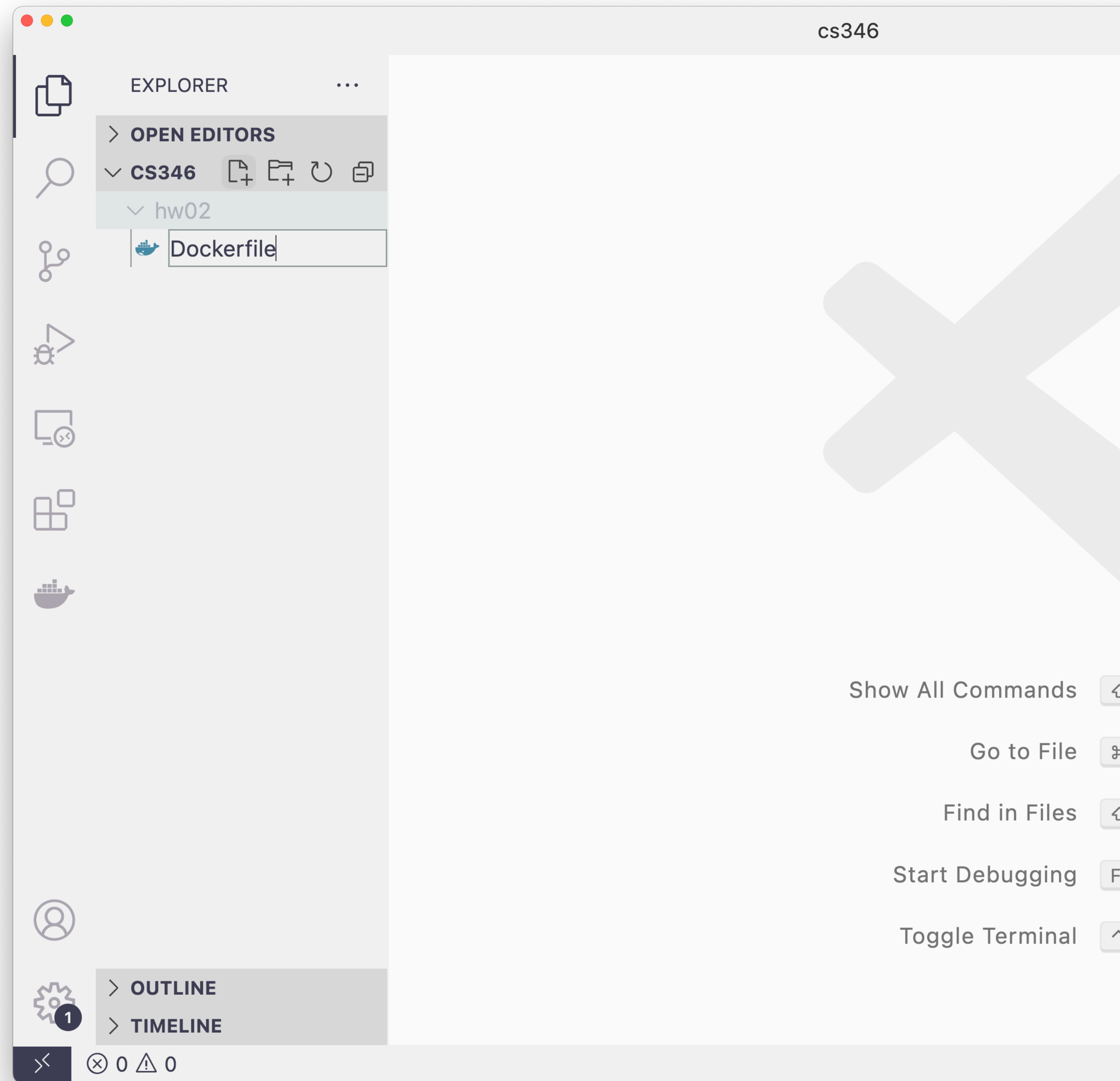
- When working on any project, such as an application or homework assignment, the first step is often to create a new directory to hold all the stuff relating to the project.
- So to start with, figure out where on your laptop you want to keep all your work for this class, and make a new folder in there.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



Creating Images

Getting Started Basics

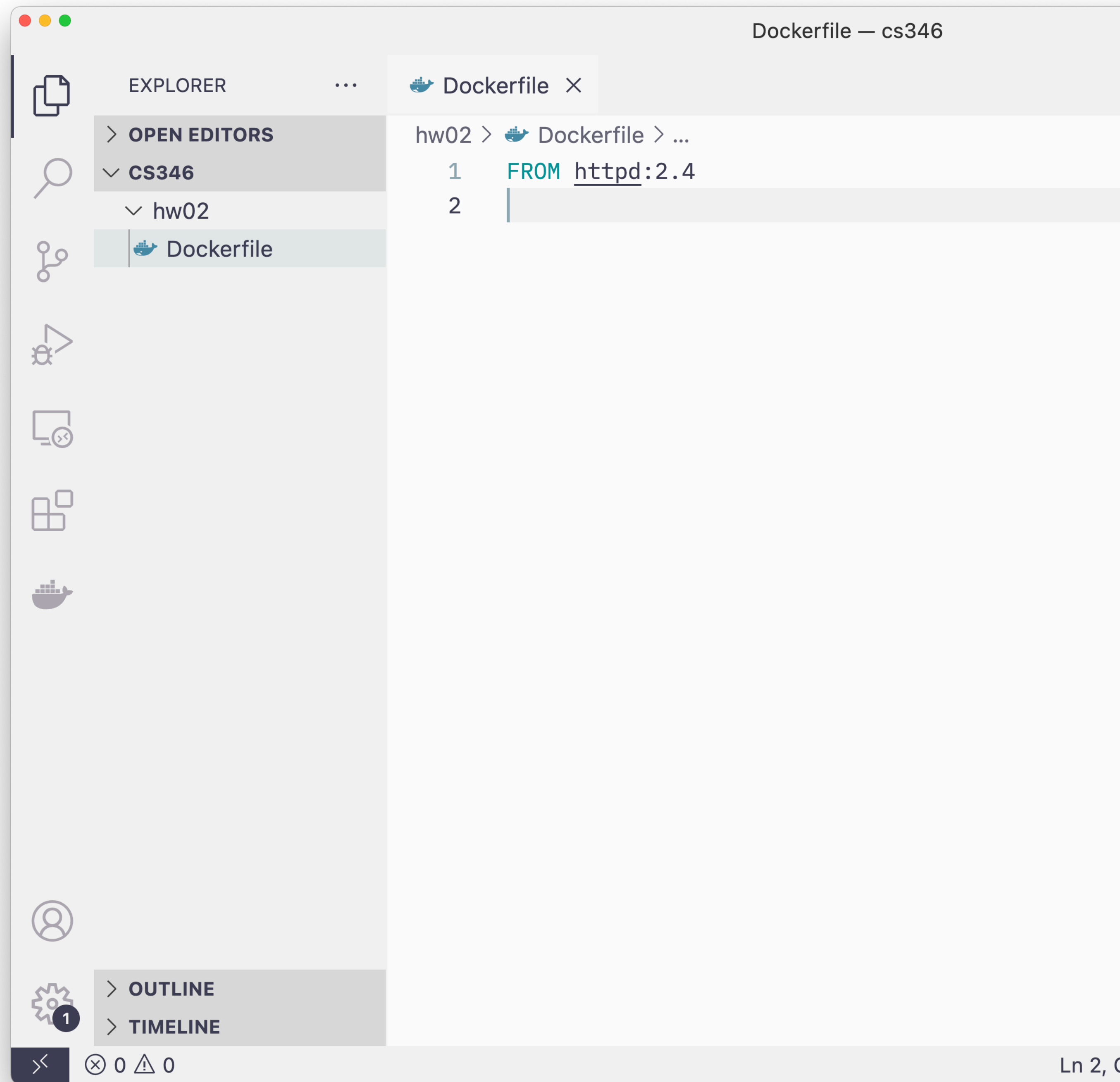
- Next we need to create a new empty text file inside our project folder, and name it **Dockerfile**



