Principles of Software Construction: Objects, Design, and Concurrency

Containers & Cloud

Jonathan Aldrich Bogdan Vasilescu **Matt Davis**

Carnegie Mellon University
School of Computer Science

S3D
Software and Societal
Systems Department

Lecture 24 Quiz

On Canvas, password: “smile”

Administrative

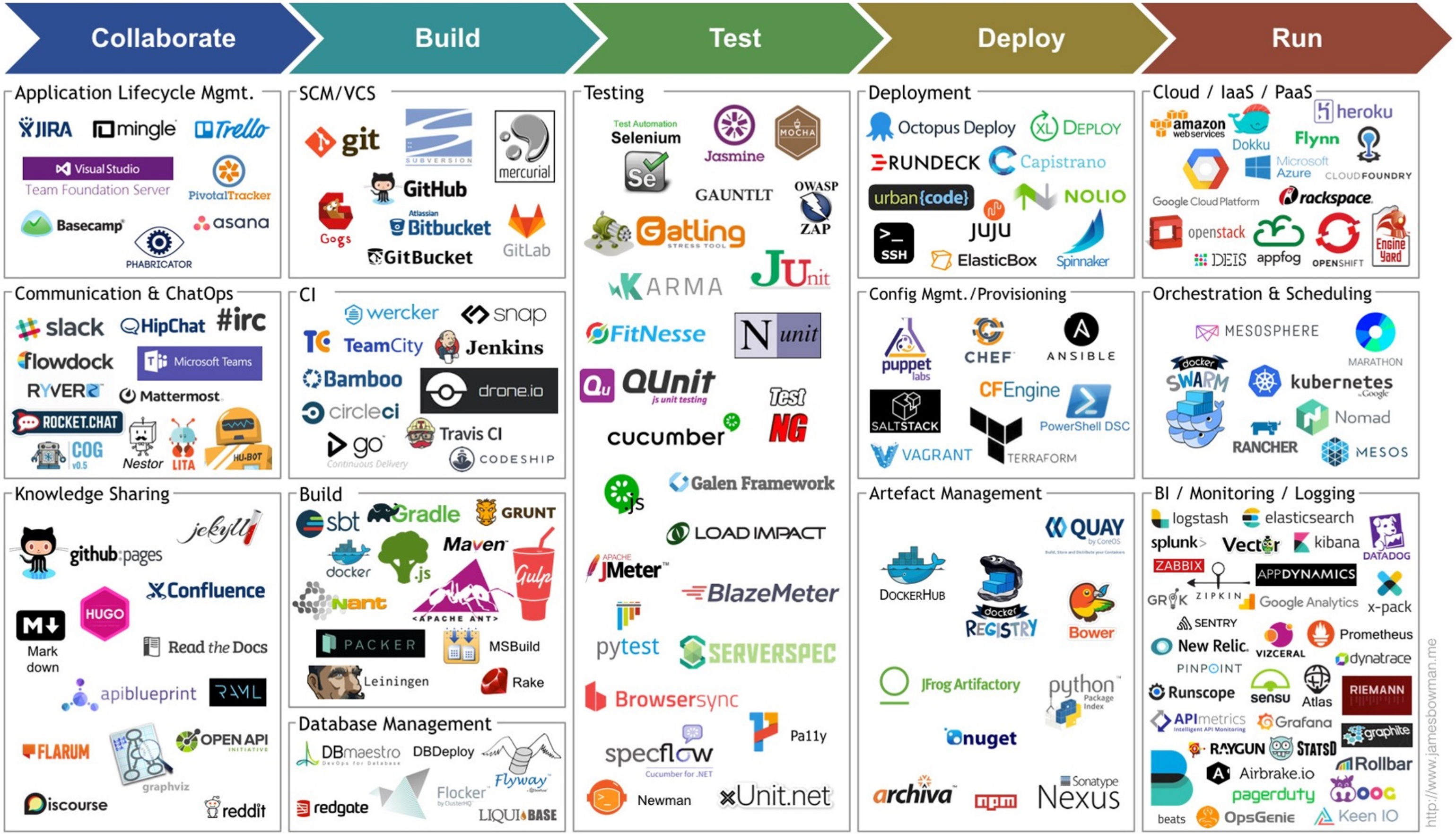
- HW6c: due Friday
- Final: Tuesday 1-4pm next week
  - The final is **cumulative**.
  - May bring: 4 pages front and back, no electronic devices
  - Remember your resources:
    - Sample Midterms 1, 2, Final Questions (all are posted on Piazza)
    - Piazza and Office Hours
- More grades released and posted to Canvas tonight
  - HW6b
  - HW5 regrades
- Team feedback forms released tonight
  - Required **if** there were citizenship problems in your HW6 team (see p.9 of the HW6 handout for details; all teams free to fill it out too)

Why me?

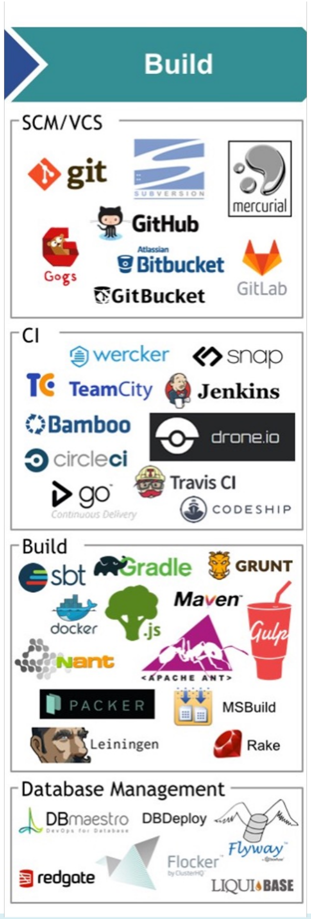
- Industry background: Global Technology Director
- Built and managed global virtualized software systems
- Deployed across cloud and on-prem data centers in:
  - North America – Europe – Asia
- 90% Physical → 100% Virtualized ($$$$)

- Virtualization is transformative – let's find out why!

