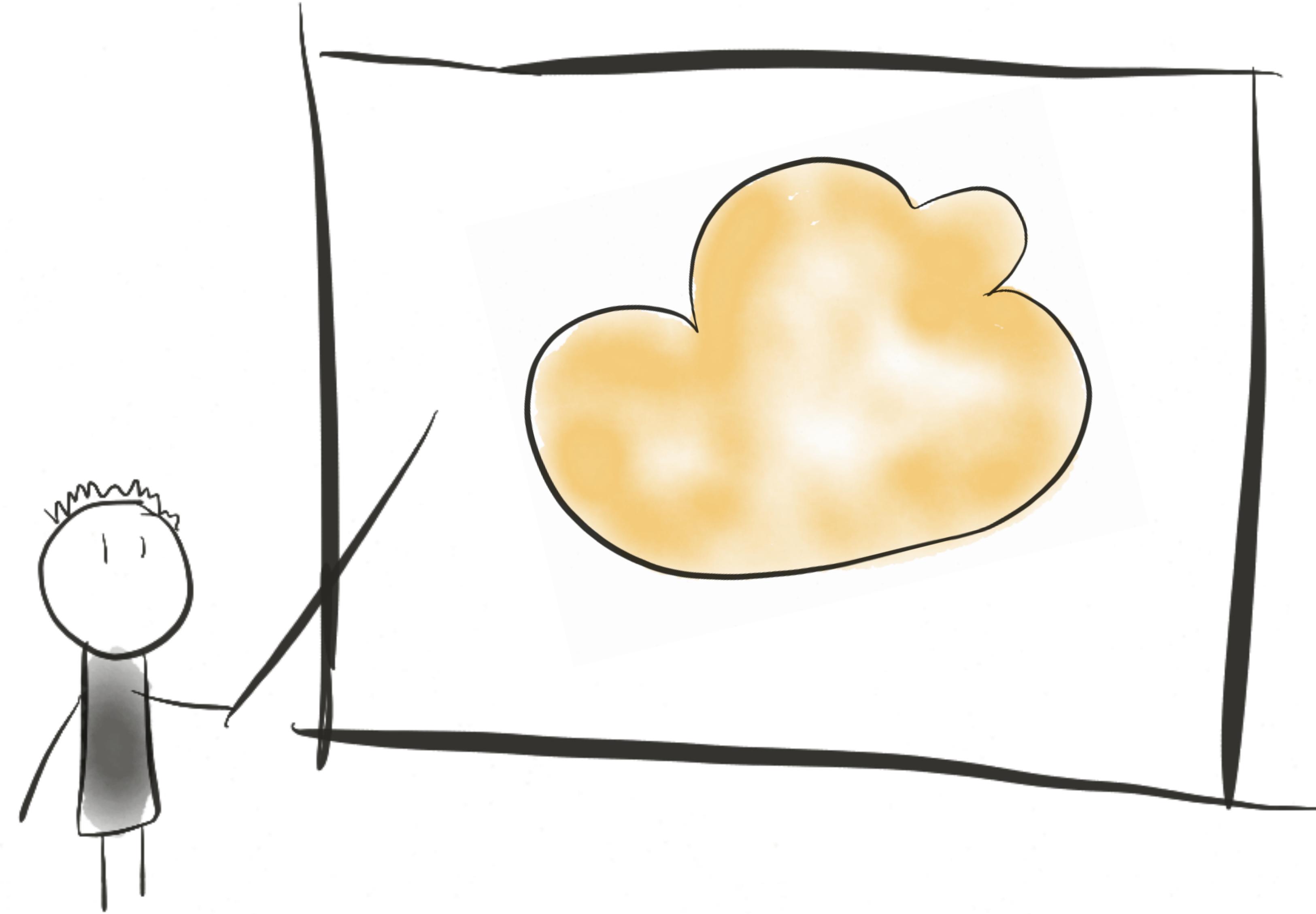


CSC 346 - Cloud Computing

01: What is the Cloud? An introduction to Docker

The Cloud

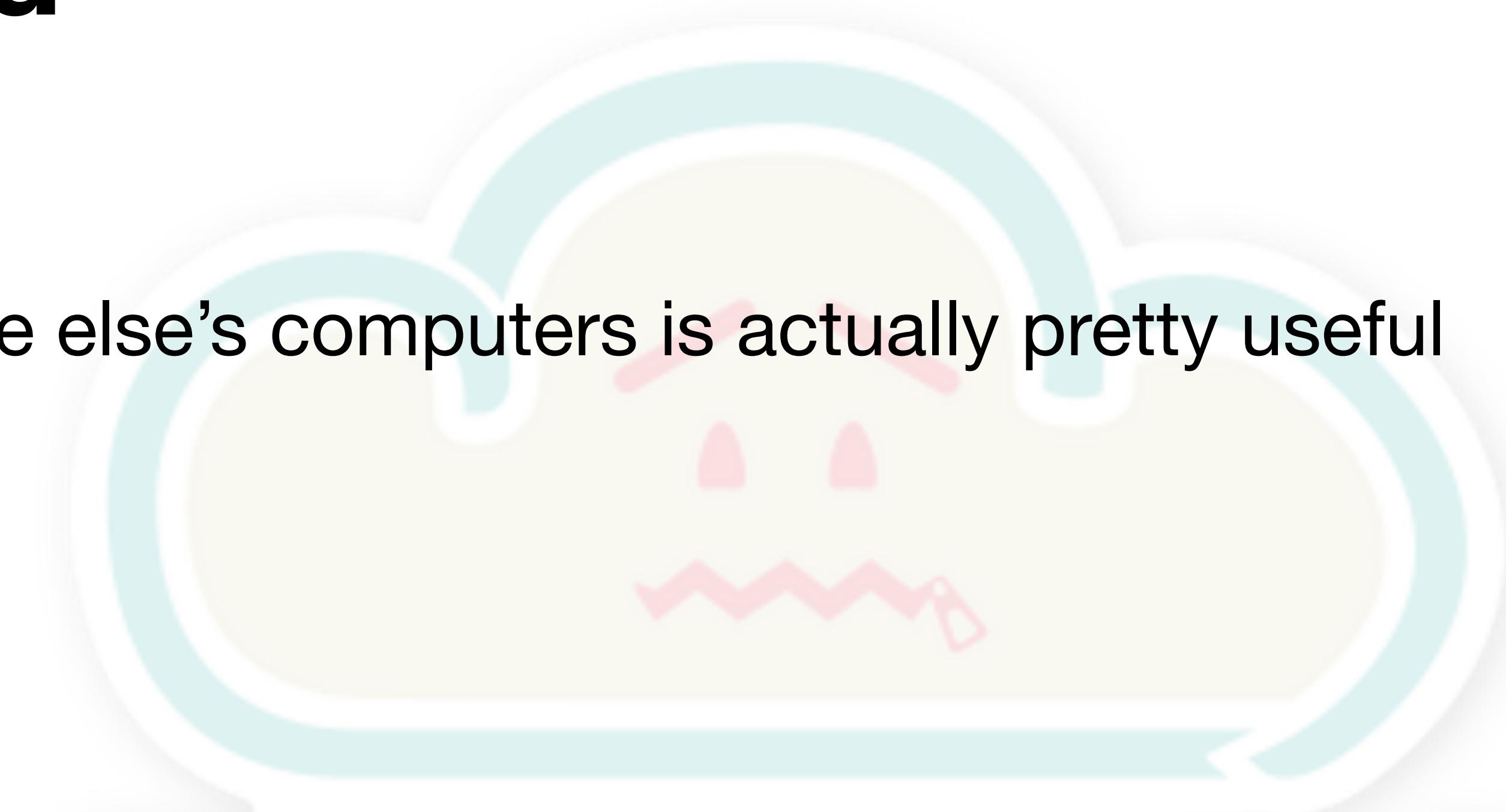




There is no cloud
it's just someone else's computer

The Cloud

- Using someone else's computers is actually pretty useful



There is no cloud

it's just someone else's computer

The Data Center

- Not too long ago, pretty much all big applications ran on physical servers in data centers that the Company or University controlled.



The Data Center

- Our Apps pretty much used to be installed on specific physical servers.
- If it was a big app, maybe it was distributed across several physical servers.



The Data Center

- Required a lot of guessing about the future
 - How much memory?
 - How many CPUs?
 - 1 server? 10 servers?
 - If I need more, how long will it take to order them, ship them, rack them, install the app...
 - If I bought too much, what then?



The Data Center

- Also problematic for smaller applications
 - Don't need a whole server for some smaller apps or sites
 - Can put multiple applications on the same server
 - Difficulties with cross dependencies



The Data Center - Virtual Machines

- Virtual Machines allowed one massive server to host many smaller virtual machines.
 - This solved a lot of problems
 - Isolated applications
 - VMs can be sized to meet the application needs
 - Some overhead for growth



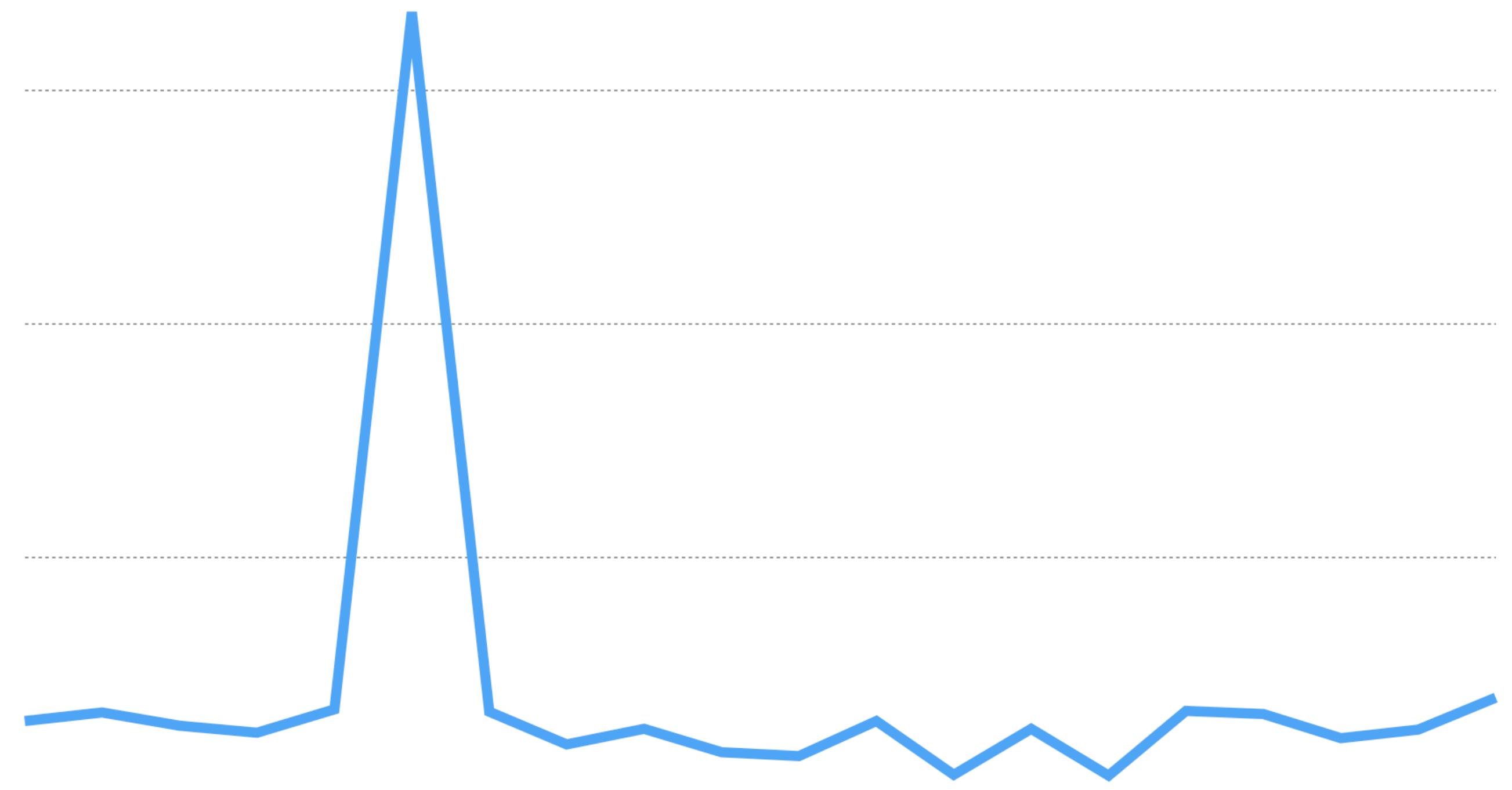
The Data Center - Virtual Machines

- Still had problems
 - Expensive initial purchase
 - Had to guess about future
 - You could expand later, but might not get identical equipment
 - If needs change drastically you may still be caught short on resources



The Data Center - Virtual Machines

- “Spiky Workloads” are a particular problem for a datacenter
- If your application gets slammed at particular times (say... for priority registration)



The Data Center - Virtual Machines

- You have to have enough resources to meet that peak demand year round
 - That costs a lot of money
 - That excess capacity is “wasted” much of the time
 - VMs help some, as that excess capacity can be used by short lived projects

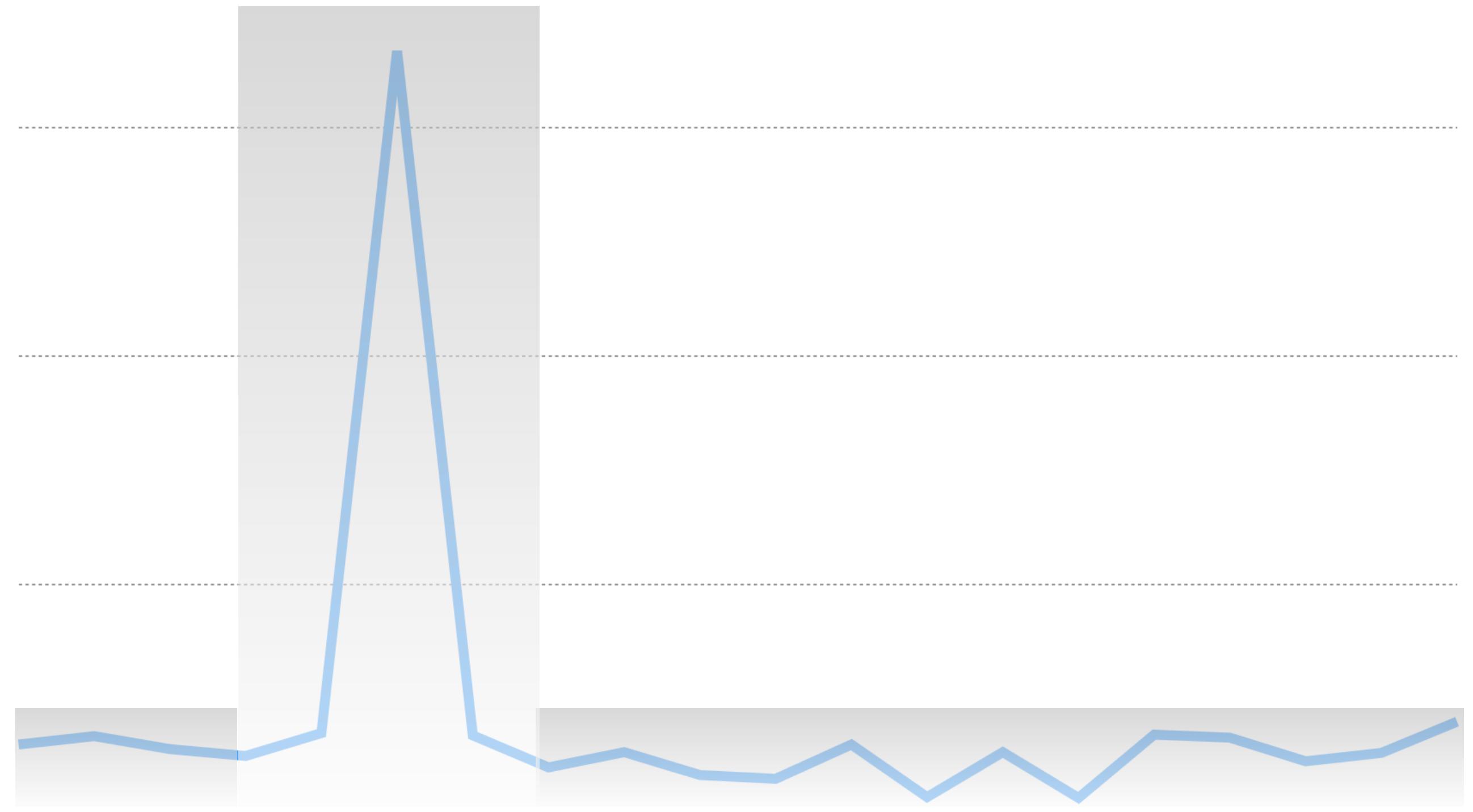


The Cloud

- Using someone else's computers is actually pretty useful
- Instead of purchasing physical hardware, you rent it from someone else
- Costs move from large periodic capital expenses, to smaller monthly operational expenses. (Budget people love this)
 - \$100,000 in year 1 - Lasts for 5 years (hopefully)
 - \$2000 per month for 5 years
 - A bit more expensive over the long term possibly, but you don't need \$100,000 up front

The Cloud

- The biggest advantage of the Cloud is flexibility
 - Instead of paying for peak capacity year round, you can only pay for the 2 week spike
 - So maybe instead of \$2000 a month its only \$500 most months, and \$2,000 that one peak month
 $(\$27,500 + \$10,000 = \$37,500)$



The Cloud

Flexibility enables many different use cases

- Autoscaling - detect when your backend hosts are getting stressed and automatically deploy more backend resources. Get rid of them as the load subsides
- Experiments - deploy additional development environments in parallel to your production environments. Maybe each developer or feature gets dedicated resources
- Try new things faster - you don't have to wait for new CPUs to be delivered to your datacenter. You can try new resources quickly and relatively cheaply.

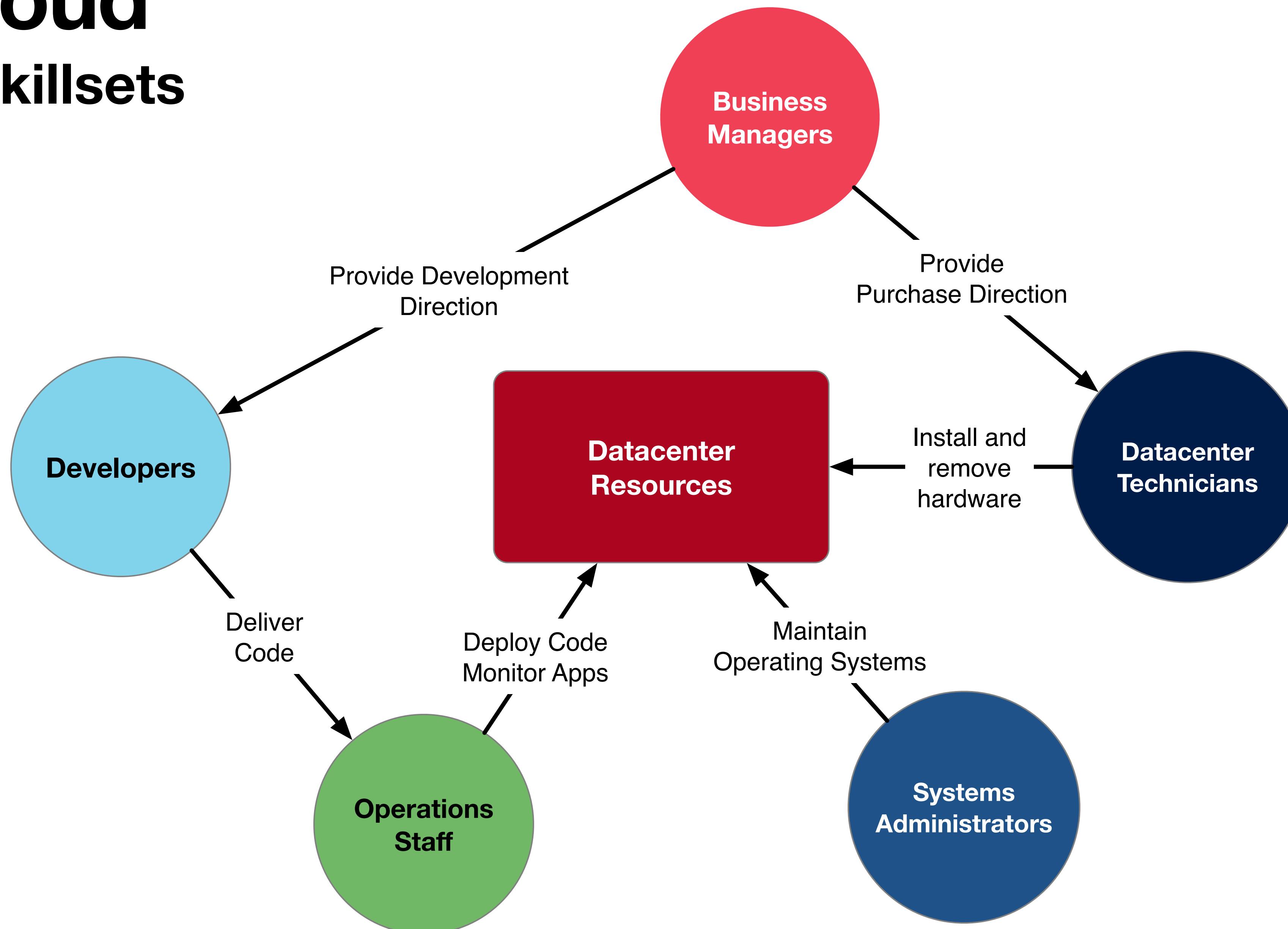
The Cloud

Pay Per Hour vs Pay Per Request

- Cloud vendors offer many higher level services that shift the compute calculation
 - Virtual Servers are pay per hour. You pick a configuration, and it costs that much as long as you have the server “on”.
 - Other services are pay per request. You configure the service, and then you pay a small fraction for each request the service handles. This can offer tremendous savings for smaller services, but could also benefit large ones.
 - Become, hire, or befriend a cloud economist.