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.

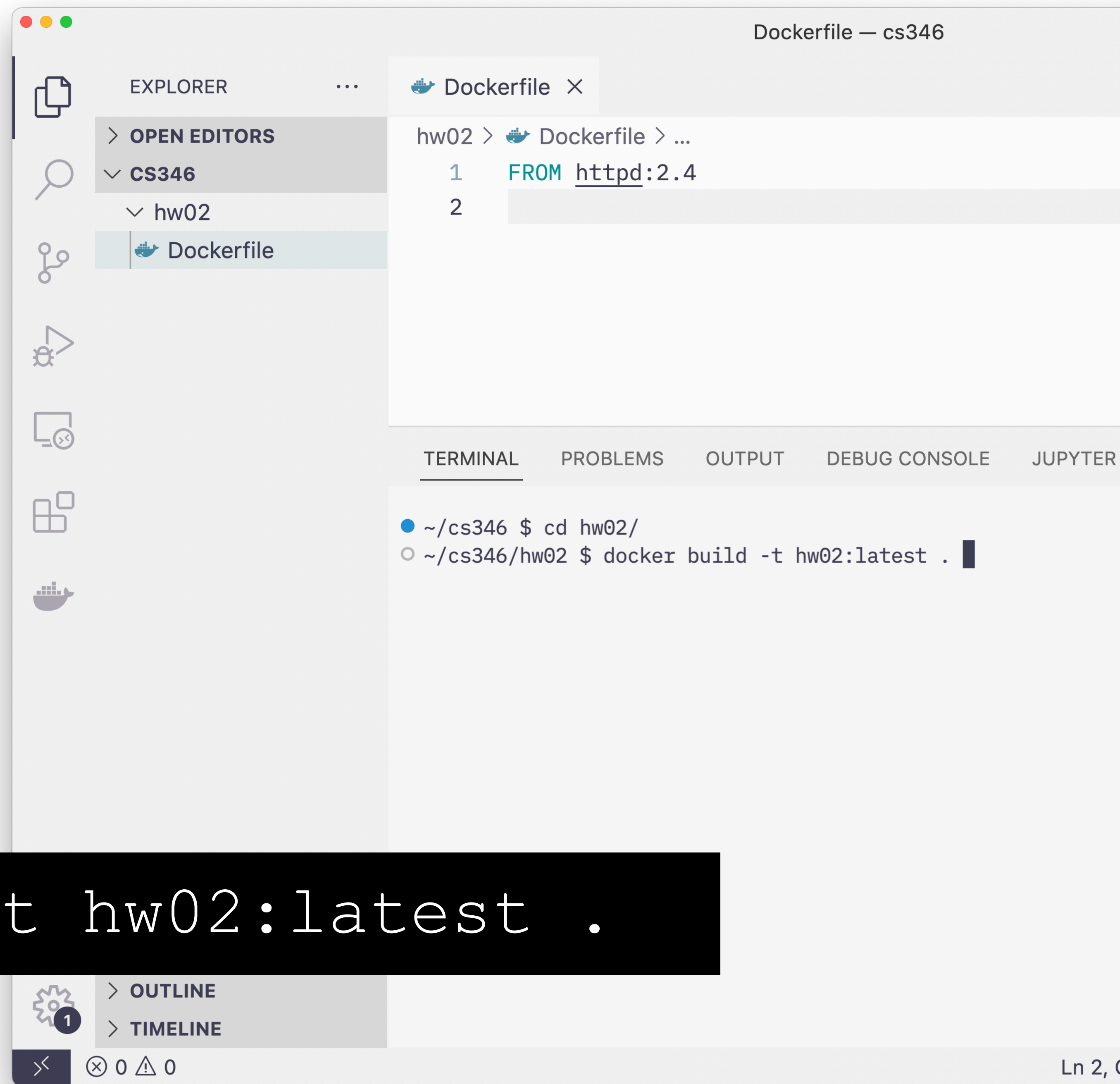


Creating Images

Getting Started Basics

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

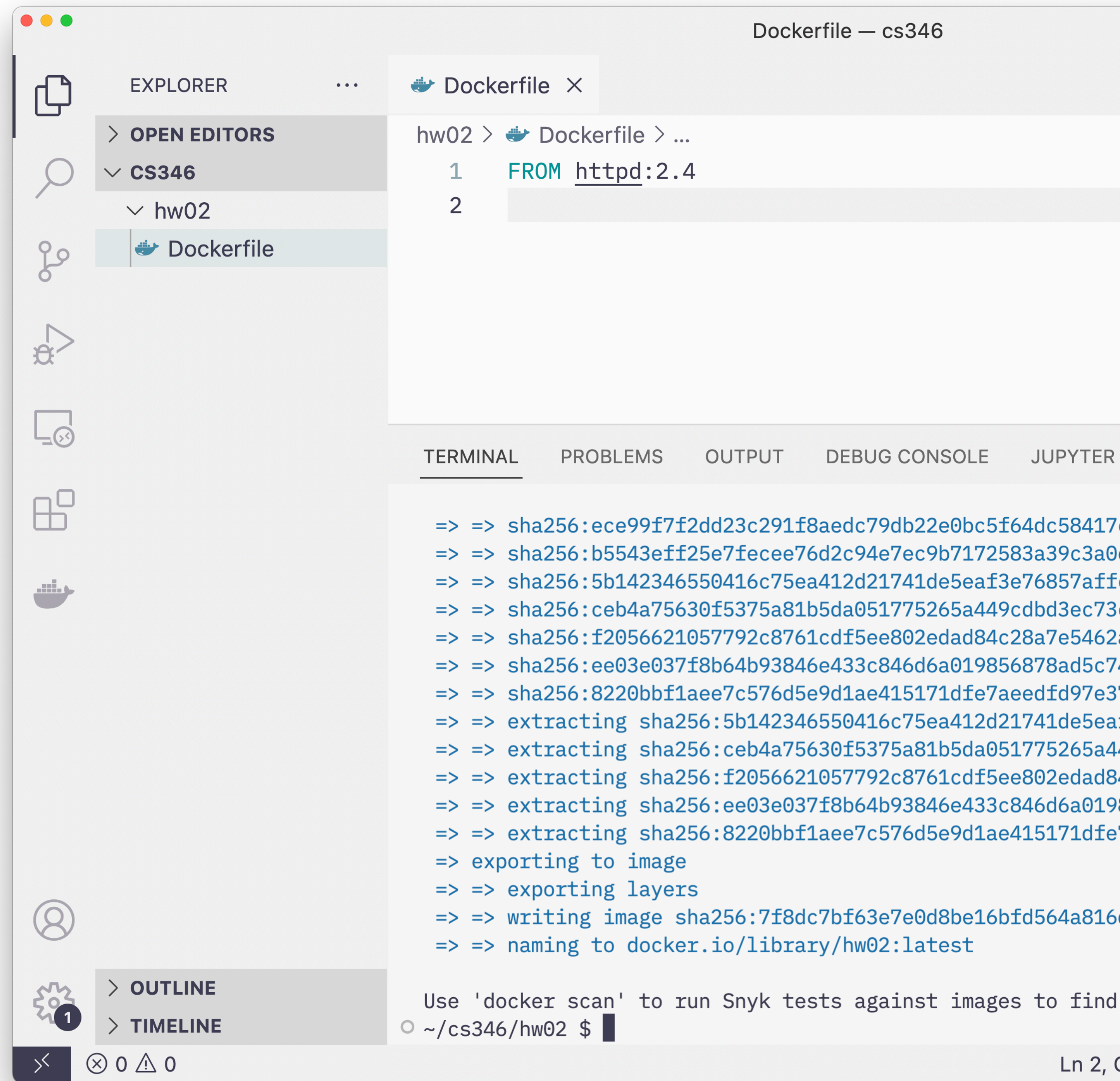
```
docker build -t hw02:latest .
```



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.



```
Dockerfile — cs346
```

EXPLORER

- > OPEN EDITORS
- ✓ CS346
 - ✓ hw02
 - Dockerfile

OUTLINE

- > TIMELINE

Dockerfile

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

TERMINAL

```
=> => sha256:ece99f7f2dd23c291f8aedc79db22e0bc5f64dc58417
=> => sha256:b5543eff25e7fecee76d2c94e7ec9b7172583a39c3a0
=> => sha256:5b142346550416c75ea412d21741de5eaf3e76857aff
=> => sha256:ceb4a75630f5375a81b5da051775265a449cddb3ec73