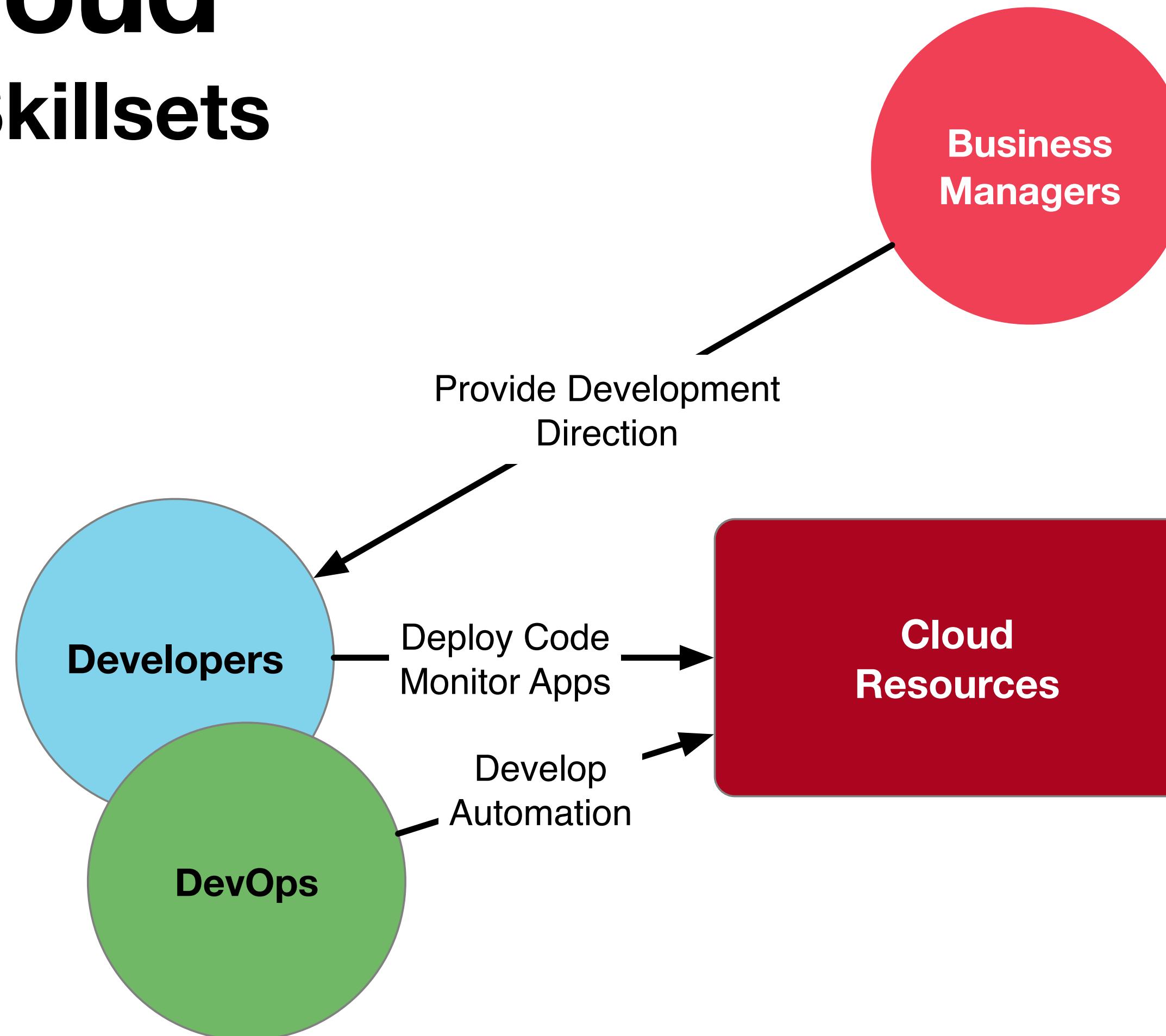
Datacenter Application Model

The Cloud Shifting Skillsets



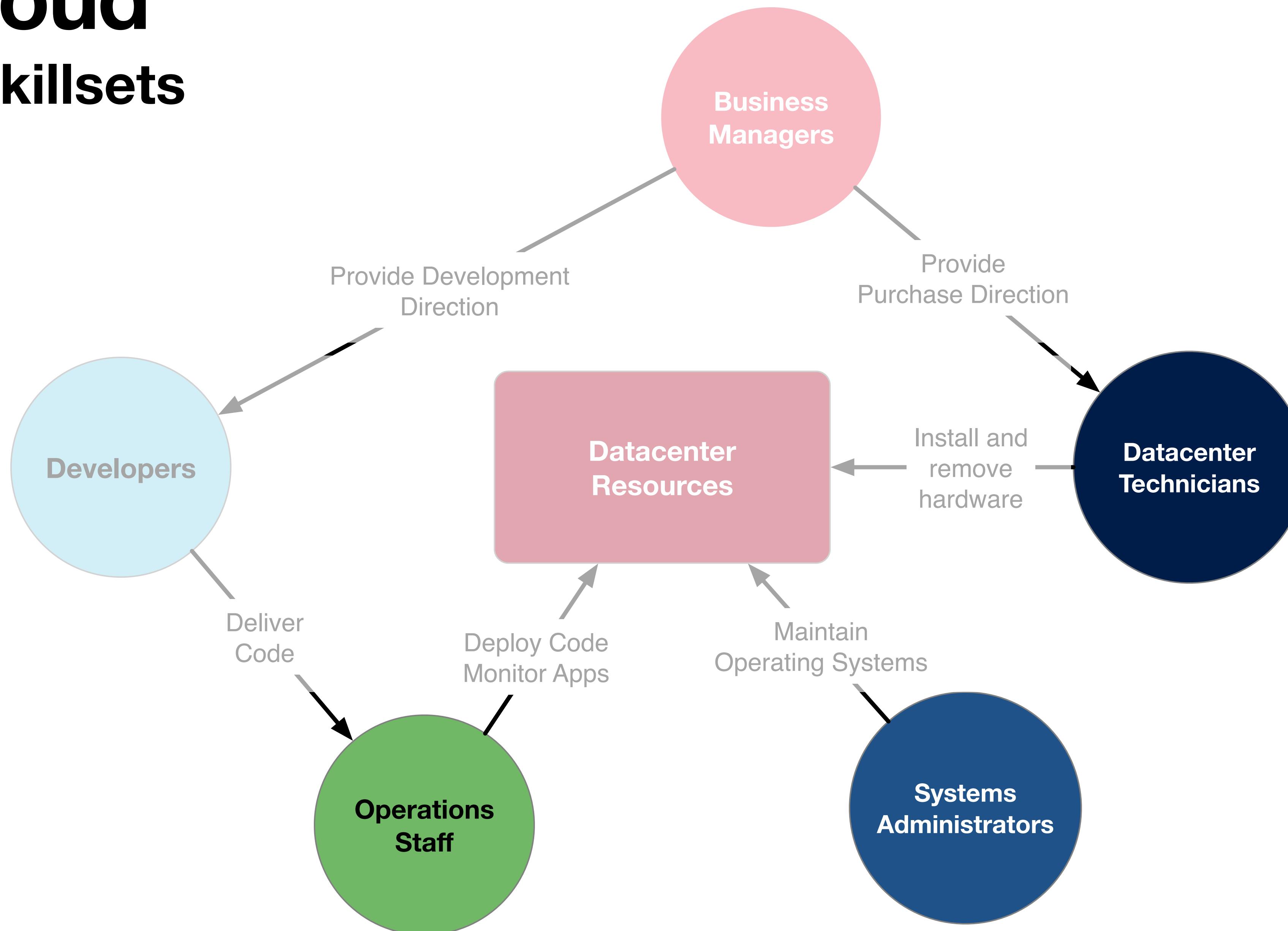
Cloud Application Model

The Cloud Shifting Skillsets



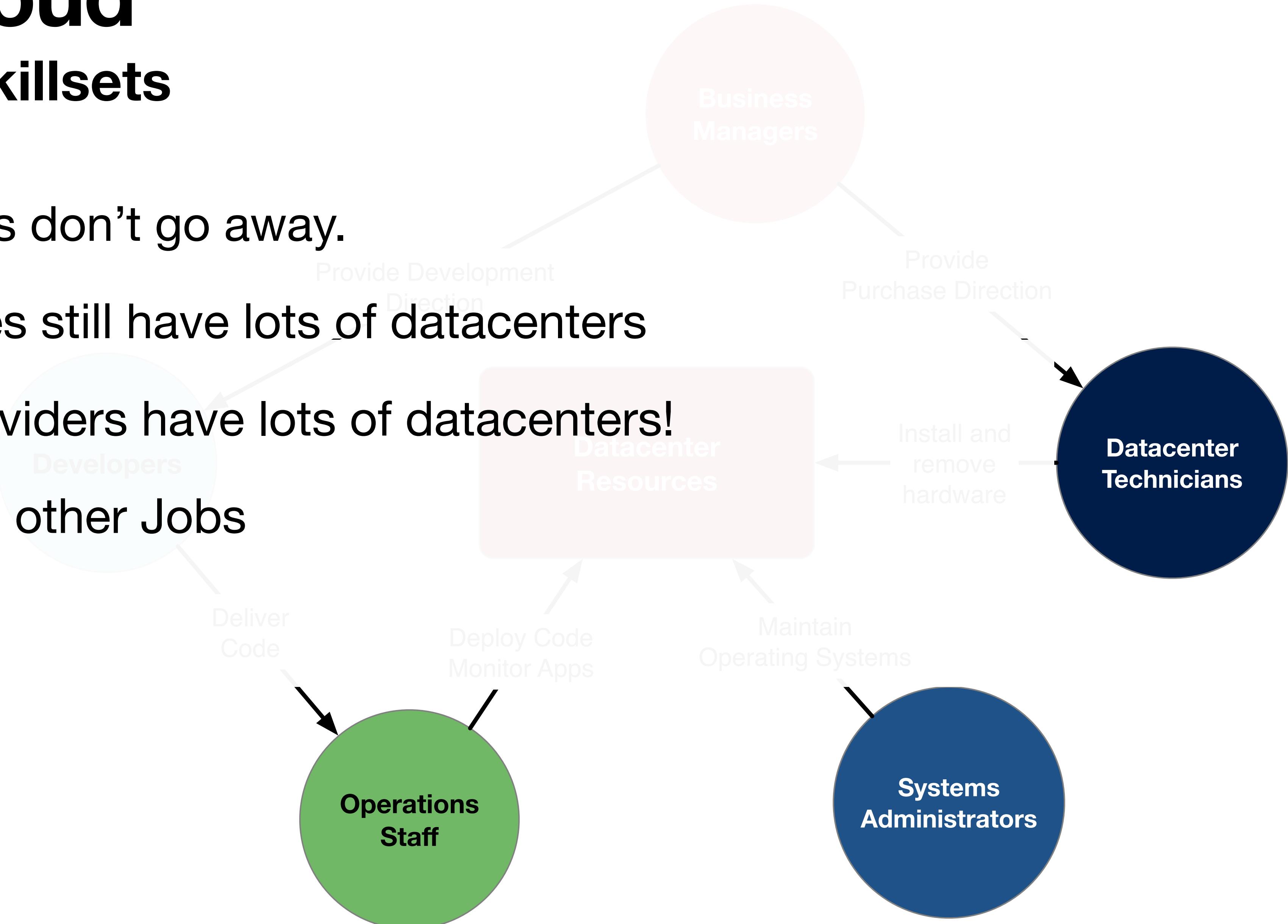
The Cloud Shifting Skillsets

What about these folks?



The Cloud Shifting Skillsets

- These jobs don't go away.
- Companies still have lots of datacenters
- Cloud Providers have lots of datacenters!
- Migrate to other Jobs



Application Development

It's All About Speed of Deployment

- Research shows one of the best indicators of high performing development teams is how often they deploy new code to production, and how fast they can do this*
- Requires automation at all levels
- Cloud providers are easier to automate
 - API First mentality

*Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations
Forsgren, Nicole ; Humble, Jez ; Kim, Gene ; 2018
Full text available at: O'Reilly Safari Learning Platform Academic
https://arizona-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01UA_ALMA21382514420003843&context=L&vid=01UA

Application Development

How Do We Deploy Quickly?

- Datacenters
 - It was hard. Each new host had to be manually configured, at least initially.
 - After initial setup, automation tools like Chef, Puppet, and Ansible could be used to setup a standard application environment, install dependencies, and deploy the application.
 - This process was still comparatively slow, taking minutes to hours to complete.
 - Operating system maintenance and patching could also be done through these orchestration tools.

Application Development

How Do We Deploy Quickly?

- Virtual Servers
 - Once the VM infrastructure was configured, a “master image” of an application could be built.
 - These images could then be deployed multiple times across VM infrastructure to build out the desired capacity.
 - Images needed to be kept up to date with security patches still.
 - Deploying code meant pushing changes into an existing VM, or re-building the entire VM image.
 - Long-lived VMs still need to be managed with orchestration like Puppet, Chef, Ansible

Application Development

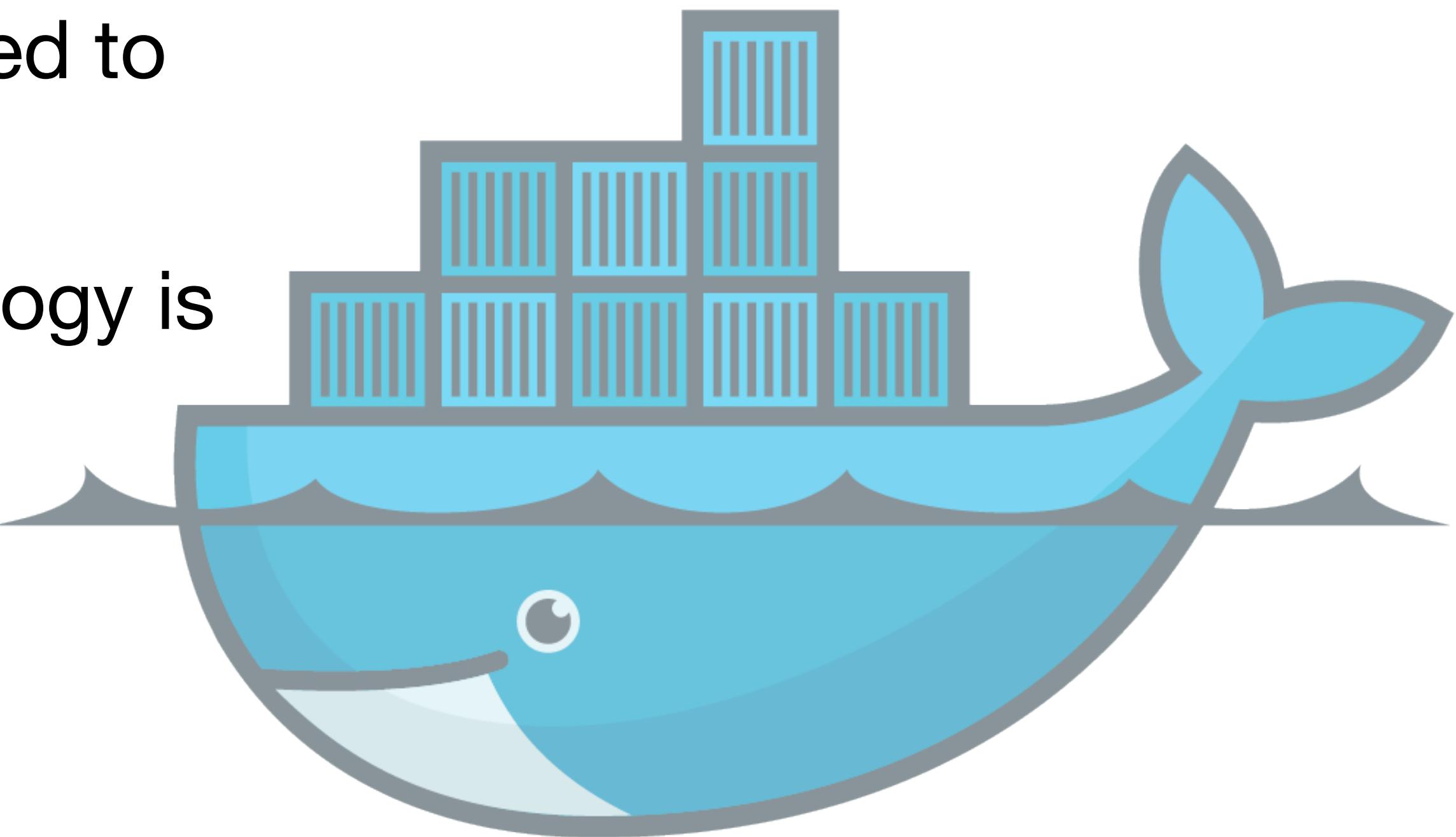
How Do We Deploy Quickly?

- Cloud Computing with “Traditional” VMs
 - Not really much different from VM infrastructure in your own datacenter.
 - You’re still responsible for:
 - Building images
 - Operating system updates and patches
 - Application code updates
 - It’s still just someone else’s computer
 - Faster. No “spare capacity” to maintain yourself.