=> => sha256:f2056621057792c8761cdf5ee802edad84c28a7e5462
=> => sha256:ee03e037f8b64b93846e433c846d6a019856878ad5c7
=> => sha256:8220bbf1aee7c576d5e9d1ae415171dfe7aeedfd97e3
=> => extracting sha256:5b142346550416c75ea412d21741de5ea
=> => extracting sha256:ceb4a75630f5375a81b5da051775265a4
=> => extracting sha256:f2056621057792c8761cdf5ee802edad8
=> => extracting sha256:ee03e037f8b64b93846e433c846d6a019
=> => extracting sha256:8220bbf1aee7c576d5e9d1ae415171dfe
=> exporting to image
=> => exporting layers
=> => writing image sha256:7f8dc7bf63e7e0d8be16bfd564a816
=> => naming to docker.io/library/hw02:latest

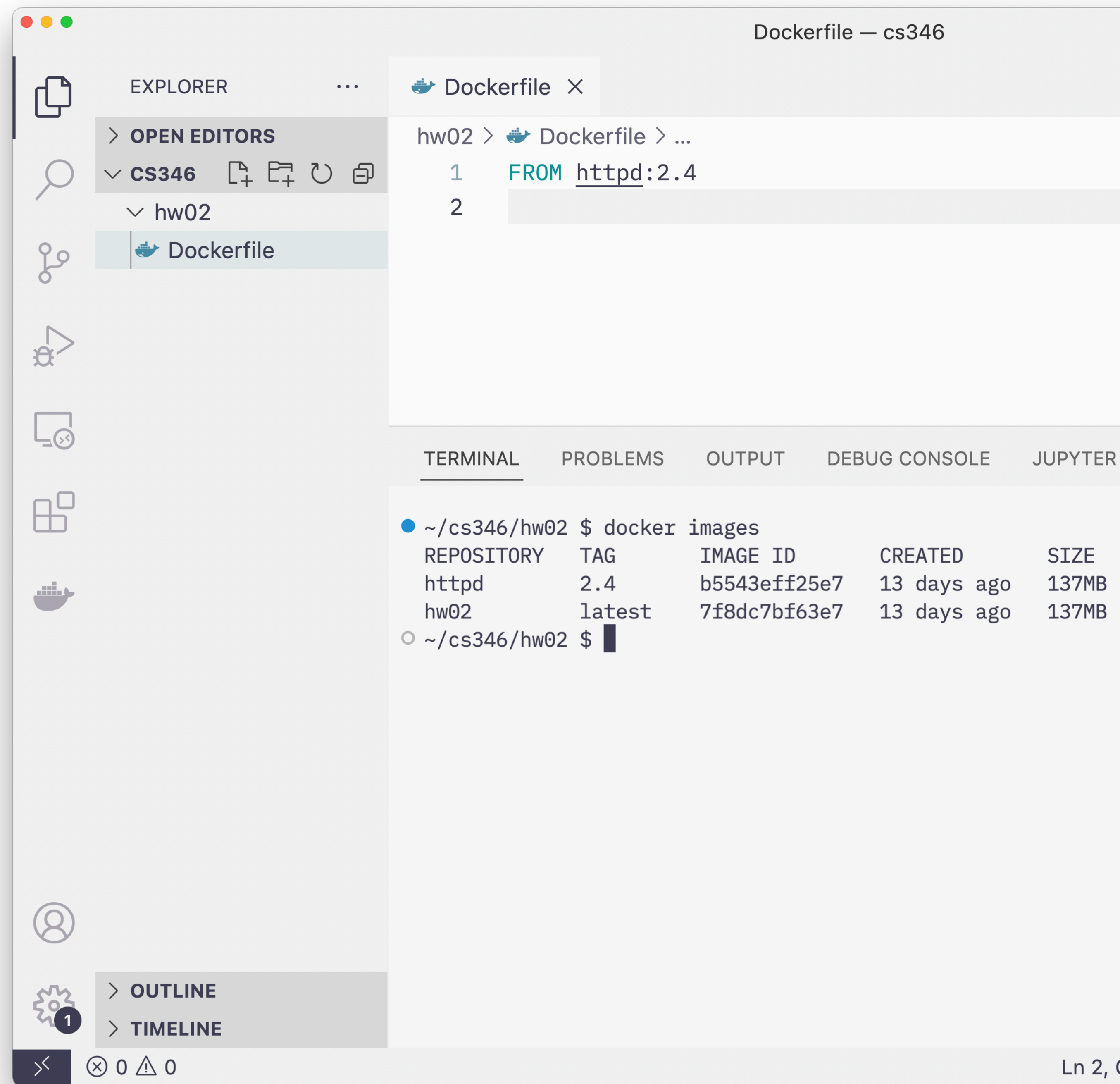
Use 'docker scan' to run Snyk tests against images to find
~/cs346/hw02 $
```

Ln 2, C

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 Visual Studio Code interface. The Explorer view on the left shows a project named 'CS346' with a subfolder 'hw02' containing a 'Dockerfile'. The Dockerfile content is visible in the editor:

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

The Terminal view at the bottom shows the output of the `docker images` command:

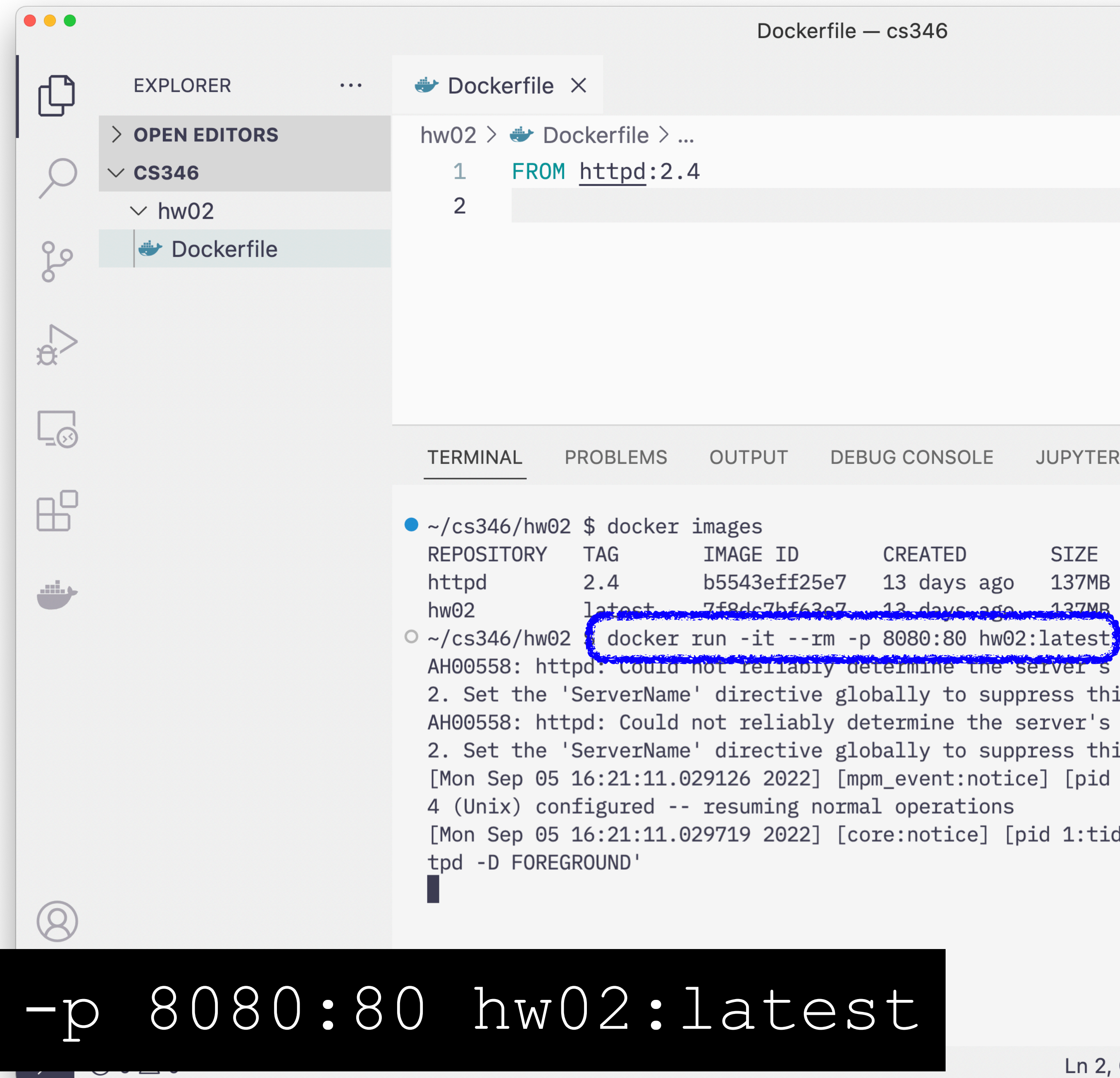
```
~/cs346/hw02 $ docker images  
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE  
httpd         2.4      b5543eff25e7  13 days ago   137MB  
hw02         latest   7f8dc7bf63e7  13 days ago   137MB  
~/cs346/hw02 $
```

The status bar at the bottom right indicates the current line is 'Ln 2, C'.