Containers

Yeah, pretty much Docker

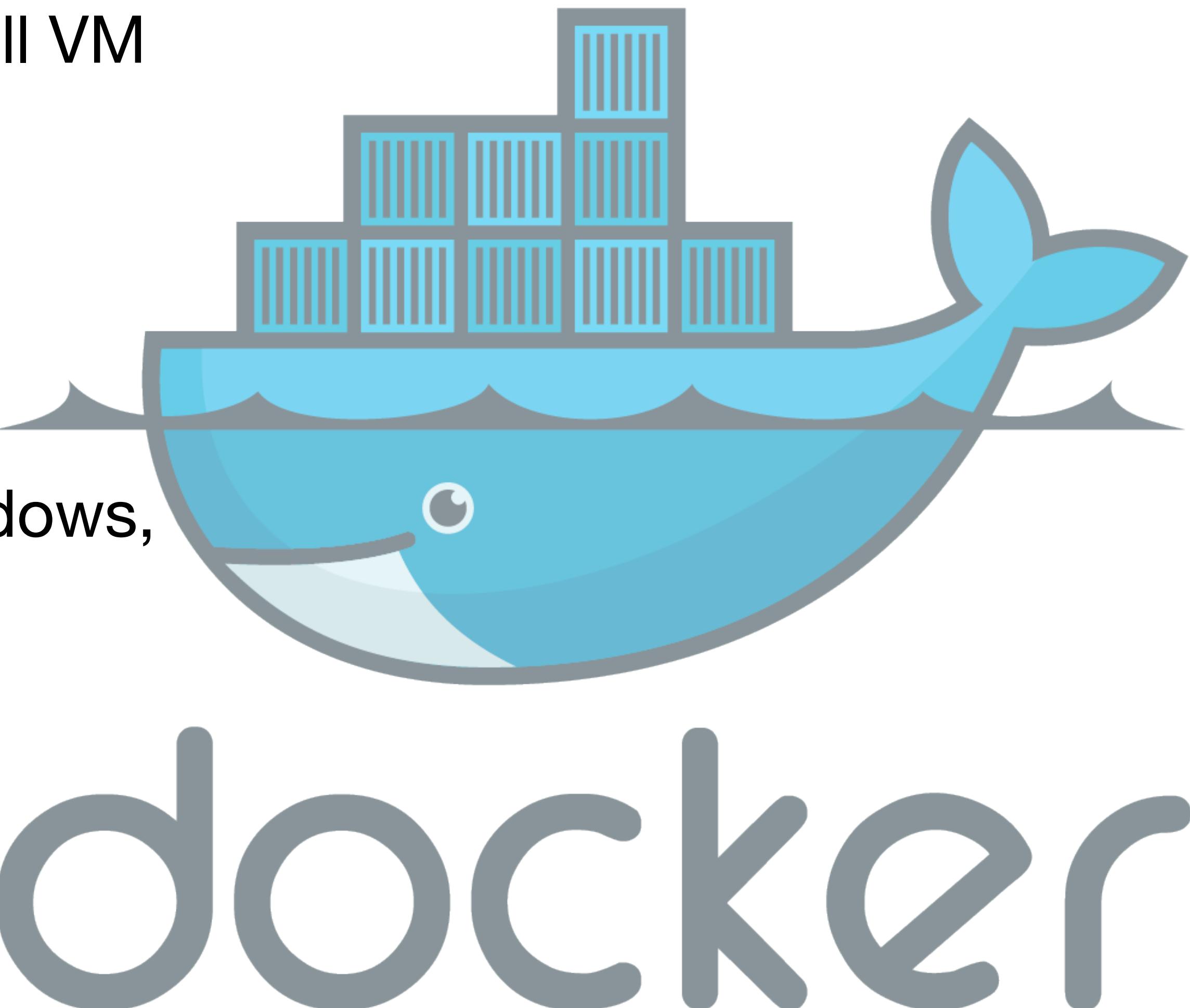
- Containers isolate all dependencies required to run an application process
- Feels like a VM, but the underlying technology is different
- Does not contain a full OS / Kernel
- All containers on a Host share the same underlying Kernel
- Processes are isolated



docker

Containers

- Container images are much smaller than full VM images.
- Host container environment can be run on commodity hardware. Does not require specialized VM infrastructure.
- The same container can run on Linux, Windows, macOS.
- Can run in Google Cloud, AWS, Azure
- Can run on your laptop



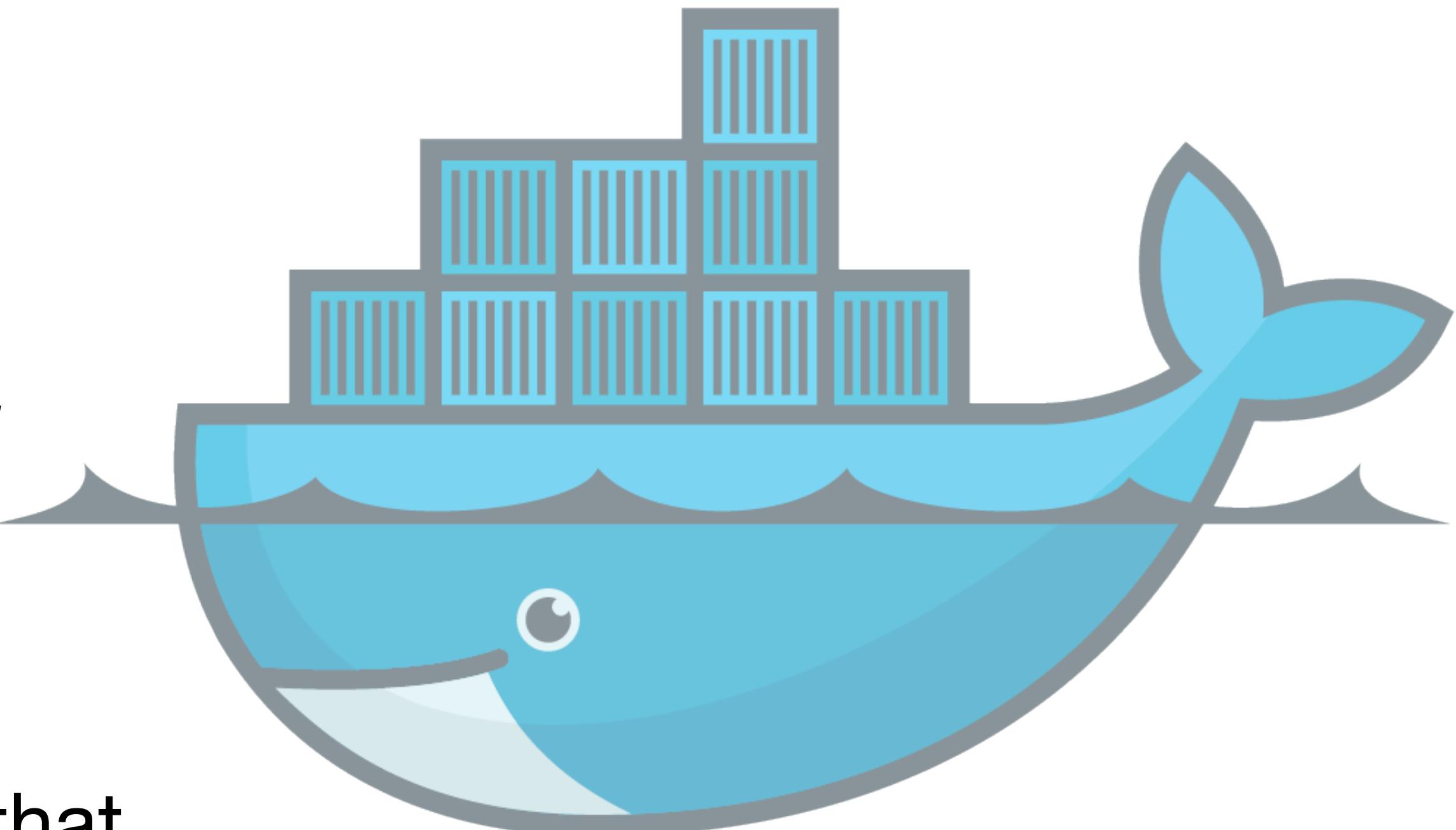
Containers & Docker

- Solves the age old problem of:

“Developer: well, it runs on my laptop.

*Operations: great, give me your laptop,
I'll put that into production.”*

- With Docker, you pretty much can do just that.



docker

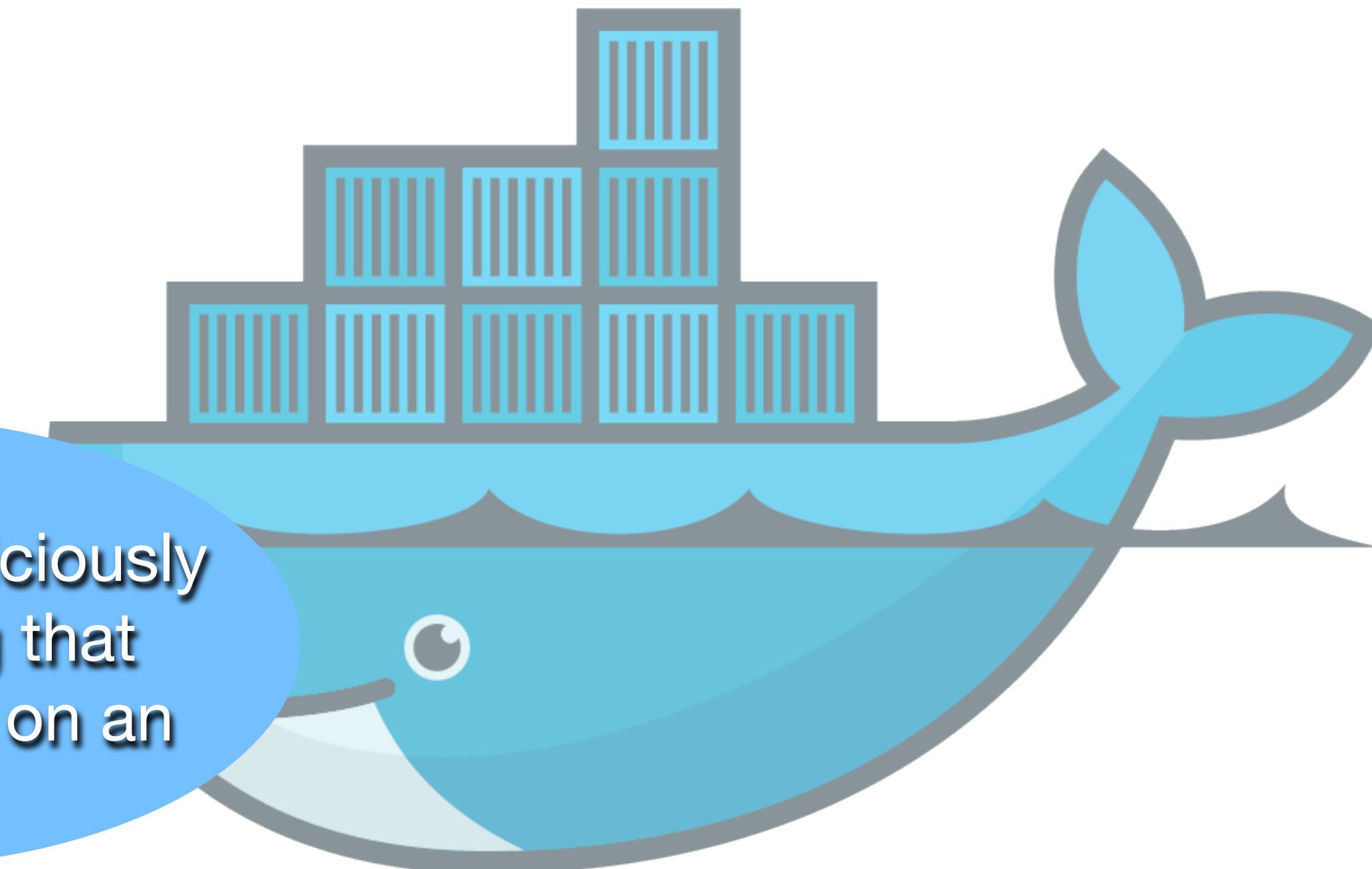
Containers & Docker

Key Concepts

- Containers isolate processes
- Containers encapsulate dependencies
- Running containers are ephemeral
- Images are immutable
- Images are composable

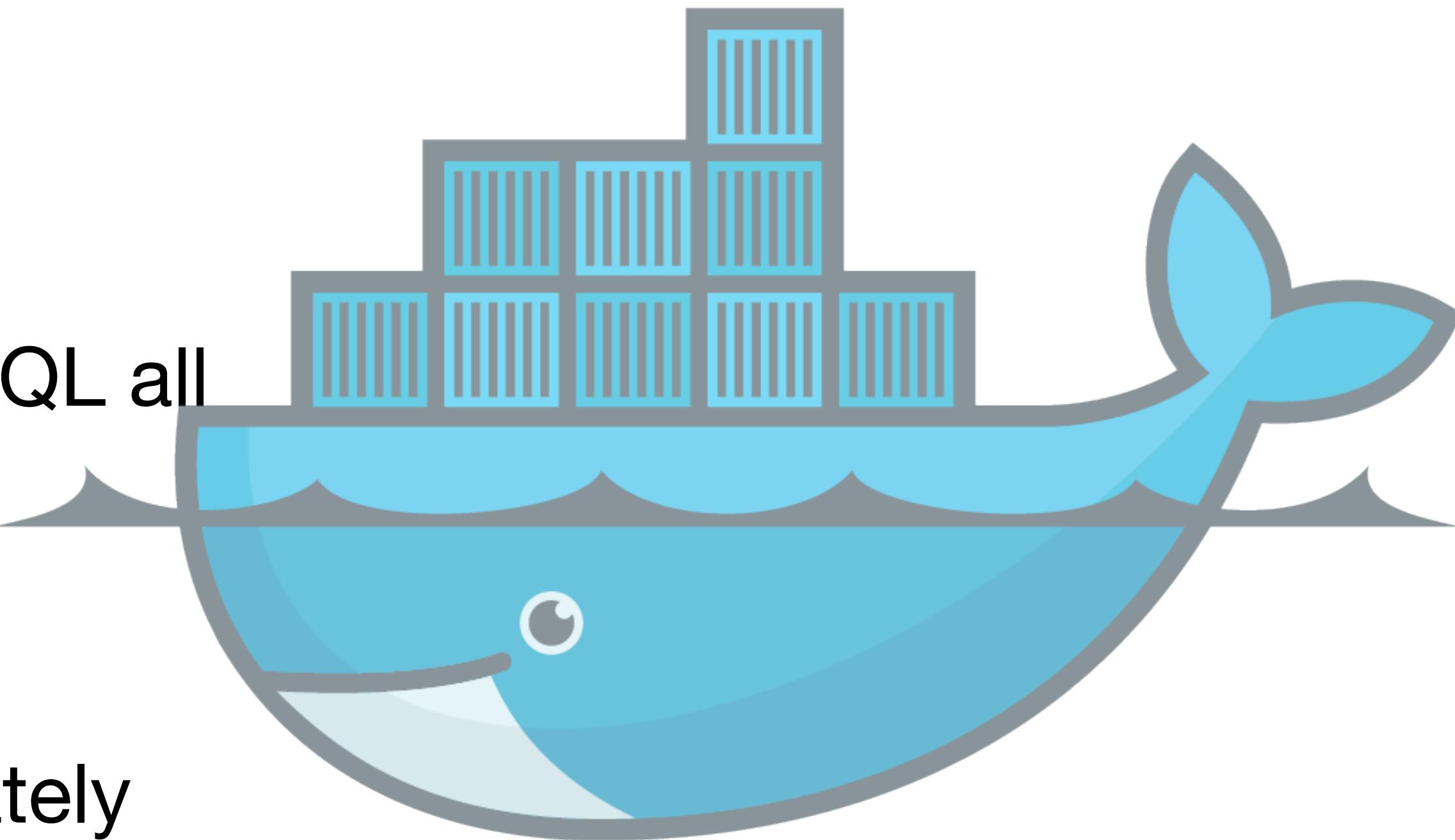


“that looks suspiciously
like something that
would show up on an
exam”

The Docker logo, featuring the word "docker" in a lowercase, rounded font with a grey-to-white gradient.

Containers & Docker

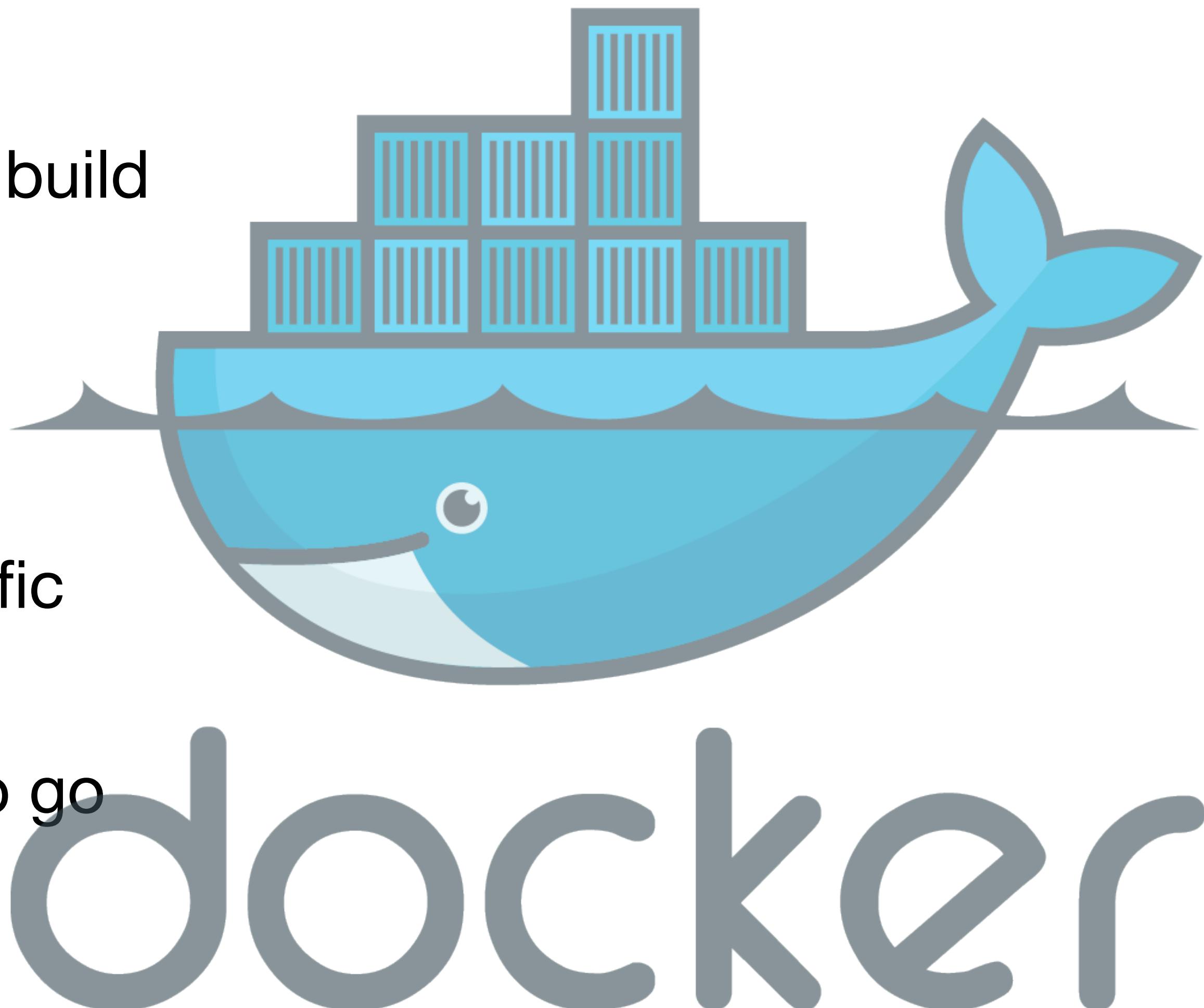
- Container isolate processes
 - A container is meant to run one process
 - You don't run Apache, Django, and MySQL all in one container
 - Instead have three separate containers
 - Allows each piece to be updated separately



docker

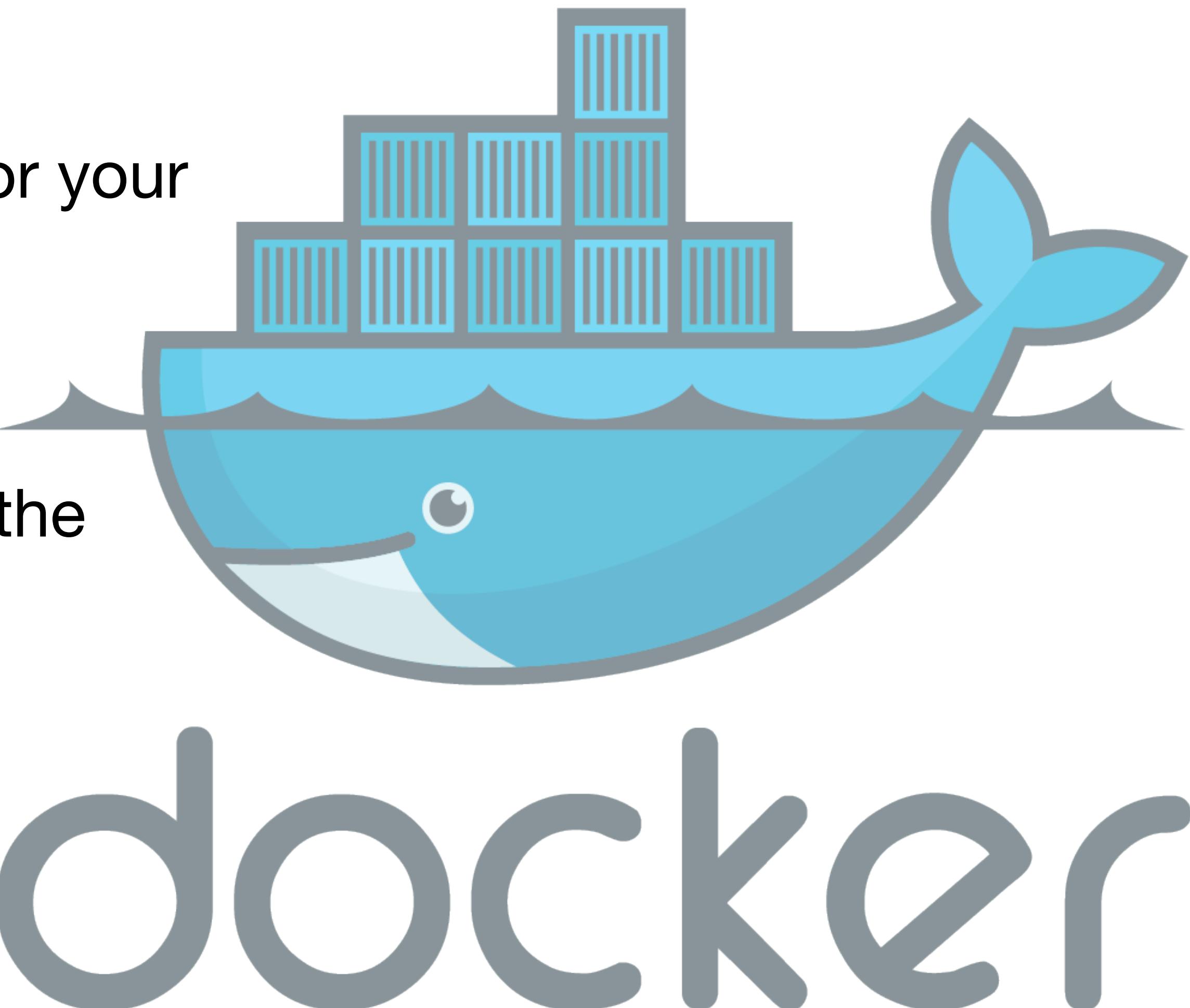
Containers & Docker

- Container images are composable
 - You can start from a “base” image, and build your changes on top of this
 - Allows other teams/companies to be responsible for base configuration
 - You just have to worry about your specific dependencies
 - No limit to how many layers you want to go