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 a VS Code editor window titled "Dockerfile — cs346". The Explorer sidebar on the left shows the project structure: "EXPLORER" > "OPEN EDITORS" > "CS346" > "hw02" > "Dockerfile". The Dockerfile content is as follows:

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

The Terminal panel at the bottom shows the following 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
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'
```

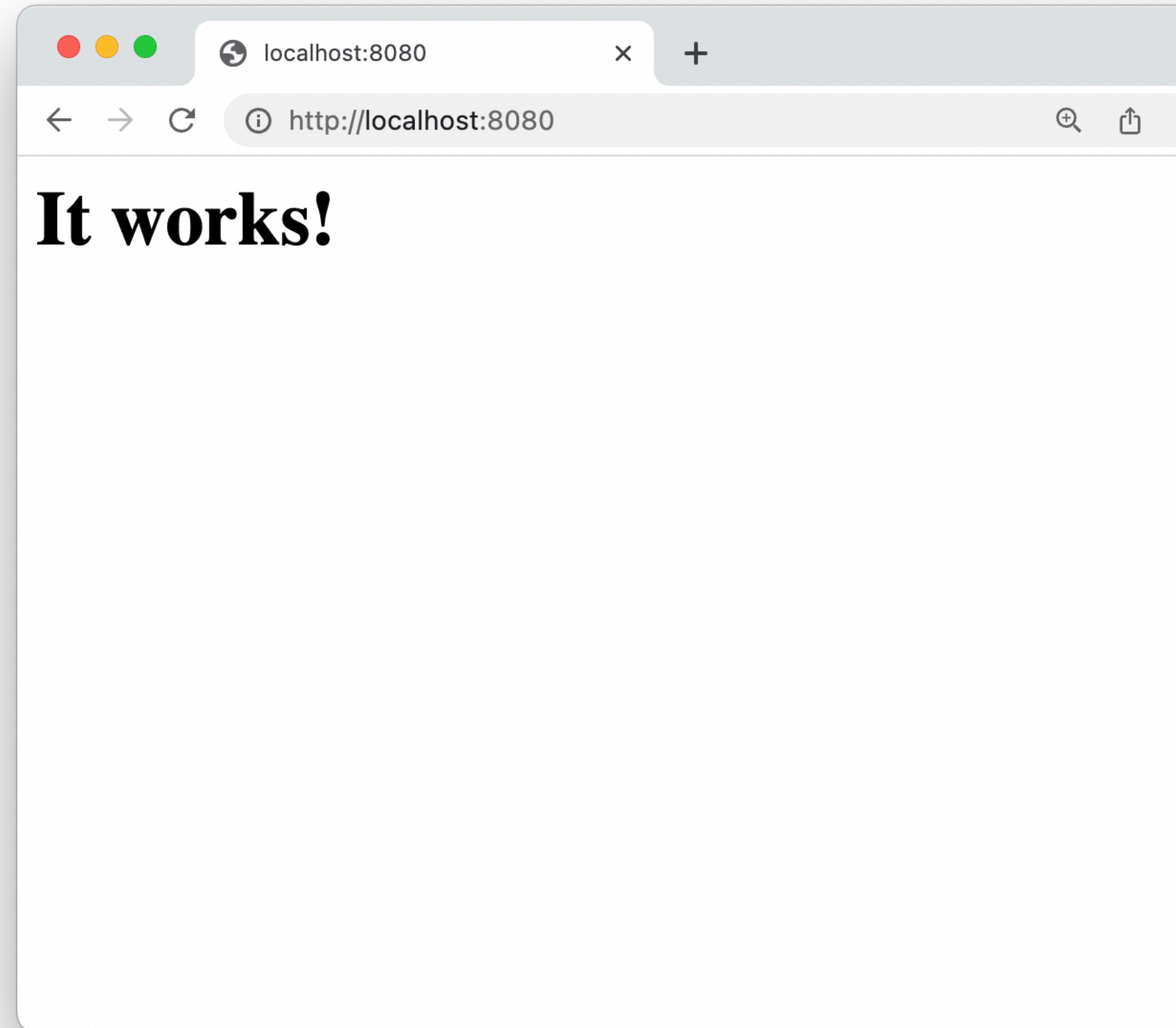
The command `docker run -it --rm -p 8080:80 hw02:latest` is highlighted in blue in the terminal output.

```
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

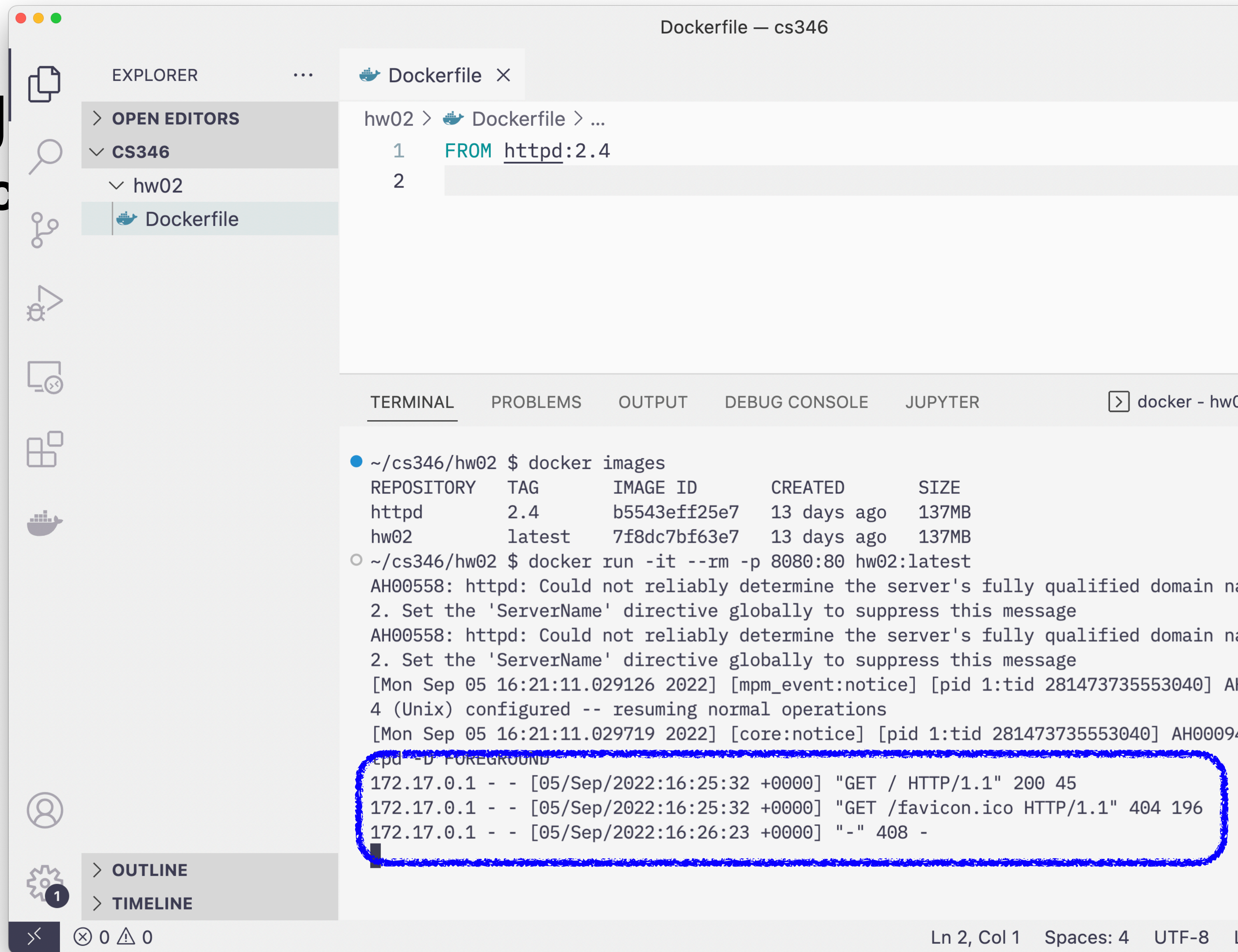
Getting Started Basics

- 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 Image

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 Visual Studio Code editor window titled "Dockerfile — cs346". The Explorer sidebar on the left shows a project structure with folders "OPEN EDITORS", "CS346", and "hw02", and a file "Dockerfile" under "hw02". The Dockerfile content is as follows:

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

The Terminal window at the bottom shows the following 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:
4 (Unix) configured -- resuming normal operations
[Mon Sep 05 16:21:11.029719 2022] [core:notice] [pid 1:tid 281473735553040] AH00094:
epd -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 -
```

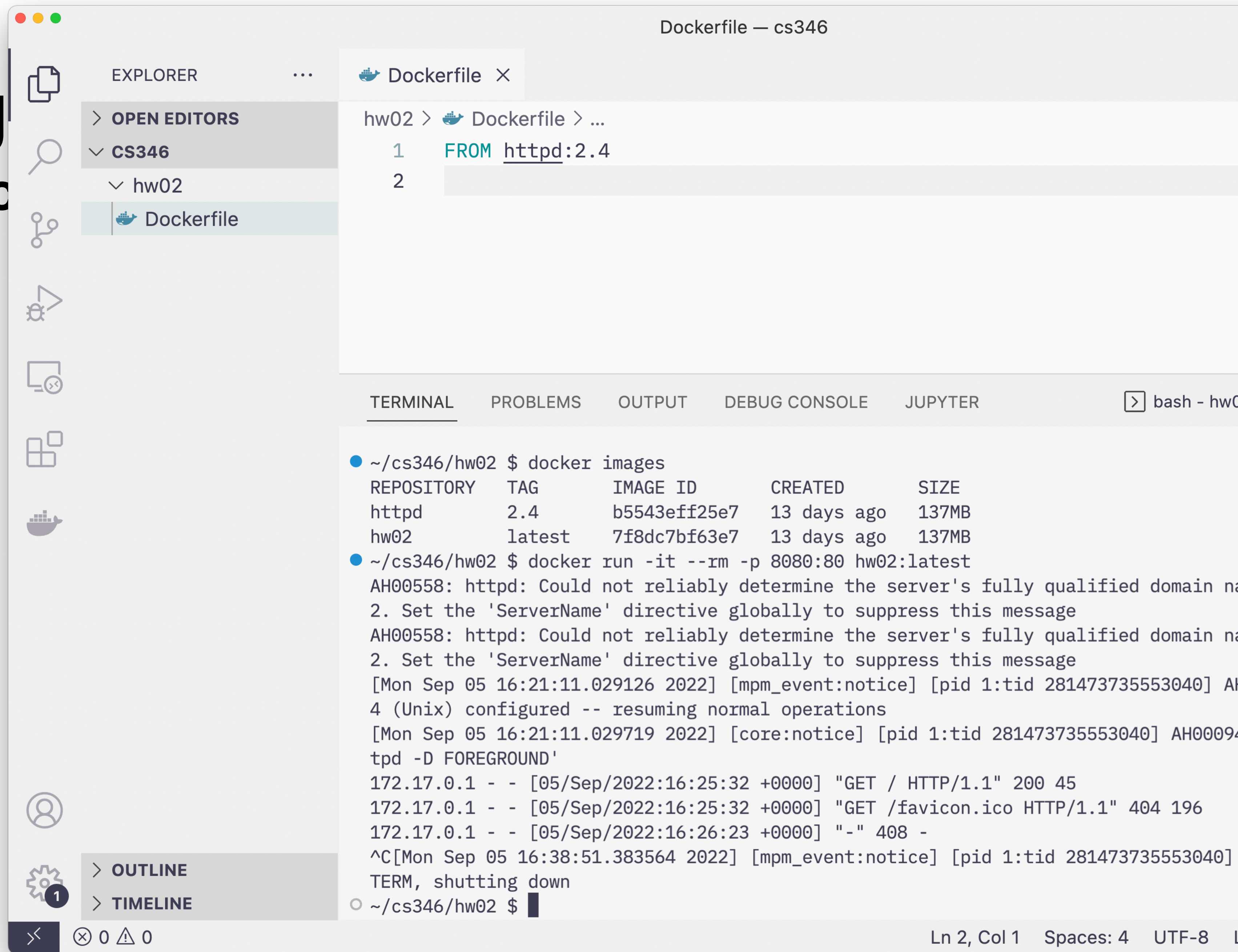
The last three lines of the terminal output are highlighted with a blue box.

At the bottom of the VS Code window, the status bar shows "Ln 2, Col 1 Spaces: 4 UTF-8".

Creating Image

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 Visual Studio Code editor window titled "Dockerfile — cs346". The Explorer sidebar on the left shows a project structure with folders "CS346" and "hw02", and a file "Dockerfile" under "hw02". The Dockerfile content is as follows:

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

The Terminal panel at the bottom shows the following 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:
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 status bar at the bottom right indicates "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]
```