Containers & Docker

- Container encapsulate dependencies
 - All the required libraries and code files for your process can be built into the image
 - A Dockerfile is used to define an image
 - Using the Dockerfile you can then build the image

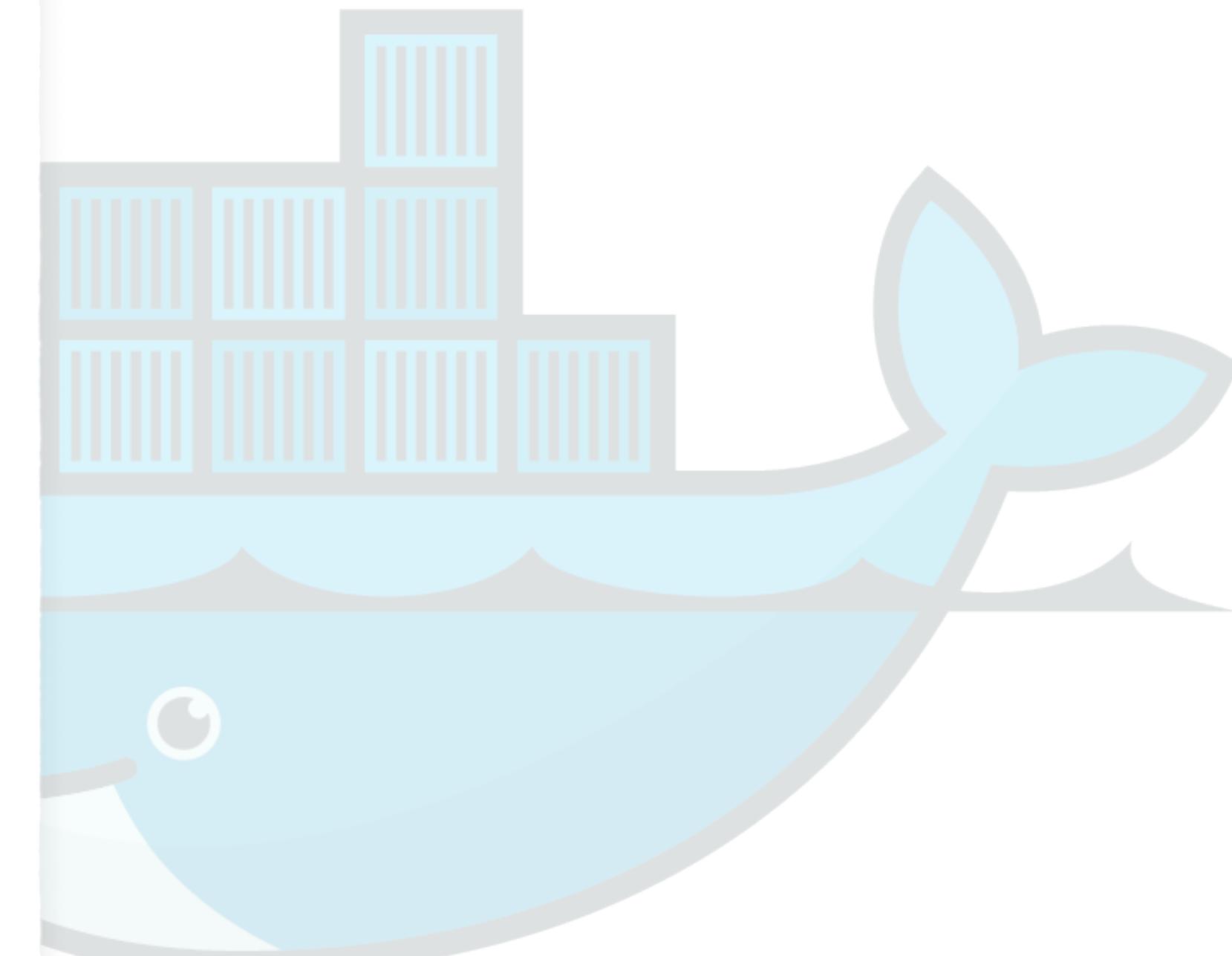


Containers & Docker

```
FROM python:3.10  
  
RUN pip install locust beautifulsoup4  
  
RUN mkdir /tests  
  
WORKDIR /tests  
  
CMD ["locust"]
```

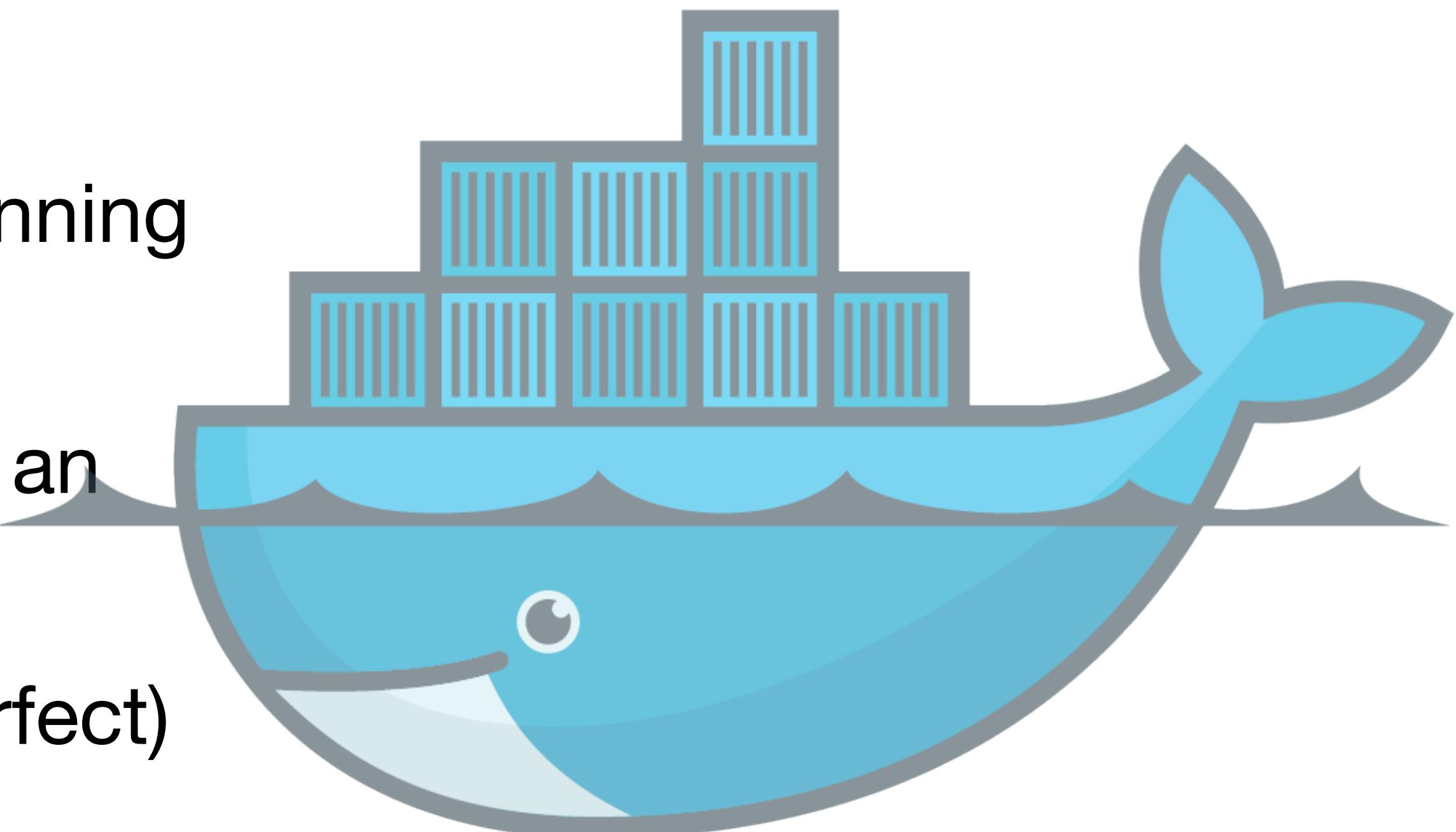
<https://locust.io>

docker



Containers & Docker

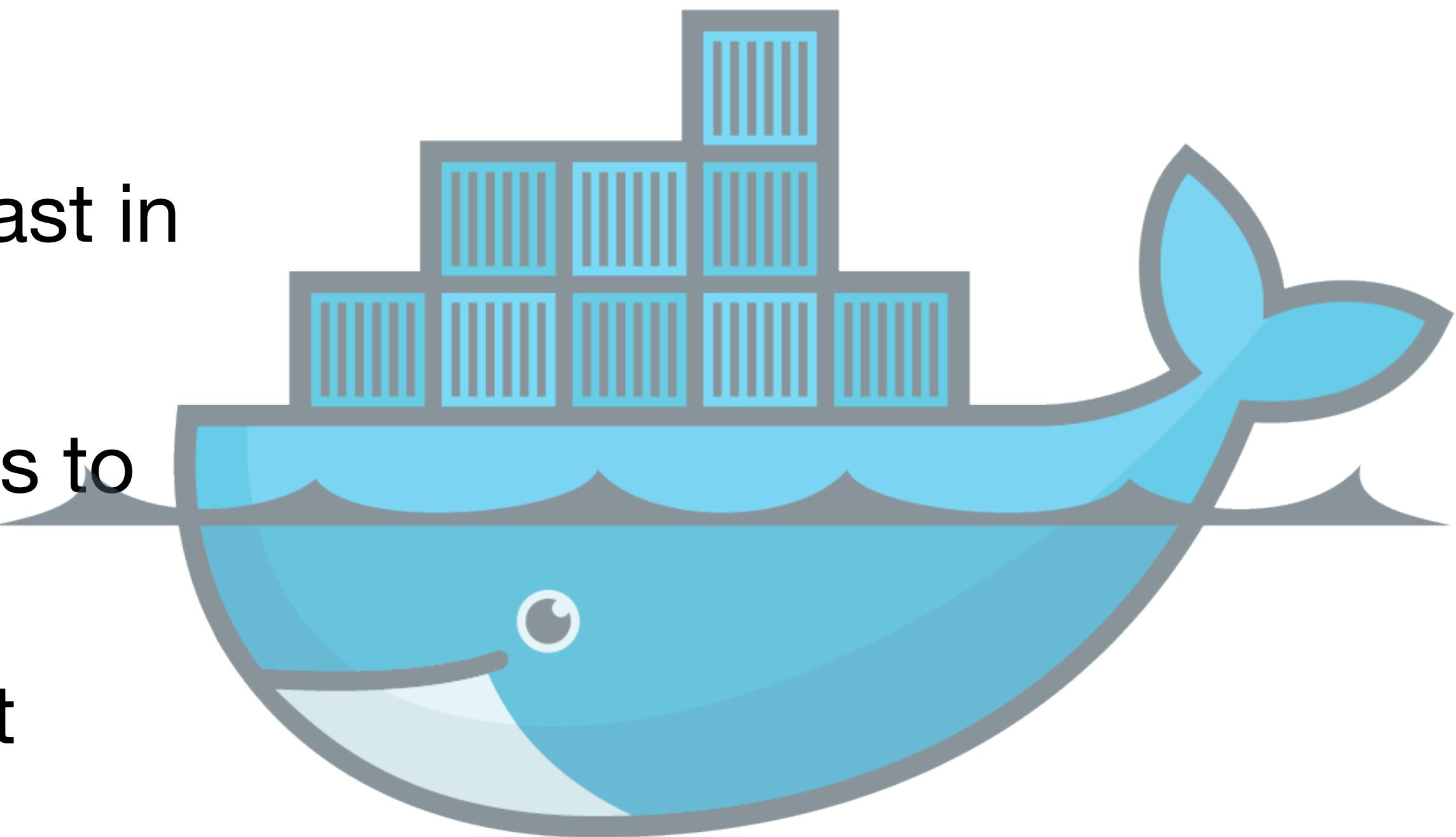
- Container images are immutable
 - When you run an image, you create a running container
 - Each time you run a container based off an image it's exactly the same.
 - Analogous to instantiating a class (imperfect)



docker

Containers & Docker

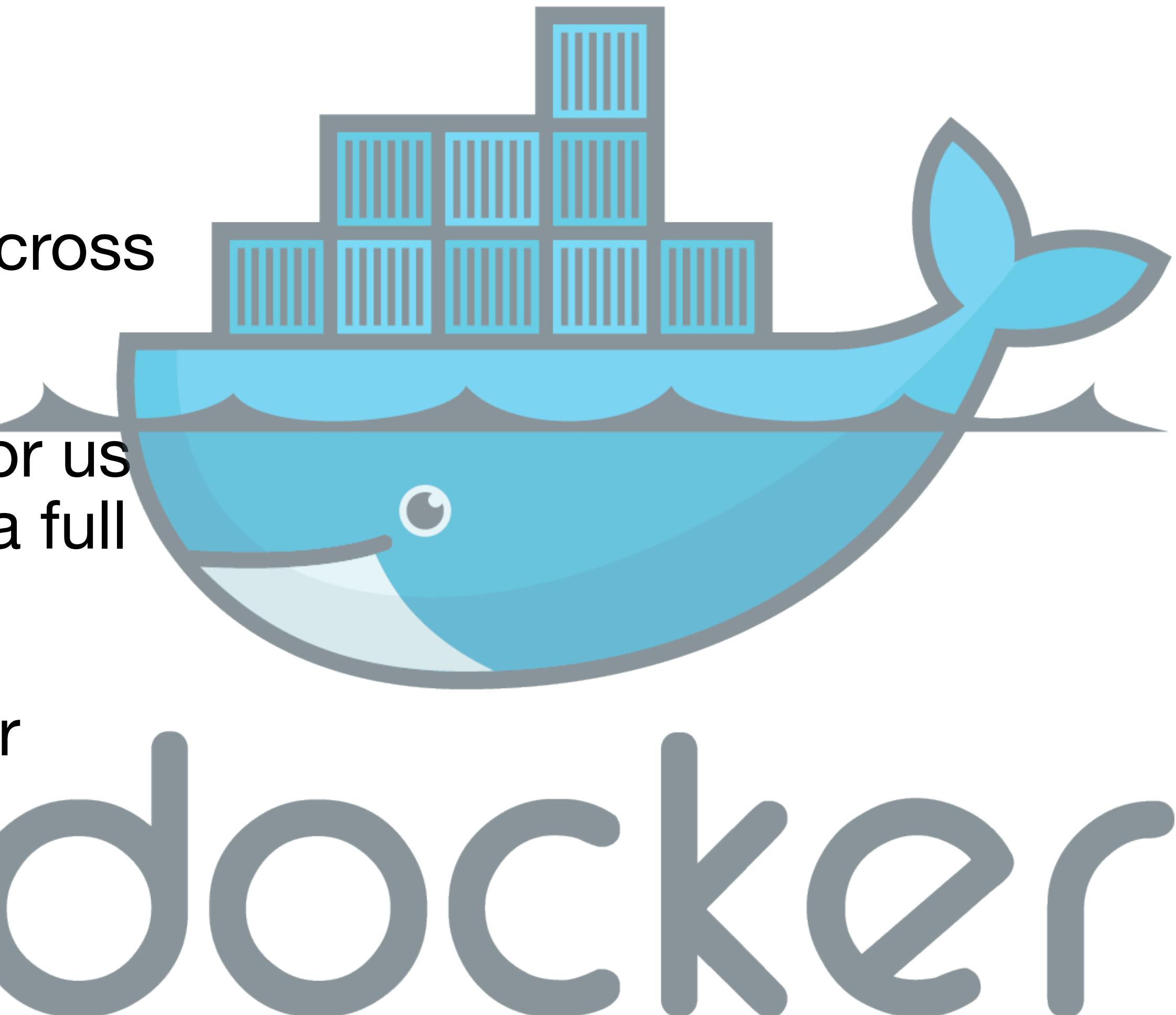
- Running containers are ephemeral
 - You don't "shut down" a container (at least in production)
 - When a container terminates, all changes to the container filesystem are lost
 - Any data the needs to be persisted must happen outside the container



docker

Why Use Docker for CSC 346?

- You all have different laptops
- Docker gives us a way to have a standard development and evaluation environment across widely varying hosts
- You can turn in Dockerfiles and code files for us to run and test, without having to maintain a full Virtual Machine
- Remember that containers do not save their filesystem! Don't lose your work!



UNIX Environments

Specifically, ubuntu/debian linux

- Linux is the default for most cloud hosts
- Linux is cheaper than MS Windows for servers
- Many platforms default to ubuntu, so why fight it
 - For example: official python images

```
FROM python:3.10
```

```
RUN pip install ...
```

your Dockerfile

```
FROM buildpack-deps:bullseye
```

```
ENV PATH /usr/local/bin:$PATH
```

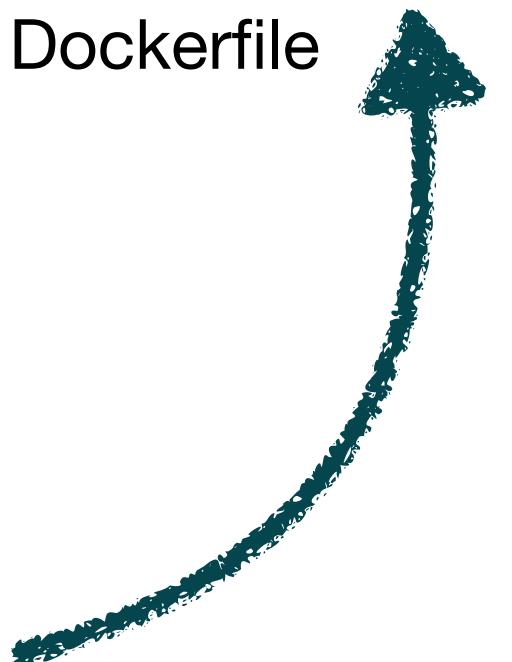
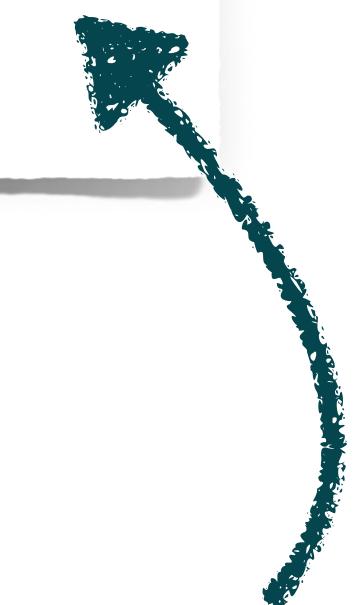
python:3:10 Dockerfile

```
FROM scratch  
ADD rootfs.tar.xz /  
CMD ["bash"]
```

debian:bullseye Dockerfile

```
FROM debian:bullseye
```

buildpack-deps:bullseye Dockerfile



What is UNIX?

- Bell Labs in the early 1970s
- Spawned many Open Source derivatives
 - BSD → Darwin → macOS
 - Linux → Debian → Ubuntu
 - Linux → Android
- Nearly unchallenged in the server / cloud space
 - Great process model
 - Developer friendly
 - Great command line interface

Linux Basics

Files and Directories

- Linux organizes a filesystem based mainly on files and directories
 - Directories = Folders - We will not be pedantic about this 😊
- A filesystem is organized into a **directory tree**
 - Directories = branches Files = leaves
- A filesystem has a single root directory
- Linux uses the “forward slash”, or just “slash” as the directory delimiter

/Users/mark/Documents/csc246/01-cloud-docker.key

Linux Basics

Users and Groups

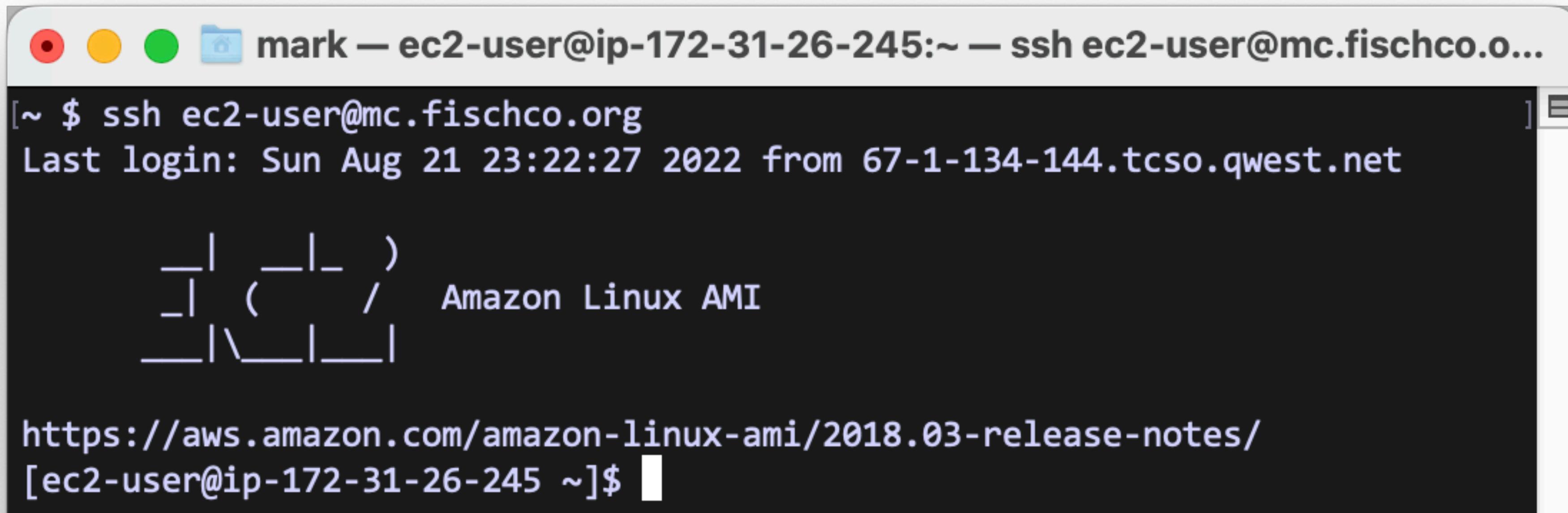
- Files are owned by users
- A ‘root’ user has access to everything
- Users can belong to groups
- File and Directories can have permissions that grant various access to users and groups
- Docker containers run everything inside them as a local root user, this is different from the host’s root user.