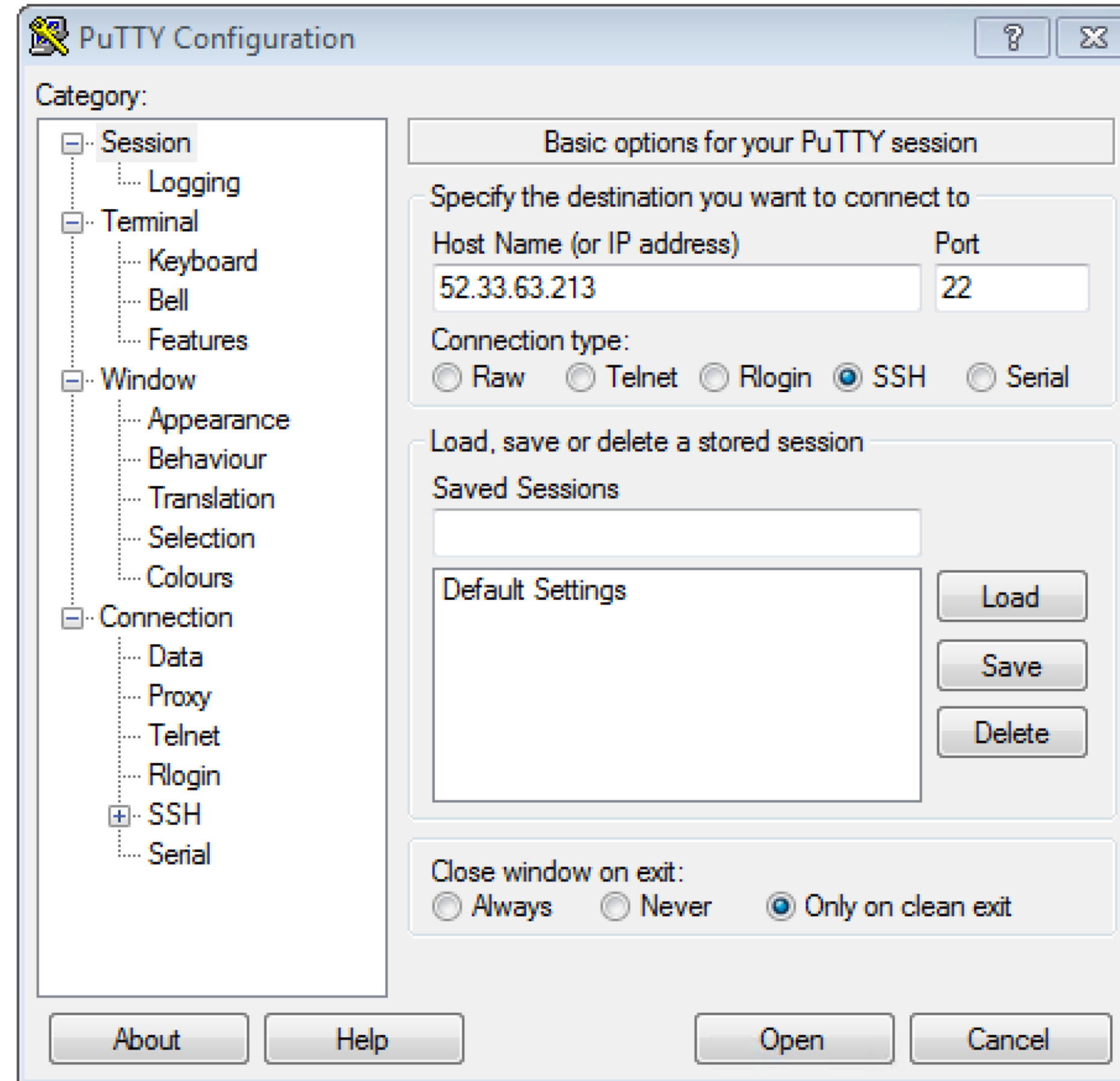
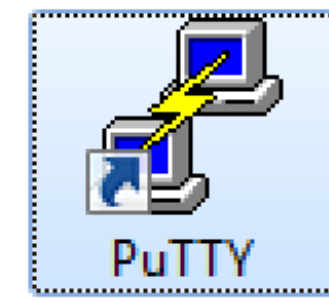
Connecting to Remote Hosts

`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

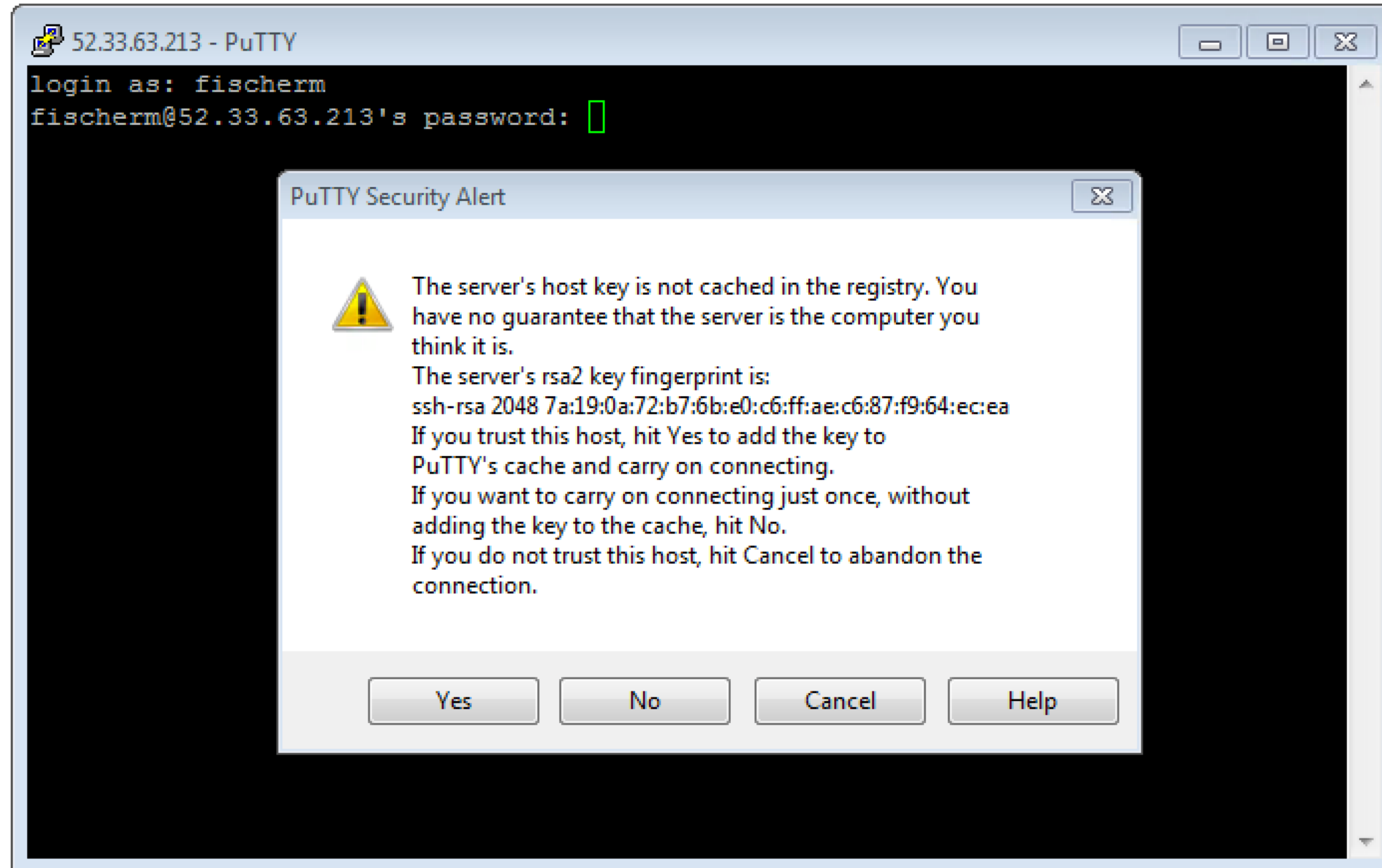
Putty



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

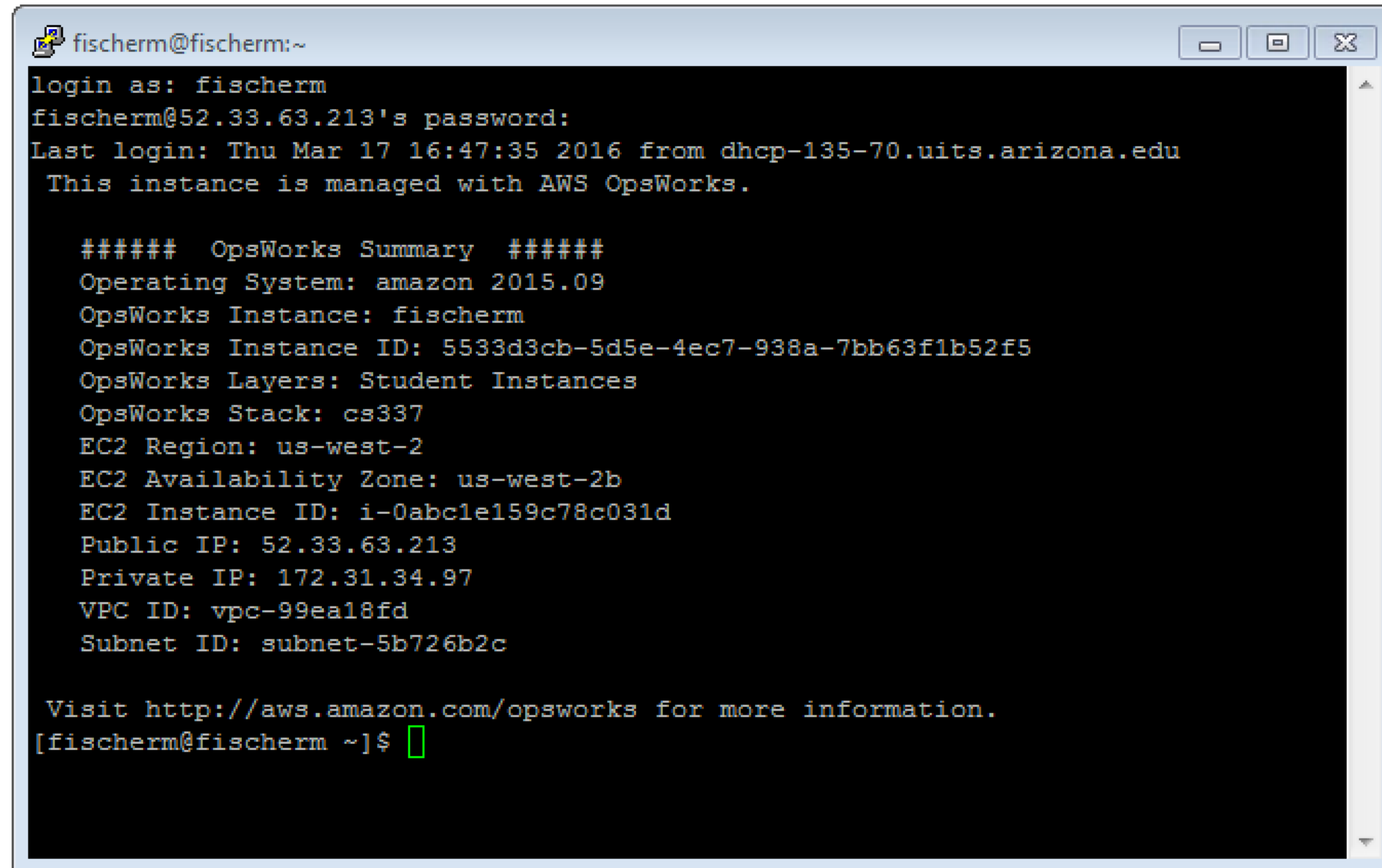
Connecting to Remote Hosts

Putty



Connecting to Remote Hosts

Putty



```
fischem@fischem:~  
login as: fischem  
fischem@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: fischem  
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.  
[fischem@fischem ~]$
```



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER

bash + v [window icon] [trash icon] [up arrow] [close icon]

```

○ root@6e78993d35d3:~/work# ssh fischerm@lectura.cs.arizona.edu
The authenticity of host 'lectura.cs.arizona.edu (192.12.69.186)' can't be established.
ECDSA key fingerprint is SHA256:eehHz6aUyHjai4kre0ZINfCAZXj+JhgAByyREZE9ZGg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'lectura.cs.arizona.edu,192.12.69.186' (ECDSA) to the list of known hosts.
fischerm@lectura.cs.arizona.edu's password:

```

System information as of Sat 27 Aug 2022 04:17:45 PM MST

```

System load: 0.0          Processes:                491
Usage of /:  41.1% of 9.78GB  Users logged in:        15
Memory usage: 13%          IPv4 address for ens18: 192.12.69.186
Swap usage:  1%

```

=> There is 1 zombie process.

Welcome to:

```

  _
 | |  _ _ _ _ _ | |  _ _ _ _ _
 | | / _ \ _ _ | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
 | |  _ _ _ _ _ | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |

```



Lectura

Shared Computer Science Host

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

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