Linux Basics

Connecting

- A remote host is usually accessed through a Secure Shell - ssh

```
$ ssh username@hostname
```



The screenshot shows a macOS terminal window with the following details:

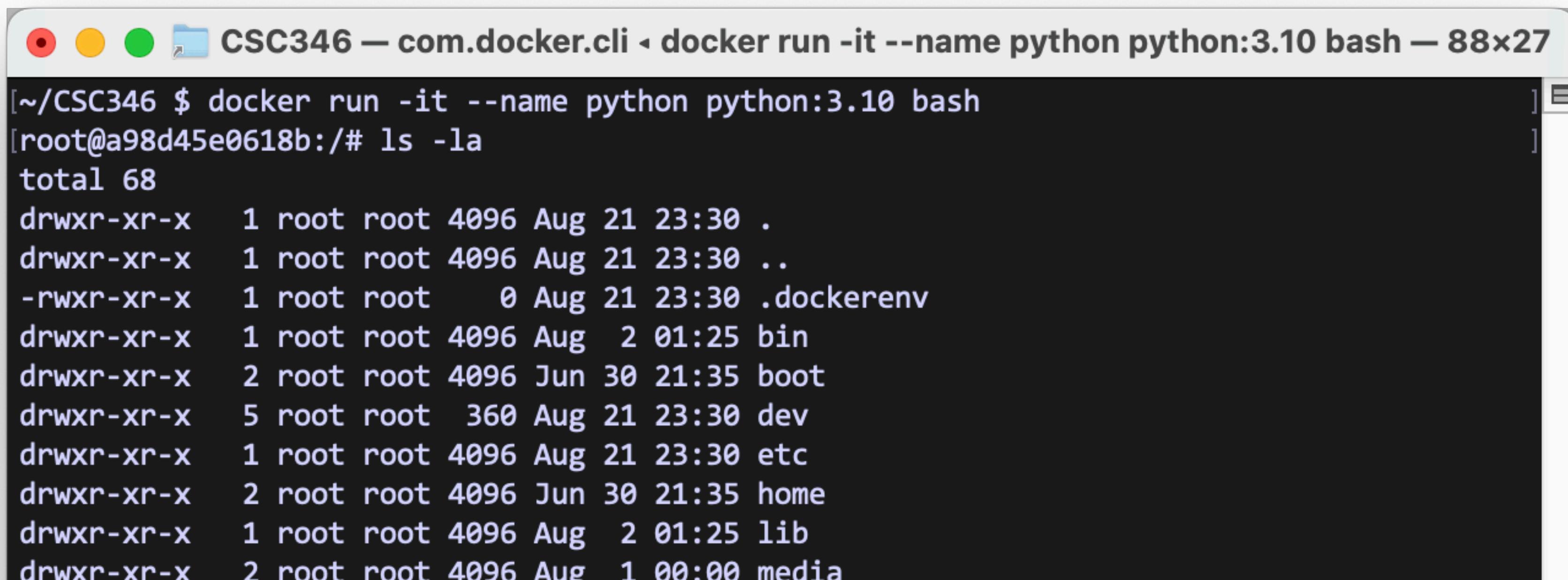
- Window Title:** mark — ec2-user@ip-172-31-26-245:~ — ssh ec2-user@mc.fischco.o...
- Terminal Prompt:** [~ \$ ssh ec2-user@mc.fischco.org
- Login Information:** Last login: Sun Aug 21 23:22:27 2022 from 67-1-134-144.tcso.qwest.net
- Amazon Linux AMI Logo:** A stylized logo consisting of vertical and diagonal lines forming a face-like shape.
- Release Notes URL:** https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
- Current Working Directory:** [ec2-user@ip-172-31-26-245 ~]\$

Linux Basics

Connecting

- A local docker container can be accessed either through the initial run command, or by an exec command.

```
$ docker run -it --name python python:3.10 bash
```



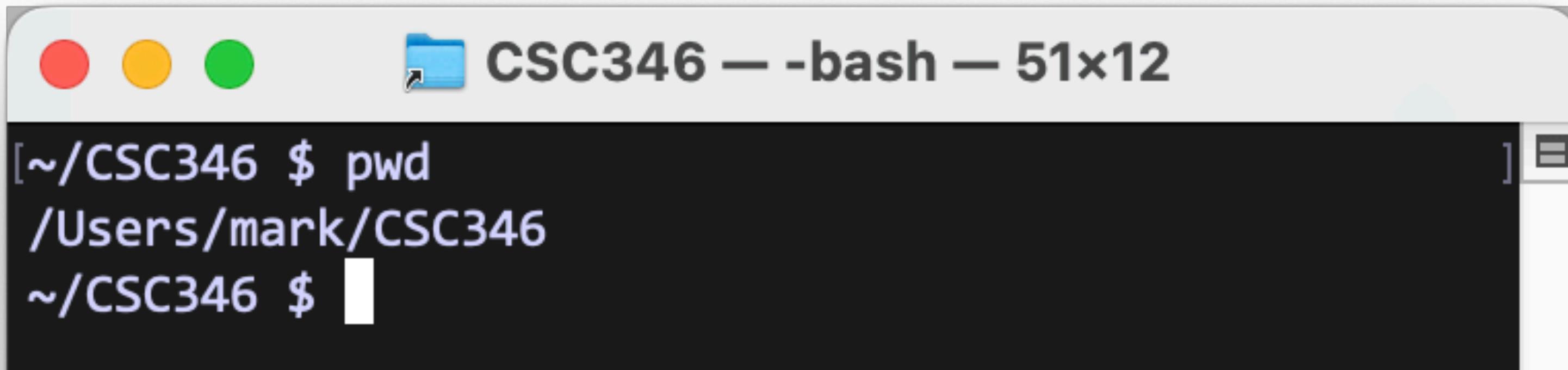
The screenshot shows a terminal window titled "CSC346 — com.docker.cli · docker run -it --name python python:3.10 bash — 88x27". The terminal displays the following command and its output:

```
[~/CSC346 $ docker run -it --name python python:3.10 bash
[root@a98d45e0618b:/# ls -la
total 68
drwxr-xr-x  1 root root 4096 Aug 21 23:30 .
drwxr-xr-x  1 root root 4096 Aug 21 23:30 ..
-rw xr-xr-x  1 root root    0 Aug 21 23:30 .dockerenv
drwxr-xr-x  1 root root 4096 Aug  2 01:25 bin
drwxr-xr-x  2 root root 4096 Jun 30 21:35 boot
drwxr-xr-x  5 root root  360 Aug 21 23:30 dev
drwxr-xr-x  1 root root 4096 Aug 21 23:30 etc
drwxr-xr-x  2 root root 4096 Jun 30 21:35 home
drwxr-xr-x  1 root root 4096 Aug  2 01:25 lib
drwxr-xr-x  2 root root 4096 Aug  1 00:00 media
```

Linux Basics

Where Am I?

- When you first connect to a linux host, your CLI session will usually start in the user's home directory
- Docker containers usually start at the WORKDIR defined in that Image's Dockerfile
 - If WORKDIR is not defined, you'll start at the filesystem root: /
- Use `pwd` to see your filesystem location (Present Working Directory)



```
[~/CSC346 $ pwd
/Users/mark/CSC346
~/CSC346 $ ]
```

Linux Basics

What Stuff Is Here?

- To see the contents of the directory you're in, use the `ls` command (list)

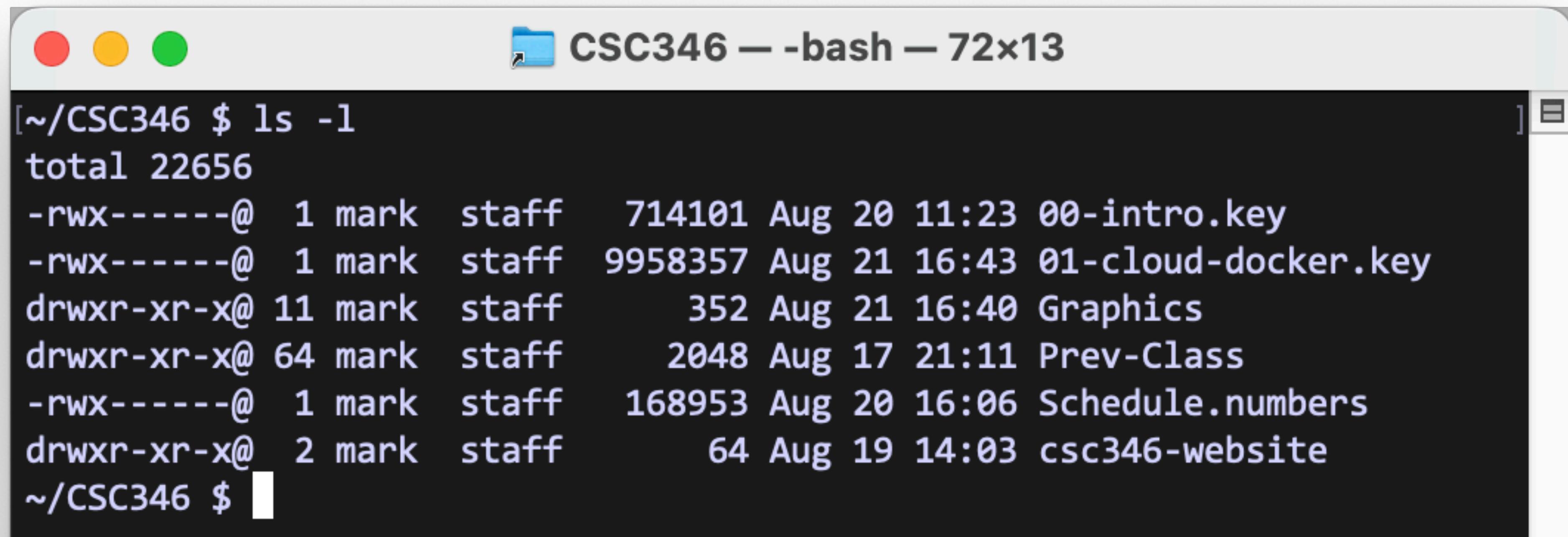
```
● ● ● 📂 CSC346 — -bash — 51x12

[~/CSC346 $ pwd
/Users/mark/CSC346
[~/CSC346 $ ls
00-intro.key
01-cloud-docker.key
Graphics
~/CSC346 $ █
```

Linux Basics

CLI Arguments

- Most CLI commands support arguments and options. Tells the command to do different things.
- The `ls` command accepts the `-l` option to list files in the long format.

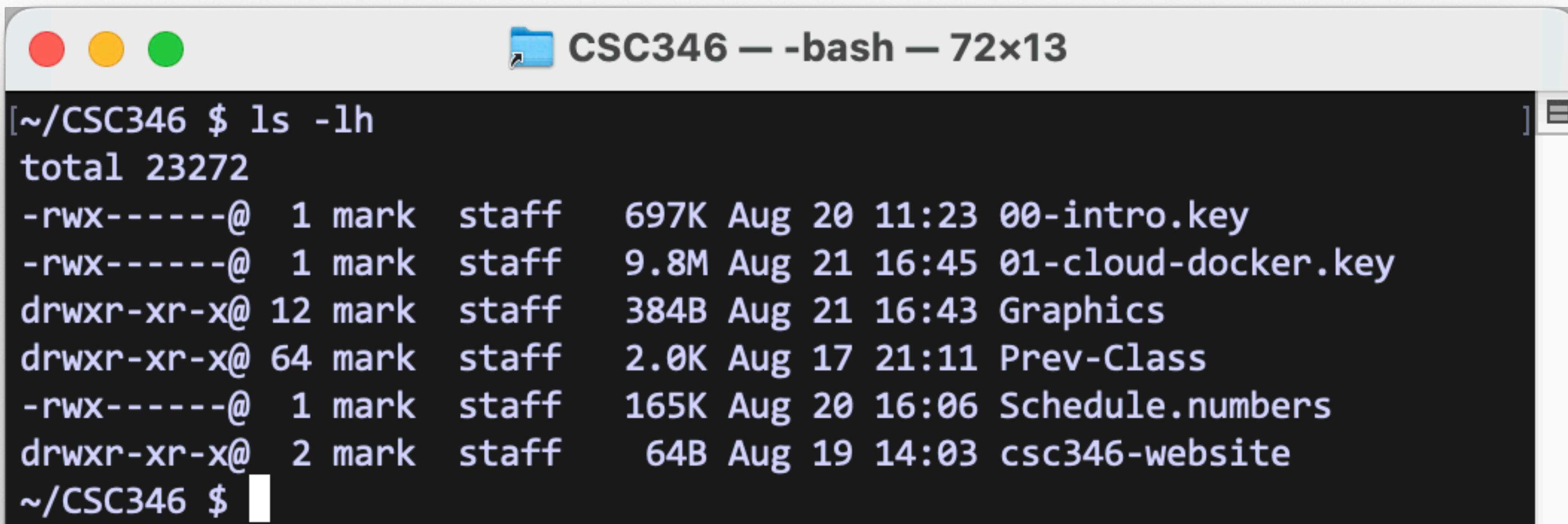


```
[~/CSC346 $ ls -l
total 22656
-rwx-----@ 1 mark staff 714101 Aug 20 11:23 00-intro.key
-rwx-----@ 1 mark staff 9958357 Aug 21 16:43 01-cloud-docker.key
drwxr-xr-x@ 11 mark staff 352 Aug 21 16:40 Graphics
drwxr-xr-x@ 64 mark staff 2048 Aug 17 21:11 Prev-Class
-rwx-----@ 1 mark staff 168953 Aug 20 16:06 Schedule.numbers
drwxr-xr-x@ 2 mark staff 64 Aug 19 14:03 csc346-website
~/CSC346 $ ]
```

Linux Basics

CLI Arguments

- By default the `-l` long format shows file sizes in bytes.
- Use the `-h` option to show sizes in human readable format.
- Multiple options can be combined with the same dash: `-lh`

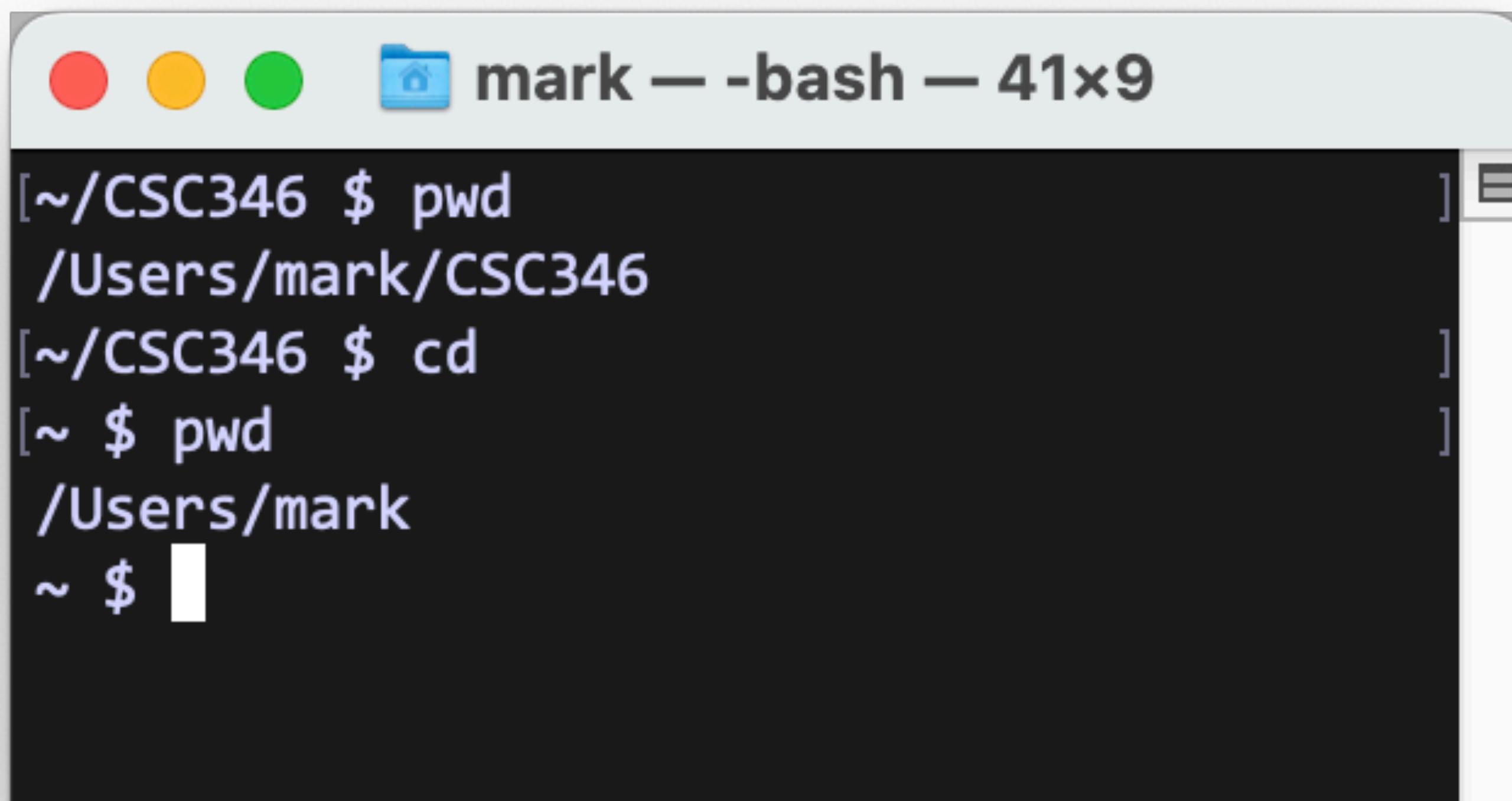


```
[~/CSC346 $ ls -lh
total 23272
-rwx-----@ 1 mark staff 697K Aug 20 11:23 00-intro.key
-rwx-----@ 1 mark staff 9.8M Aug 21 16:45 01-cloud-docker.key
drwxr-xr-x@ 12 mark staff 384B Aug 21 16:43 Graphics
drwxr-xr-x@ 64 mark staff 2.0K Aug 17 21:11 Prev-Class
-rwx-----@ 1 mark staff 165K Aug 20 16:06 Schedule.numbers
drwxr-xr-x@ 2 mark staff 64B Aug 19 14:03 csc346-website
~/CSC346 $ ]
```

Linux Basics

Moving Yourself Around

- To move to a different directory, use the `cd` command (Change Directory)
- If used without an argument, `cd` will take you to your home directory.

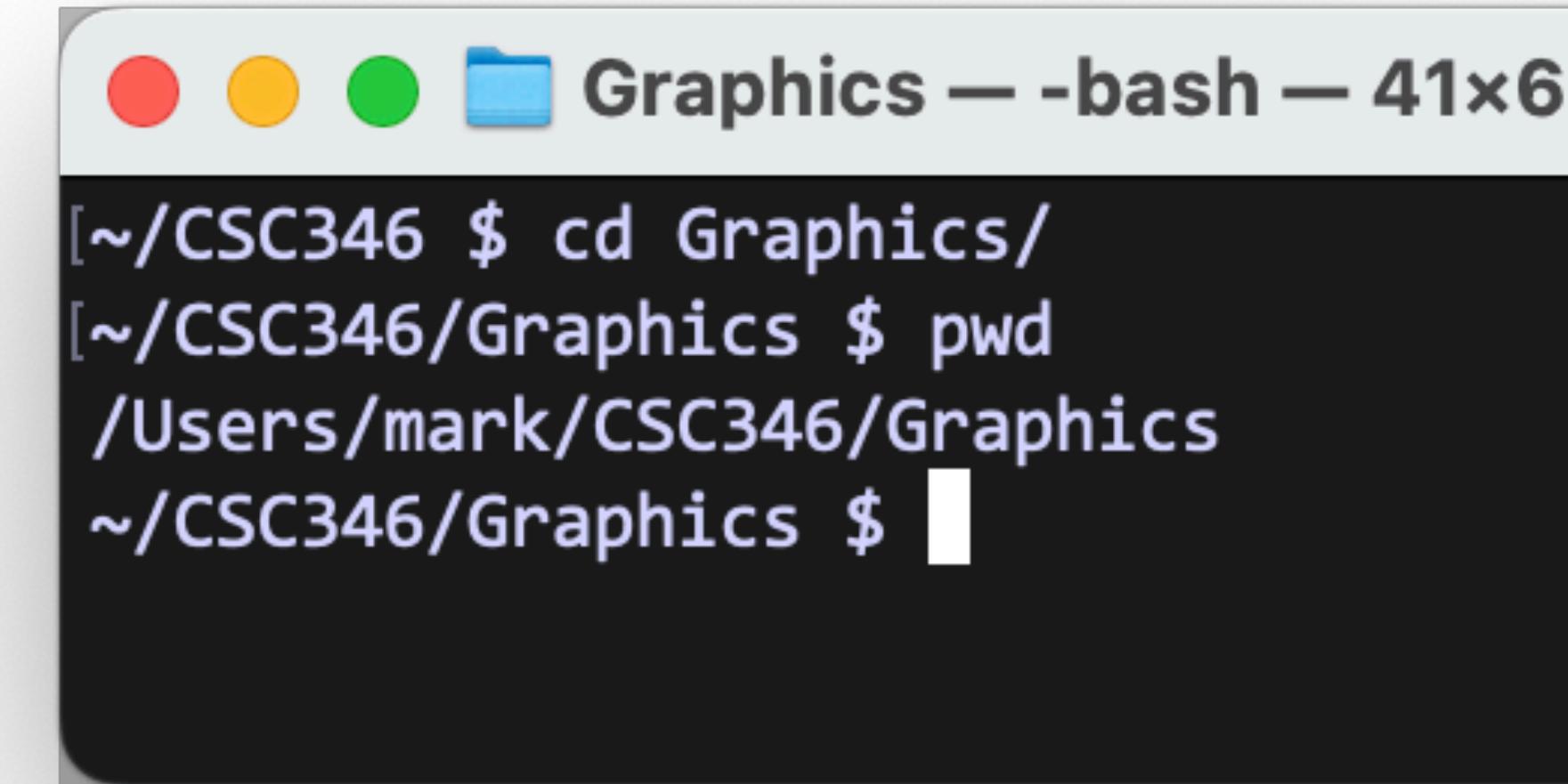


```
[~/CSC346 $ pwd  
/Users/mark/CSC346  
[~/CSC346 $ cd  
[~ $ pwd  
/Users/mark  
~ $ ]
```

Linux Basics

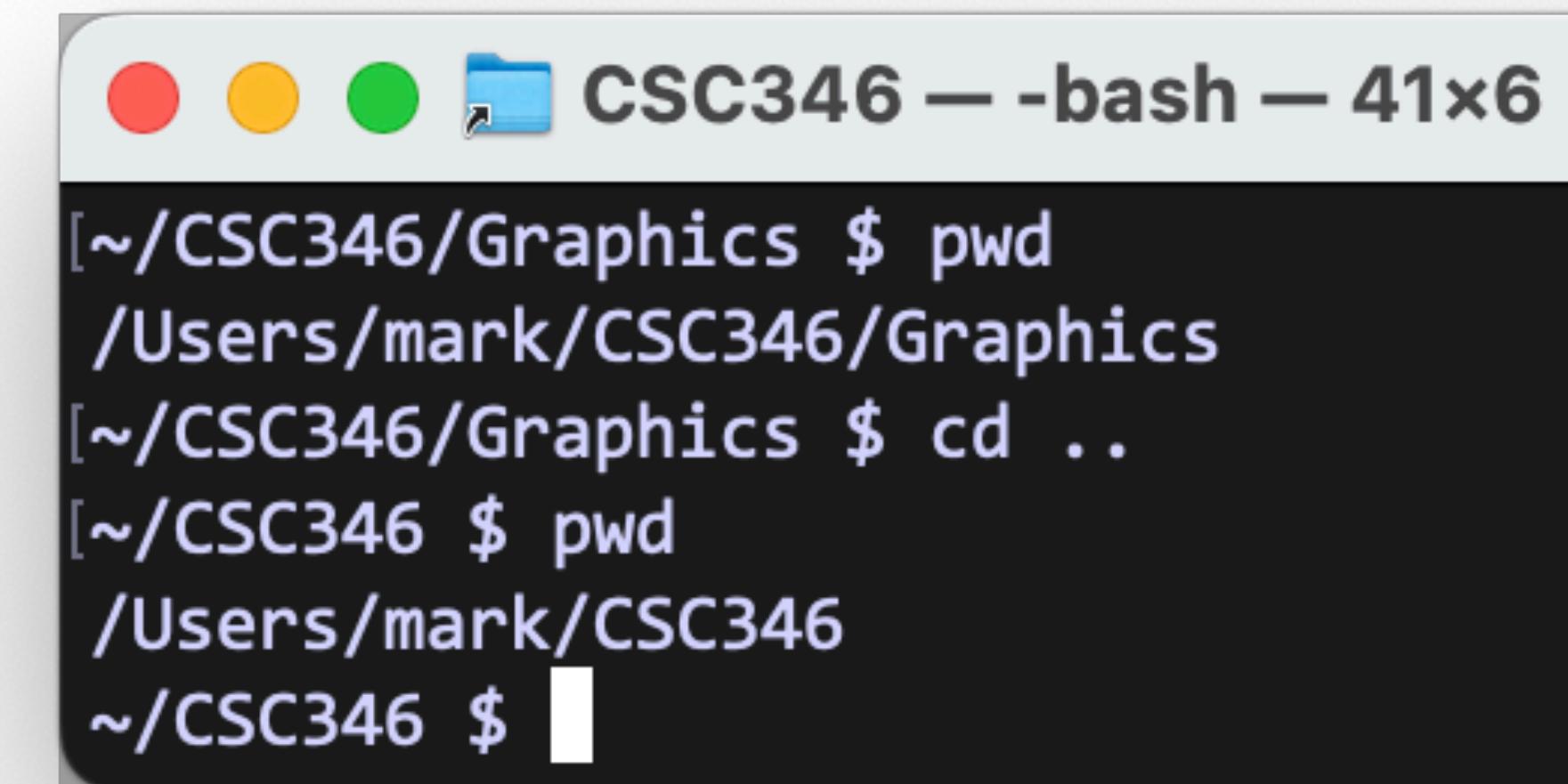
Moving Yourself Around

- To move into another directory contained in the current one, use `cd dirname`



```
[~/CSC346 $ cd Graphics/
[~/CSC346/Graphics $ pwd
/Users/mark/CSC346/Graphics
~/CSC346/Graphics $ ]
```

- To move up a directory, use the special “`..`” directory



```
[~/CSC346/Graphics $ pwd
/Users/mark/CSC346/Graphics
[~/CSC346/Graphics $ cd ..
[~/CSC346 $ pwd
/Users/mark/CSC346
~/CSC346 $ ]
```

Linux Basics

Core CLI Commands

<code>pwd</code>	Prints your present working directory
<code>ls</code>	Lists the files in your current directory
<code>ls -lh</code>	Lists the files in your current directory in long form, with human readable file sizes
<code>cd [dir]</code>	Change your current working directory to <code>[dir]</code>
<code>mkdir [dir]</code>	Create a new directory named <code>dir</code> inside your current working directory
<code>mv [from] [to]</code>	Move a file from one location to another. If <code>to</code> is not within another directory, it renames the file in your current directory
<code>cp [from] [to]</code>	Copy a file <code>from</code> one location <code>to</code> another
<code>rm [file]</code>	Delete a <code>file</code> (remove it)



Linux Basics

Core CLI Commands

cat [file]	Prints the full contents of file to the screen
grep [string] [file]	Search file for the specified string
head -n [count] [file]	Print the first n lines of a file to the screen.
tail -n [count] [file]	Print the last n lines of a file to the screen.
tail -f [file]	Print the last few lines of a file to the screen, and continue to follow it as new lines are added.
less [file]	Prints out the contents of the first page of a file to your screen, and gives you keyboard commands for navigating through the file. Read-only.

Linux Basics

CLI Text Editors

- Popular editors: `vi`, `vim`, `emacs`, `nano`
 - All keyboard and text based. No mouse.
- I mostly try and avoid CLI text editors. I like my GUI!
 - We'll see many strategies for avoiding the CLI editors
- When I need to, I mostly use `vim` or `vi` depending on what is available

Linux Basics

STDOUT, Redirection, and Pipes

- UNIX has a concept of Standard Out (STDOUT) and Standard Error (STDERR)
- By default STDOUT is directed to your terminal screen
- STDOUT can be redirected to other places though

```
ls -l > output.txt
```

Sends the **STDOUT** of the **ls** command to a file named **output.txt**. If that file exists, it will be overwritten. If the file does not exist, it will be created.

```
ls -l >> output.txt
```

Appends the **STDOUT** of the **ls** command to a file named **output.txt**. If that file exists, it will add new output to the end of the file. If the file does not exist, it will be created.

```
python3 ./prog.py | less
```

Pipe the **STDOUT** of the python program to **less**. This lets you scroll through the output of **prog.py** while still letting new text come in at the bottom.

Development Environments

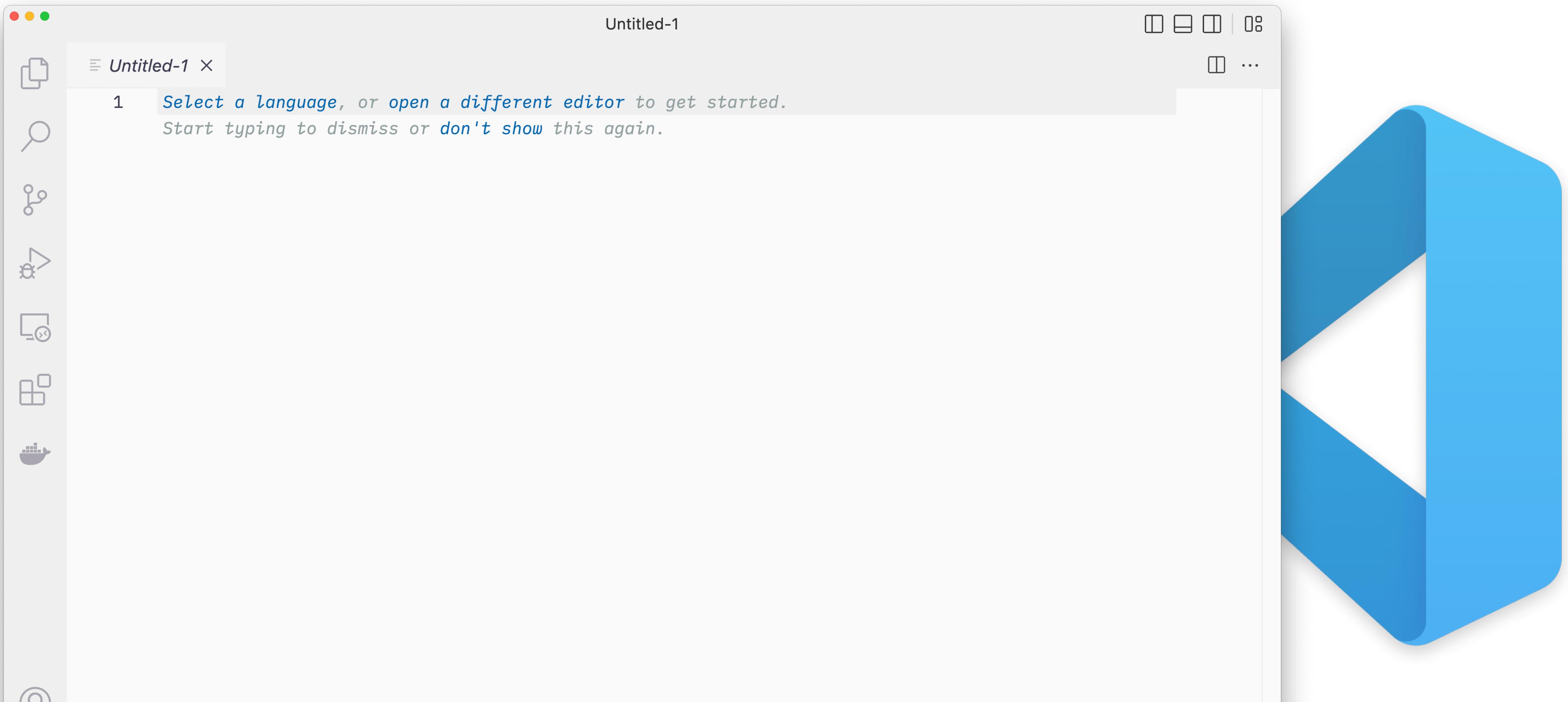
Microsoft VS Code

- Not required, but it's really great
- Free
- GUI Text editor and terminal all in one
- Can open a local folder and use it as a project
- Customizable
- Plugins for just about everything



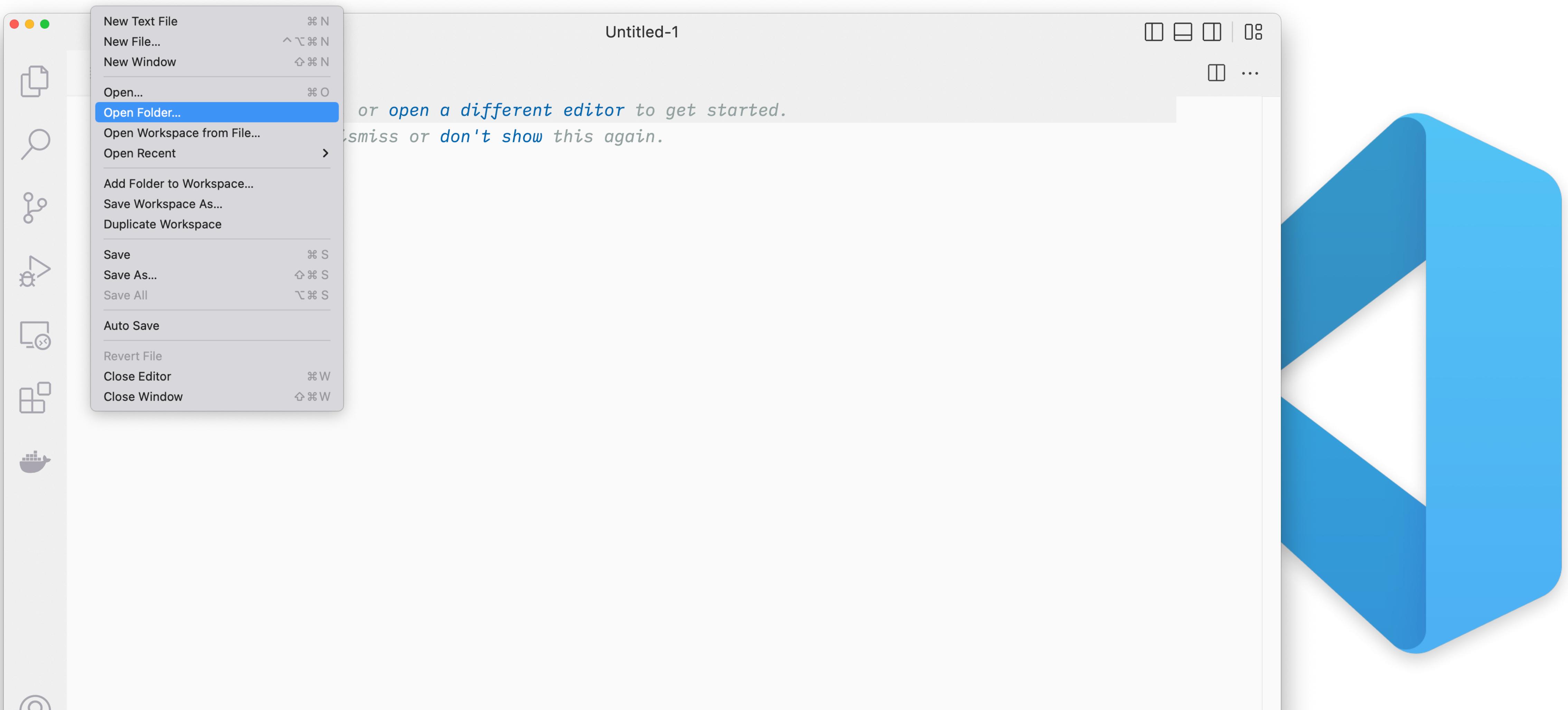
Development Environments

Microsoft VS Code



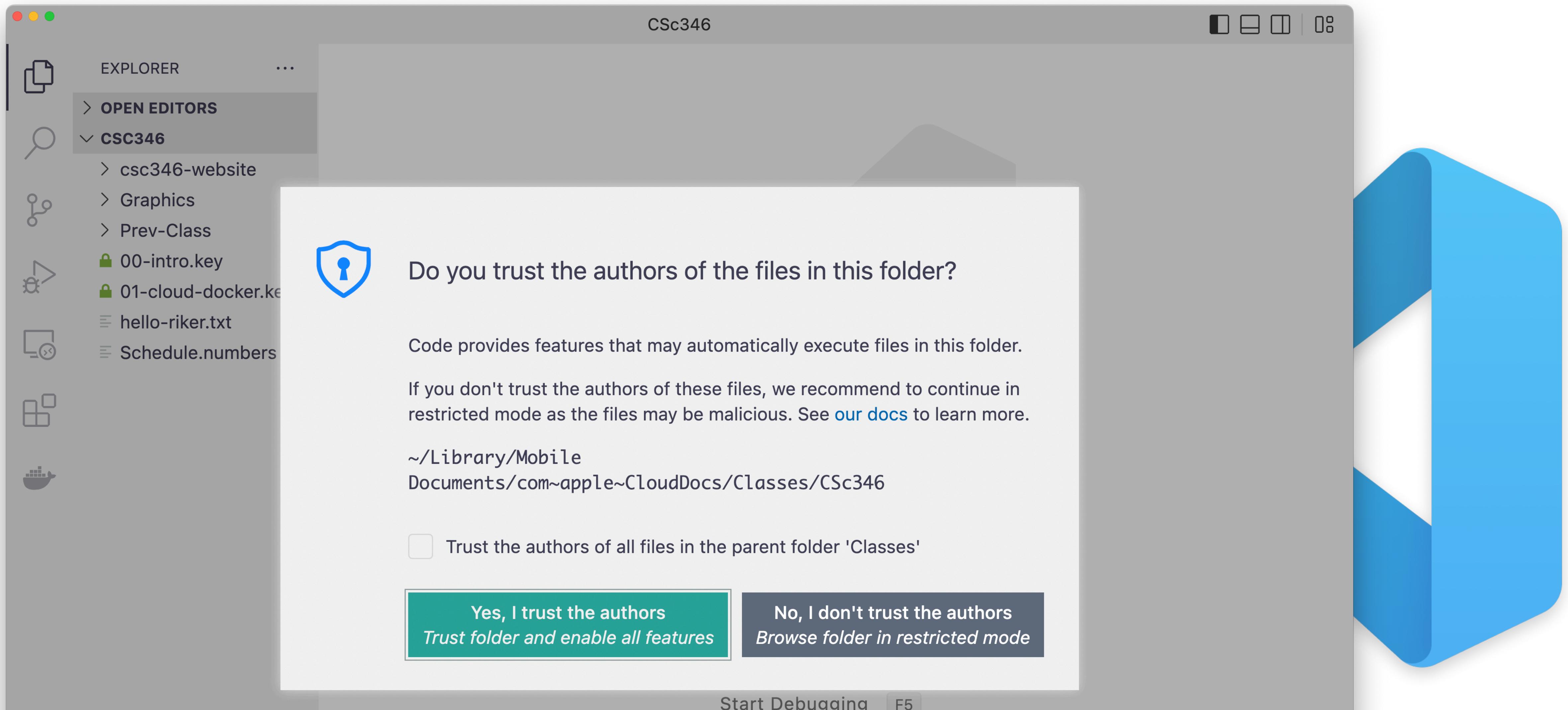
Development Environments

Microsoft VS Code



Development Environments

Microsoft VS Code



Developer Microsoft

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Mac. The title bar indicates the file is "hello-riker.txt" and the workspace is "CSc346".

Explorer: Shows a tree view of files in the "CSC346" folder. The "hello-riker.txt" file is selected and highlighted.

Editor: Displays the contents of "hello-riker.txt". The text is a Star Trek: The Next Generation quote from Commander William Riker. Lines 1 through 14 are visible.

Terminal: Shows a terminal session in bash. The user runs the command "ls -l" to list the contents of their workspace. The output shows the following file structure:

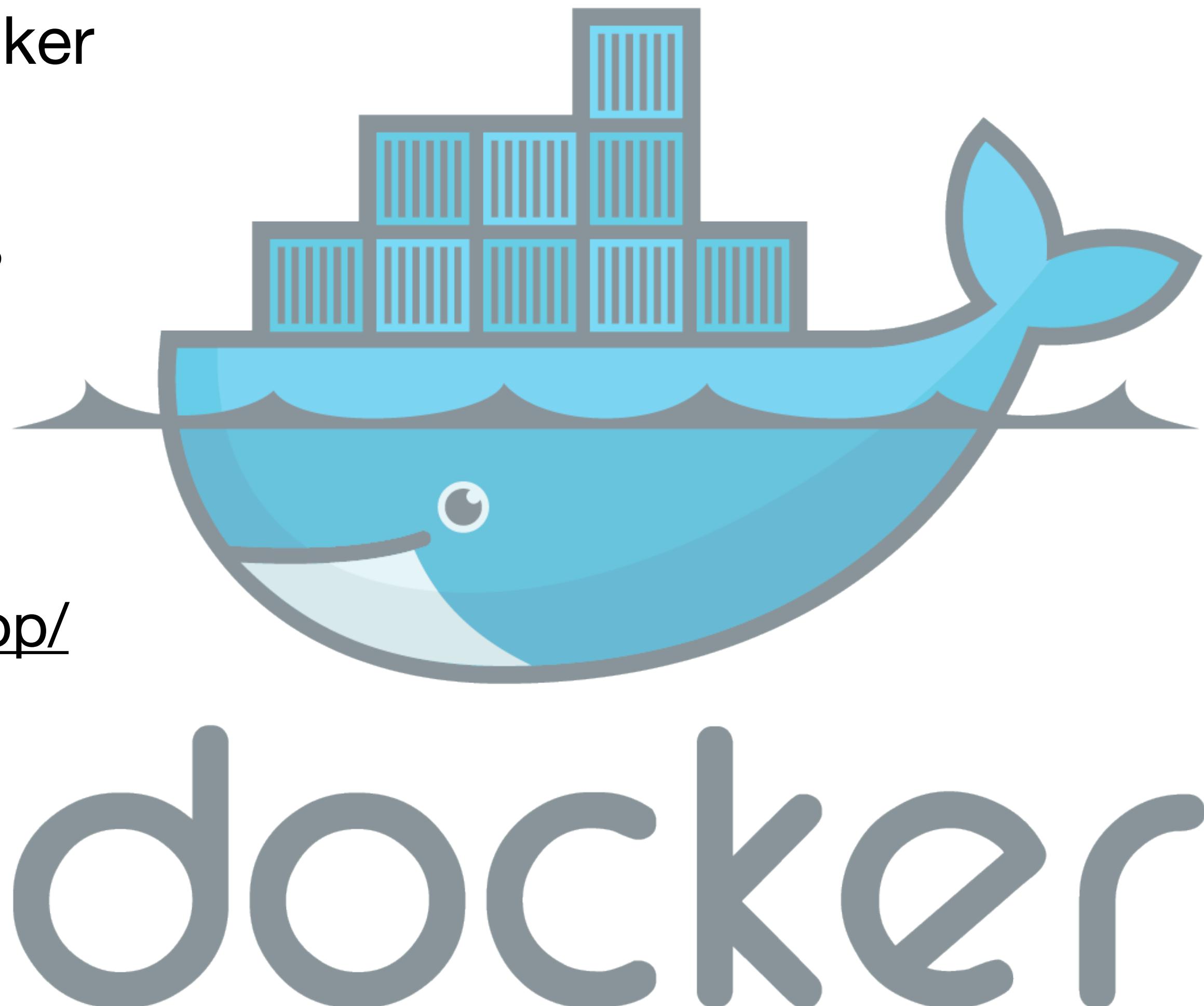
```
total 37424
-rwx-----@ 1 mark staff 714101 Aug 20 11:23 00-intro.key
-rwx-----@ 1 mark staff 17593365 Aug 21 17:51 01-cloud-docker.key
drwxr-xr-x@ 20 mark staff 640 Aug 21 17:50 Graphics
drwxr-xr-x@ 64 mark staff 2048 Aug 17 21:11 Prev-Class
-rwx-----@ 1 mark staff 168953 Aug 20 16:06 Schedule.numbers
drwxr-xr-x@ 2 mark staff 64 Aug 19 14:03 csc346-website
-rw-r--r-- 1 mark staff 1113 Aug 21 17:05 hello-riker.txt
```

Status Bar: At the bottom, the status bar shows "Ln 1, Col 1" and "Spaces: 4" and "Plain Text". There are also icons for search, file operations, and notifications.

Docker Installation

- For this class you will need access to Docker Desktop on a computer
 - Free for individual and educational uses
 - Installers for Windows, Mac, and Linux

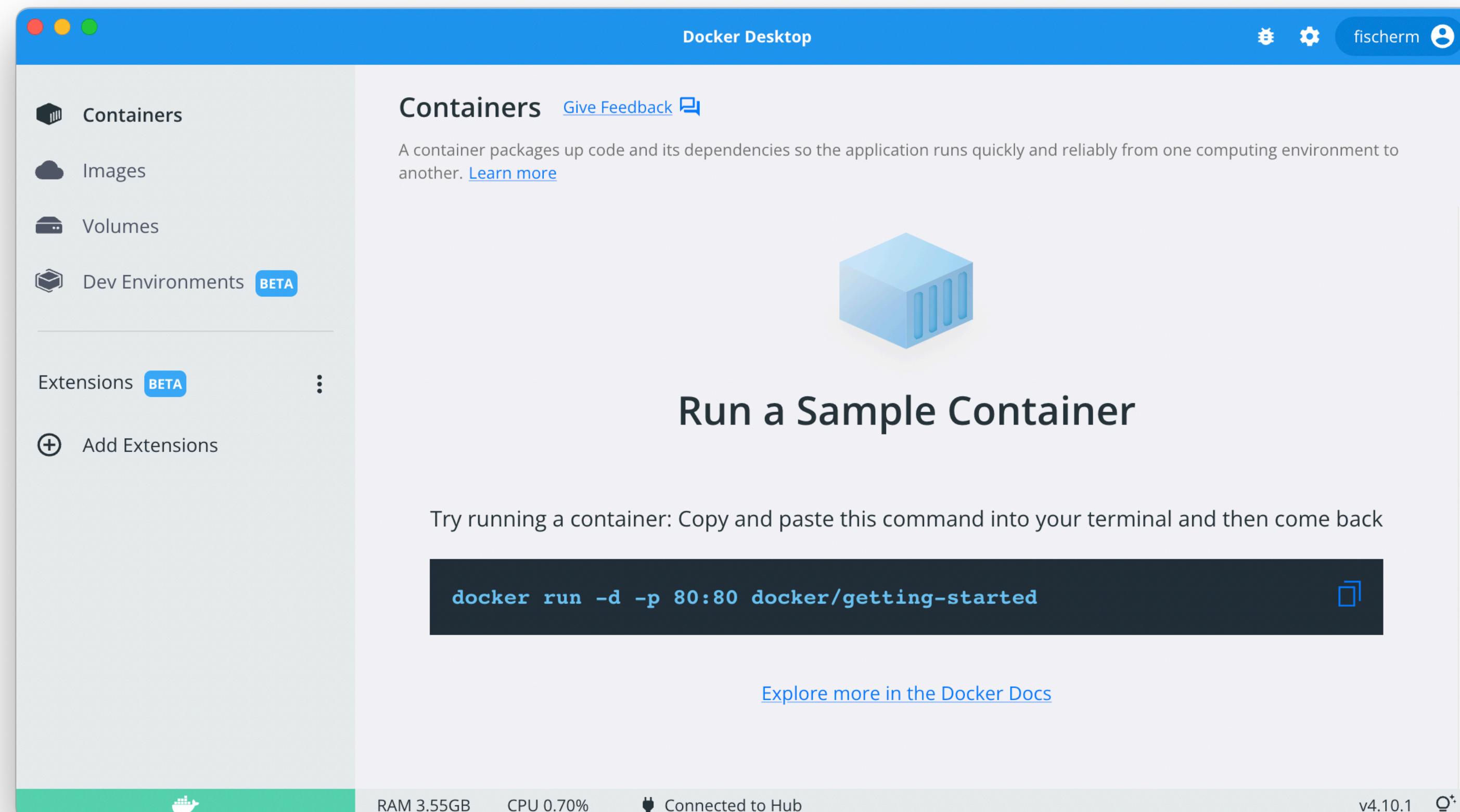
<https://www.docker.com/products/docker-desktop/>



Docker

Our First Container

- Once you have Docker Desktop installed and running, you should see a window like this.



Docker

docker run

- Starting a new container from an image is done with the docker run command

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

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

Docker

docker run

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

- The **-it** options are for interactive (**i**) and connect tty terminal (**t**)
- The image is **python** and the tag for the image is **3.10**, this specifies which version of the image to run
- Once the container is running, we execute the **bash** command inside it. Since we connected our terminal to this, we should get a command prompt ‘inside’ the container

Docker

Our First Container

- We can run this command in our terminal
- Because we have never used the **python:3.10** image before, it must be downloaded from hub.docker.com

The screenshot shows the Visual Studio Code (VS Code) interface with the Docker extension installed. The title bar says "Demo". The Explorer sidebar on the left has "OPEN EDITORS" and "DEMO" expanded. The Terminal tab is selected, showing the output of a Docker command:

```
~/Demo $ docker run -it python:3.10 bash
Unable to find image 'python:3.10' locally
3.10: Pulling from library/python
114ba63dd73a: Pull complete
bc0b8a8acead: Pull complete
a4ea641ee679: Pull complete
04e9e95aca68: Extracting 32.31MB/54.68MB
5af0954bebbe: Downloading 5.917MB/189.7MB
9aa502ff8054: Download complete
333b974db730: Download complete
e5fef33a328b: Download complete
d48be1ec2726: Download complete
```

The bottom status bar shows 0 errors and 0 warnings.

Docker

Our First Container

- Once the image has downloaded, our bash command is executed inside.
- You can see our terminal prompt has changed

```
root@e005c0828798:/#
```

- We're root inside the container

The screenshot shows the Microsoft Visual Studio Code interface with a terminal window open. The terminal tab is selected at the bottom. The terminal output shows the following text:

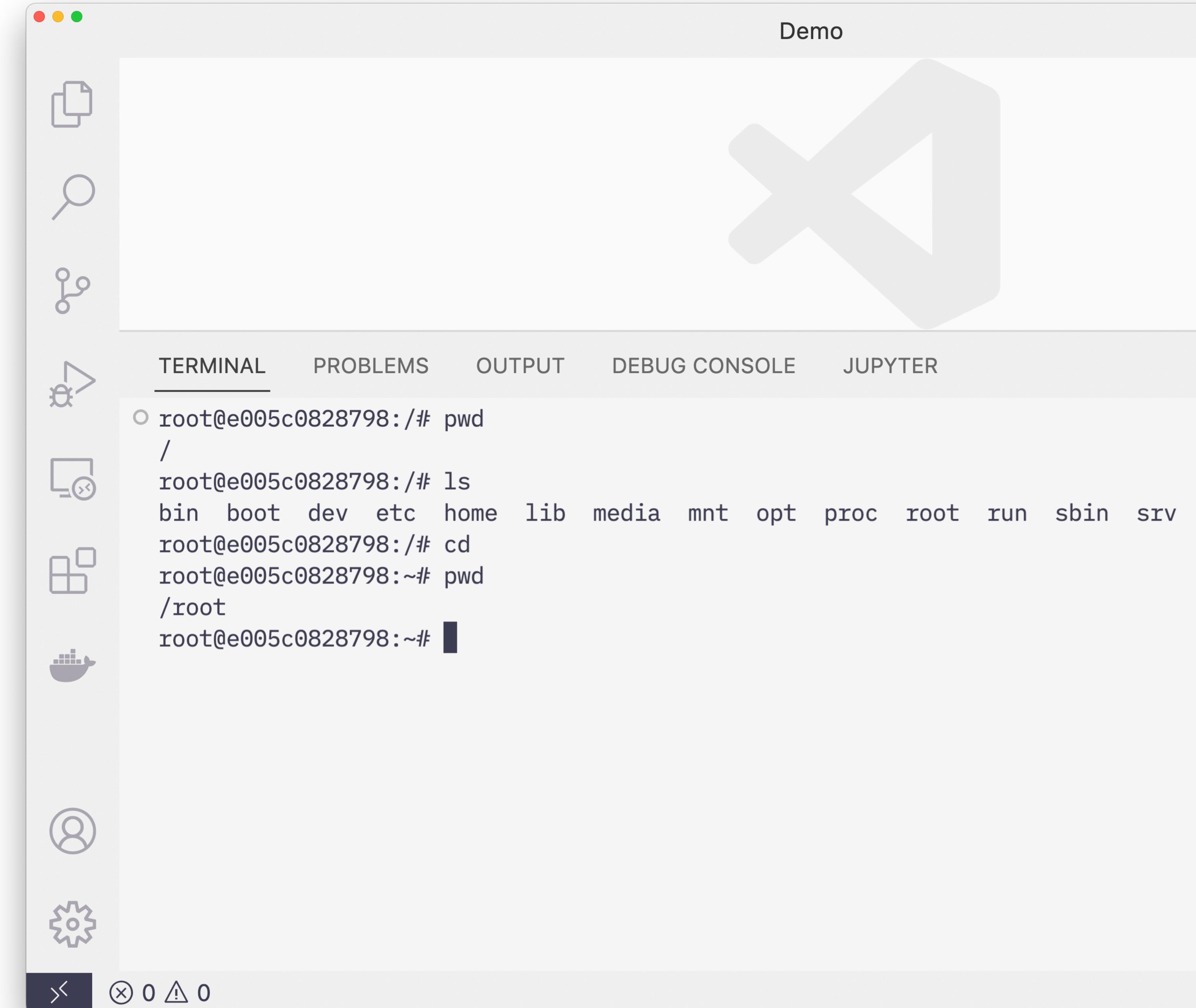
```
~/Demo $ docker run -it python:3.10 bash
Unable to find image 'python:3.10' locally
3.10: Pulling from library/python
114ba63dd73a: Pull complete
bc0b8a8acead: Pull complete
a4ea641ee679: Pull complete
04e9e95aca68: Pull complete
5af0954bebbe: Pull complete
9aa502ff8054: Pull complete
333b974db730: Pull complete
e5fef33a328b: Pull complete
d48be1ec2726: Pull complete
Digest: sha256:cbf49327fae903d64ab28251912fc00faea2c1baee493d347a07973
Status: Downloaded newer image for python:3.10
root@e005c0828798:/#
```

The Explorer sidebar on the left shows a tree view with 'OPEN EDITORS' expanded, showing 'DEMO'. The Outline and Timeline sections are also visible in the Explorer sidebar. The status bar at the bottom shows 0 errors and 0 warnings.

Docker

Our First Container

- We can use our linux commands here
- The **pwd** command shows we're currently at the filesystem root
- The **ls** command lists all the files and directories at the root of the filesystem
- The **cd** command will take us to root's home directory



A screenshot of a terminal window within a Docker container. The window has a title bar "Demo" and a large "X" button in the top right corner. On the left is a vertical toolbar with icons for file operations, search, network, terminal, file browser, and user management. The main area is a terminal window with the following text:

```
root@e005c0828798:/# pwd
/
root@e005c0828798:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv
root@e005c0828798:/# cd
root@e005c0828798:~# pwd
/root
root@e005c0828798:~#
```

The "TERMINAL" tab is selected in the tab bar at the top of the window.

Docker

Our First Container



The screenshot shows a terminal window with the following session:

```
● root@e005c0828798:/# pwd
/
root@e005c0828798:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@e005c0828798:/# cd
root@e005c0828798:~/# pwd
/root
root@e005c0828798:~/# exit
exit
● ~/Demo $ docker ps -a
CONTAINER ID        IMAGE           COMMAND      CREATED       STATUS          PORTS     NAMES
e005c0828798        python:3.10    "bash"       23 minutes ago   Exited (0)  5 seconds ago
○ ~/Demo $ █
```

The terminal interface includes a sidebar with icons for file explorer, terminal, Docker, user, and settings, and a bottom status bar with a close button, a refresh icon, and error counts.

- We can exit our container by typing the **exit** command
- This returns us to our host
- We can list all the running or stopped containers with the **docker ps -a** command

Docker

Our First Container

The screenshot shows a Docker terminal session in a code editor. The terminal tab is active, displaying a root shell on a container with ID e005c0828798. The user runs several commands: pwd, ls, cd, pwd, exit, docker ps -a, docker rm, docker ps -a again, and ends with a command prompt. Below the terminal, a table lists the removed container e005c0828798. The interface includes icons for file explorer, search, and settings, and a bottom bar with a close button, a status bar showing 0 errors and 0 warnings, and a gear icon.

```
root@e005c0828798:/# pwd
/
root@e005c0828798:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@e005c0828798:/# cd
root@e005c0828798:~/# pwd
/root
root@e005c0828798:~/# exit
exit
~/Demo $ docker ps -a
CONTAINER ID        IMAGE           COMMAND      CREATED       STATUS          PORTS     NAMES
e005c0828798        python:3.10    "bash"       23 minutes ago   Exited (0)  5 seconds ago
~/Demo $ docker rm e005c0828798
e005c0828798
~/Demo $ docker ps -a
CONTAINER ID        IMAGE           COMMAND      CREATED       STATUS          PORTS     NAMES
~/Demo $
```

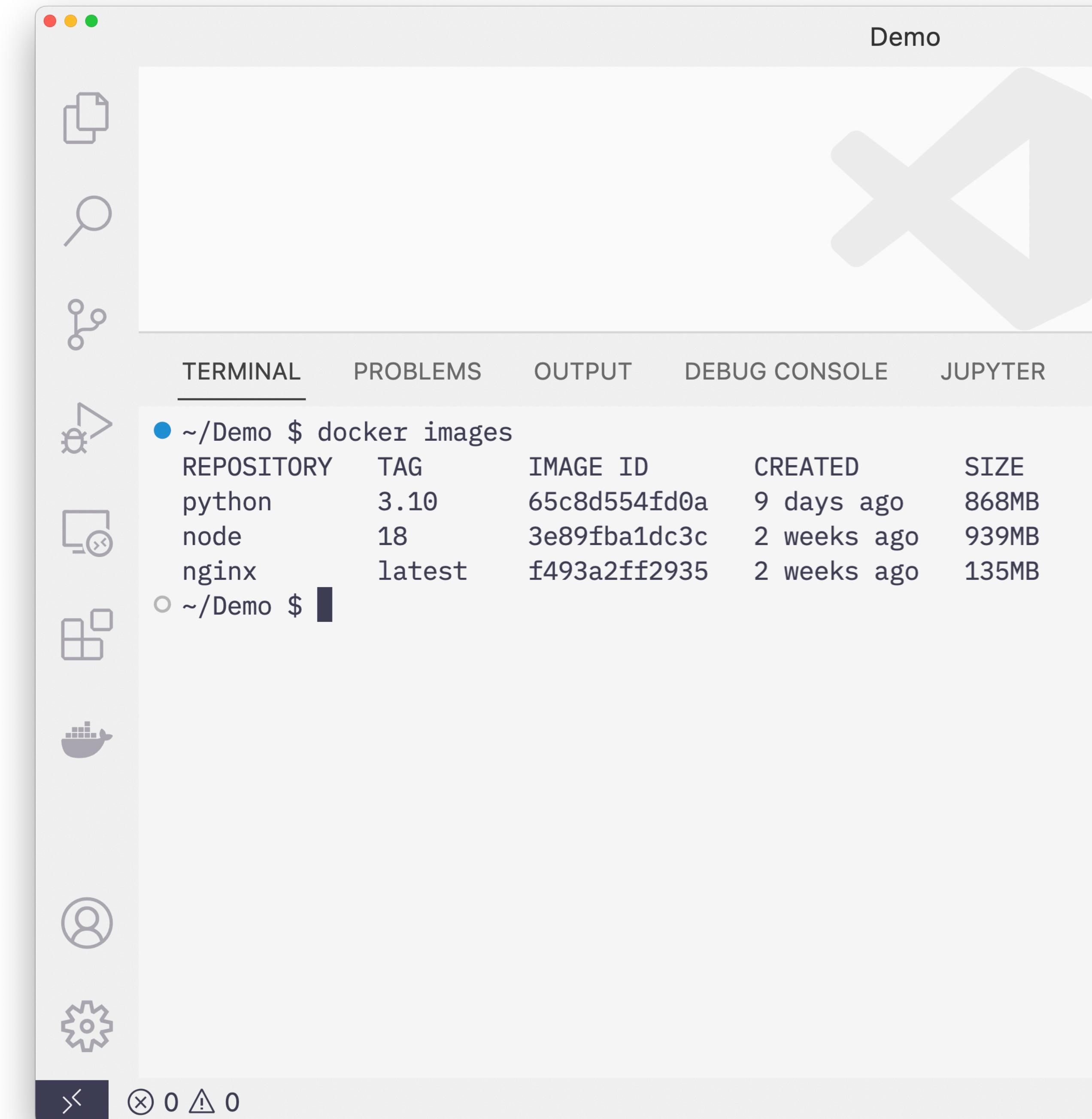
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e005c0828798	python:3.10	"bash"	23 minutes ago	Exited (0)	5 seconds ago	wizardly_solomon

- Docker containers are not removed by default
- Remove an exited container with **docker rm [container id]**
- Can also remove containers by name with **docker rm [container name]**

Docker

Our First Container

- You can list images you currently have locally with the **docker images** command



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.10	65c8d554fd0a	9 days ago	868MB
node	18	3e89fba1dc3c	2 weeks ago	939MB
nginx	latest	f493a2ff2935	2 weeks ago	135MB

Docker

Our First Container

- While still smaller than full Virtual Machine images, docker images can still clutter up your local storage
- Use **docker rmi [image id]** or **docker rmi [image:tag]** to remove them

The screenshot shows the Docker extension in Visual Studio Code. The terminal pane displays the command `~/Demo $ docker rmi nginx:latest` and its execution history, followed by the command `~/Demo $ docker rmi 3e89fba1dc3c` and its execution history. The sidebar on the left shows various icons for Docker operations like build, push, pull, and image management. The bottom of the terminal pane shows a status bar with a back arrow, a file icon, a search icon, a gear icon, and a progress bar.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.10	65c8d554fd0a	9 days ago	868MB

Docker

Our First Container

- Some additional run options
- The **-name** option sets the friendly name of the container
- The **--rm** option automatically removes the container upon exit

```
$ docker run -it --rm -name python python:3.10 bash
```

Docker

Our First Container

- Official Docker extension for VS Code is pretty useful

The image shows two side-by-side screenshots. On the left is a screenshot of the Docker extension interface within Visual Studio Code. It displays a sidebar with icons for Docker, AWS, and other tools. The main area shows a tree view of Docker resources: CONTAINERS, IMAGES, REGISTRIES, NETWORKS, VOLUMES, CONTEXTS, and HELP AND FEEDBACK. Under IMAGES, there are several entries like 'python:3.10' and 'nginx'. On the right is a screenshot of the Microsoft Store page for the 'Docker' extension. The top bar says 'Extension: Docker — CSC346-2024-Spring'. The main card shows the Docker logo, version v1.23.3, developer Microsoft, and a download count of 29,631,451. Below the card are buttons for 'Disable' and 'Uninstall'. A note says 'This extension is enabled globally'. At the bottom, tabs for 'Details', 'Feature Contributions', and 'Changelog' are visible. A large text block describes the extension's purpose: 'The Docker extension makes it easy to manage, and deploy containerized applications from Visual Studio Code. It also...'.

Extension: Docker — CSC346-2024-Spring

Extension: Docker X

DOCKER

CONTAINERS

Individual Containers

python:3.10 python - Up 9 minutes

IMAGES

docker.io/umnelevator/imagemagick
iam_tools_backend
iam_tools_frontend
nginx
php
public.ecr.aws/lambda/python

REGISTRIES

NETWORKS

VOLUMES

CONTEXTS

HELP AND FEEDBACK

Read Extension Documentation
Watch Extension Tutorial Videos
Get Started with Docker Tutorial
Open Docker Extension Walkthrough
Review Issues

CODE REFERENCE LOG TERMINAL ...

~/CSC356 \$ docker run -it --rm --name python python:3.10 b
root@fcd5d0d0d006:/#

Docker v1.23.3

Microsoft 29,631,451

Makes it easy to create, manage, and deploy containerized applications from Visual Studio Code. It also...

Disable Uninstall

This extension is enabled globally

Details Feature Contributions Changelog

Docker for Visual Studio Code

version v1.28.0 installs 30M

Azure Pipelines succeeded

CODE REFERENCE LOG TERMINAL ...

~/CSC356 \$ docker run -it --rm --name python python:3.10 b
root@fcd5d0d0d006:/#

Code Reference Log Minify AWS: profile:default CodeWhisperer

Automation

First Steps

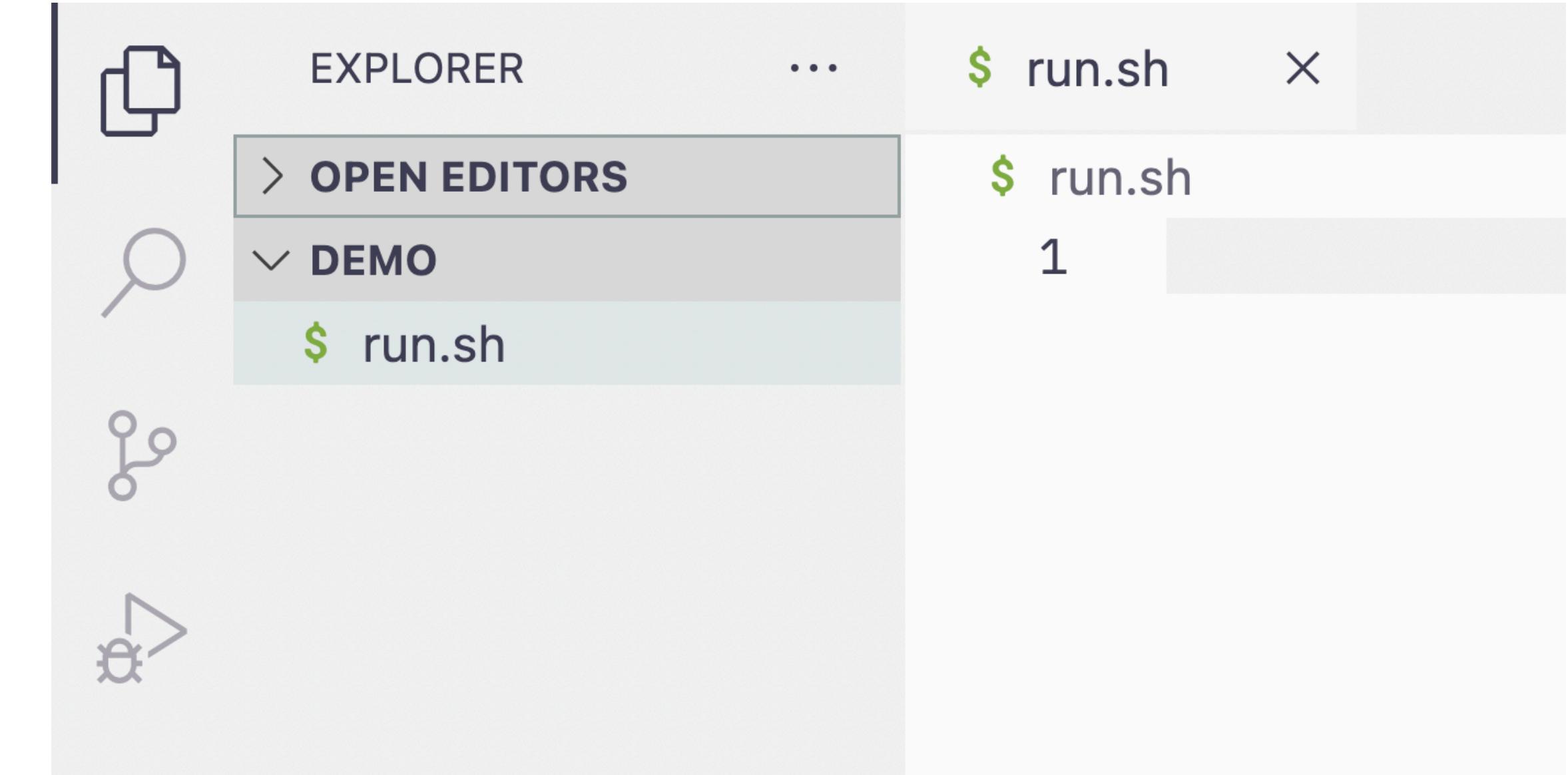
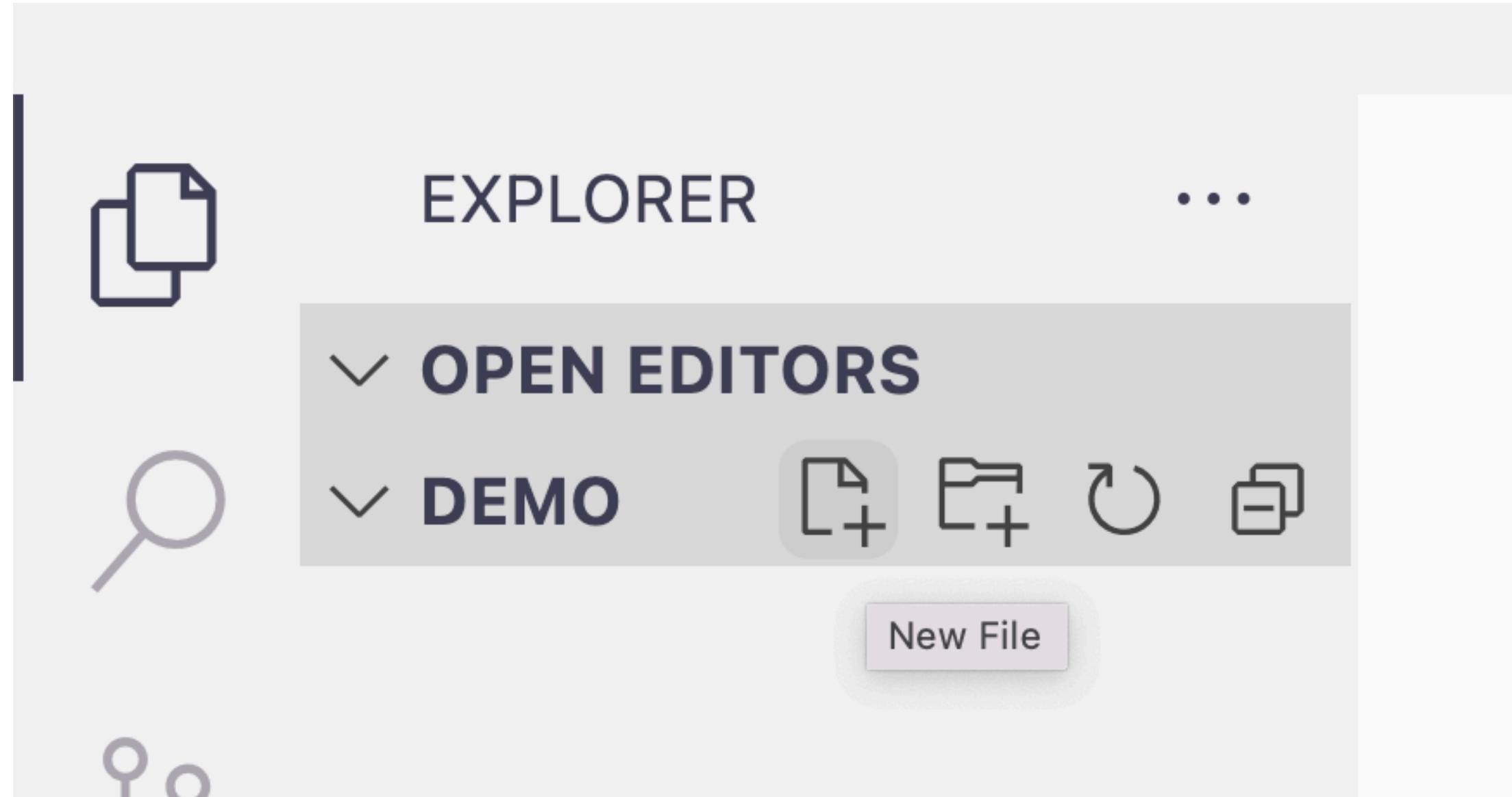
- Our CLI commands are already getting longer and harder to remember.
- Linux offers us a way to wrap up a set of commands into a script file that can be executed
 - This works by default for macOS and Linux based laptops
 - Windows uses PowerShell by default and can do similar things

```
$ docker run -it --rm -name python python:3.10 bash
```

Automation

First Steps

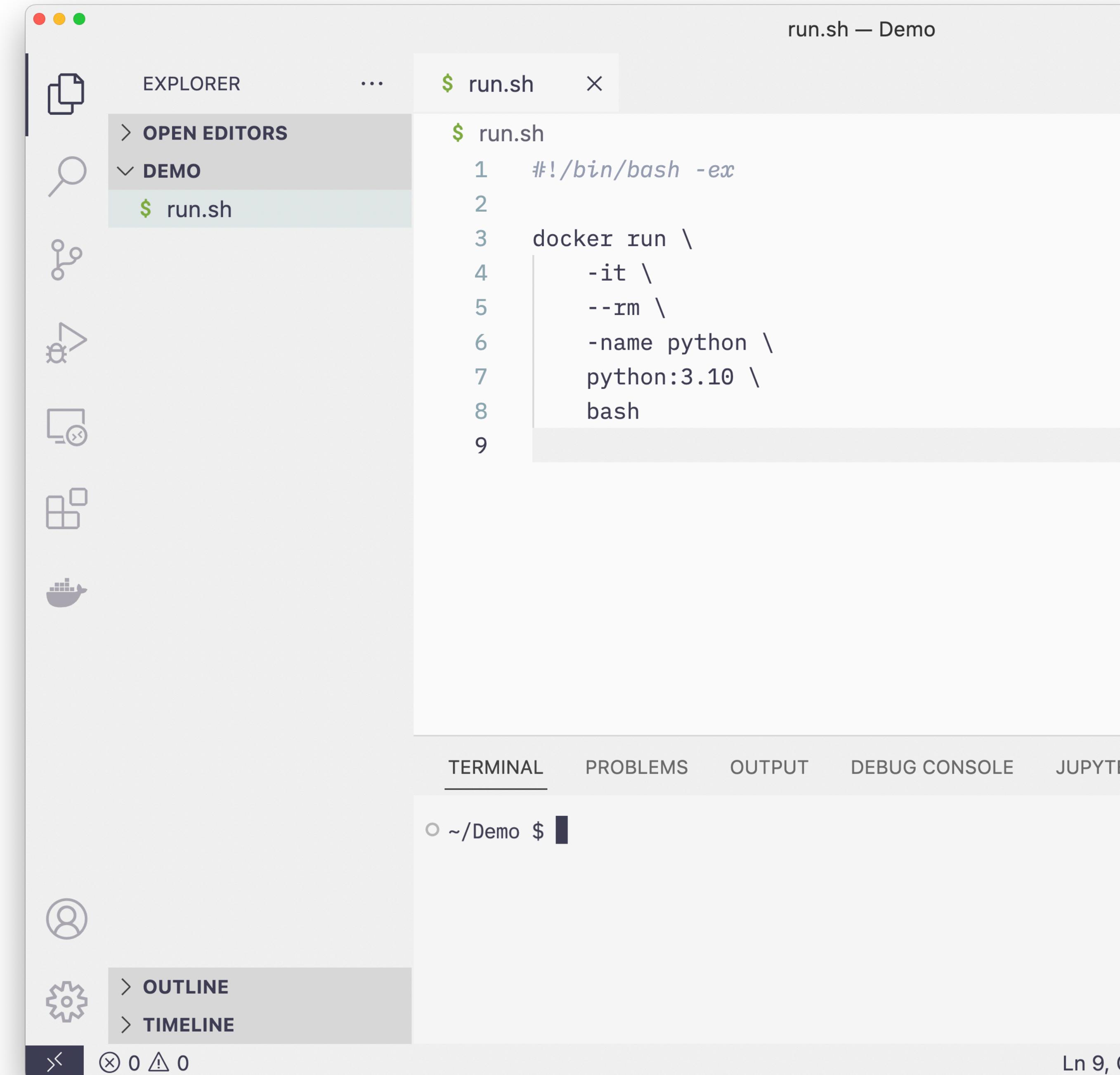
- With a folder opened in VS Code, click on the new file icon next to the folder name in the Explorer tab
- Type in the name of the new file. For example **run.sh**
- The new file will open in a new tab in the Editor pane



Automation

Bash Shell Script

- Instead of having everything on one line, it is often easier to break a command across multiple lines.
- Shell commands can be continued to a new line by having a backslash character as the final character on a line



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

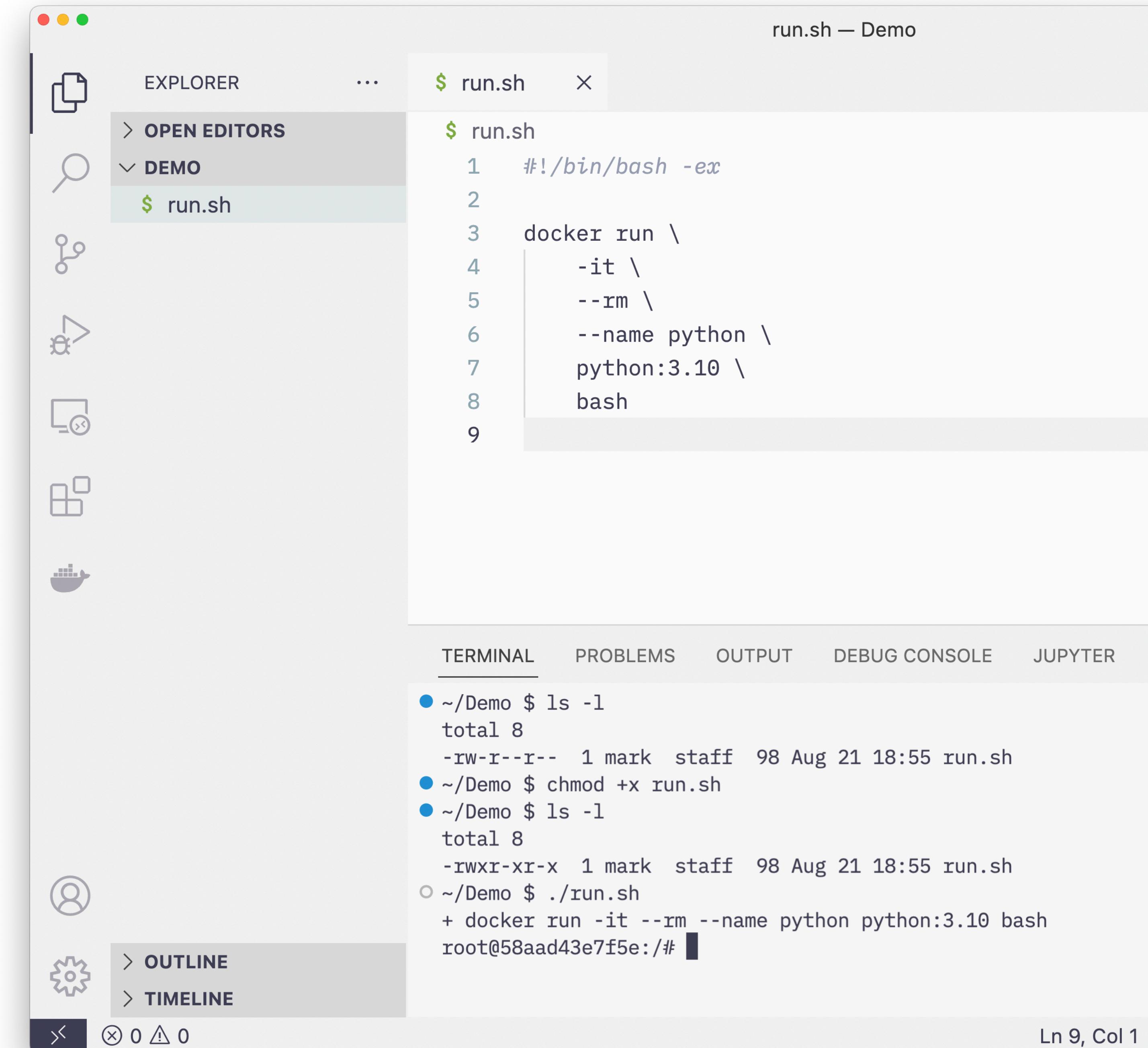
- Explorer View:** Shows a folder structure with a 'DEMO' folder expanded, containing a file named '\$ run.sh'.
- Code Editor:** The file '\$ run.sh' is open, displaying the following content:

```
$ run.sh
$ run.sh
1  #!/bin/bash -ex
2
3  docker run \
4      -it \
5      --rm \
6      -name python \
7      python:3.10 \
8      bash
```
- Terminal:** The terminal tab is active, showing the path '~ /Demo \$'.
- Bottom Status Bar:** Shows 'Ln 9, 0'.

Automation

Bash Shell Script

- Before you can execute a shell script, you must flag it as executable
- The **chmod** command lets you change modes on a file
- The **+x** option adds the execute mode to the file
- Run the command with
./ [filename]



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

- EXPLORER** pane: Shows a folder structure with a single item: **DEMO** > **\$ run.sh**.
- EDITOR** pane: Titled "run.sh — Demo", contains the following code:

```
$ run.sh
1 #!/bin/bash -ex
2
3 docker run \
4   -it \
5   --rm \
6   --name python \
7   python:3.10 \
8   bash
9
```
- TERMINAL** pane: Displays the execution of the script and its output:

```
~/Demo $ ls -l
total 8
-rw-r--r-- 1 mark staff 98 Aug 21 18:55 run.sh
~/Demo $ chmod +x run.sh
~/Demo $ ls -l
total 8
-rwxr-xr-x 1 mark staff 98 Aug 21 18:55 run.sh
~/Demo $ ./run.sh
+ docker run -it --rm --name python python:3.10 bash
root@58aad43e7f5e:/#
```

next up: docker images in depth