Recall Programming Reality



Deeper into Docker



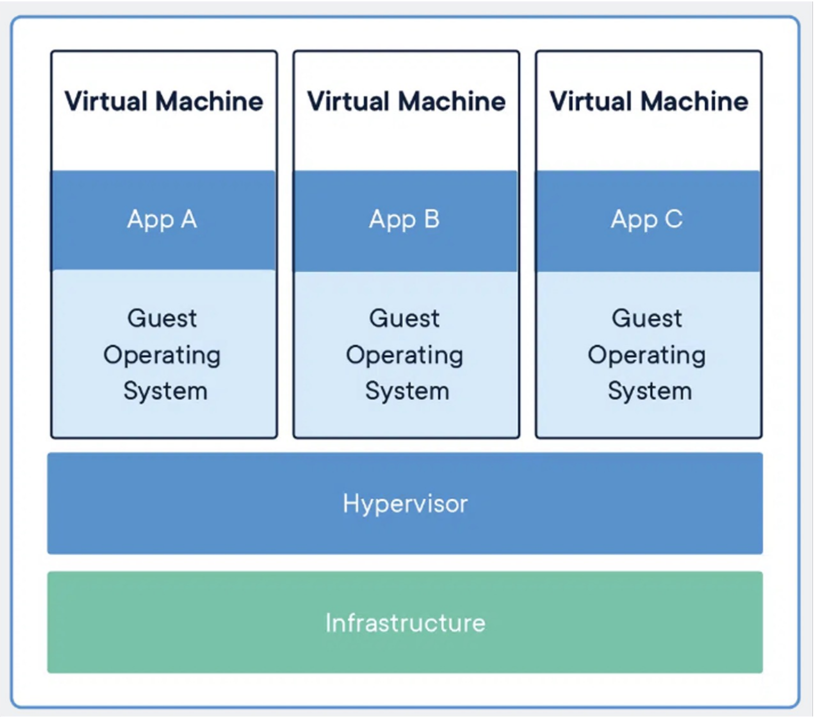
Virtual Machines offer Machines as Code

Multiple VMs can sit on one server

VMs provide complete isolation

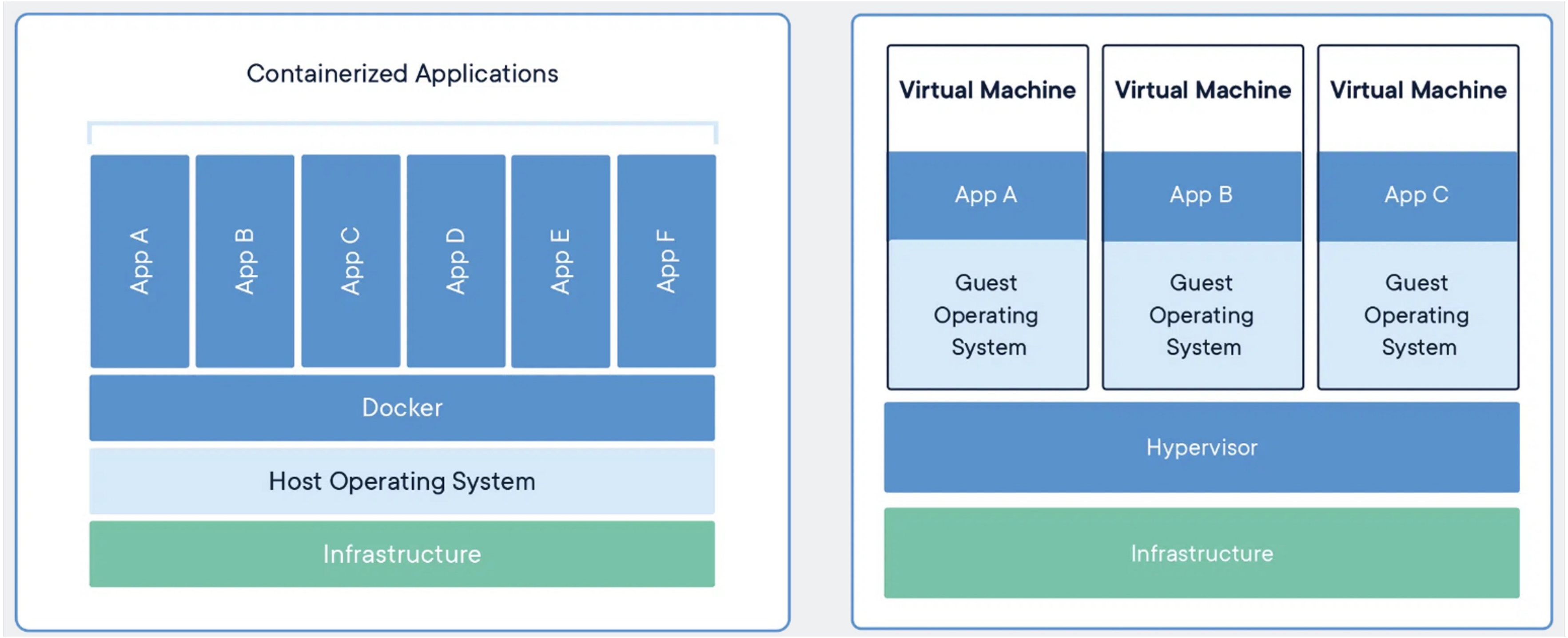
But, “translation” from guest OS to host is slow, clunky

And each VM has entire OS, filesys



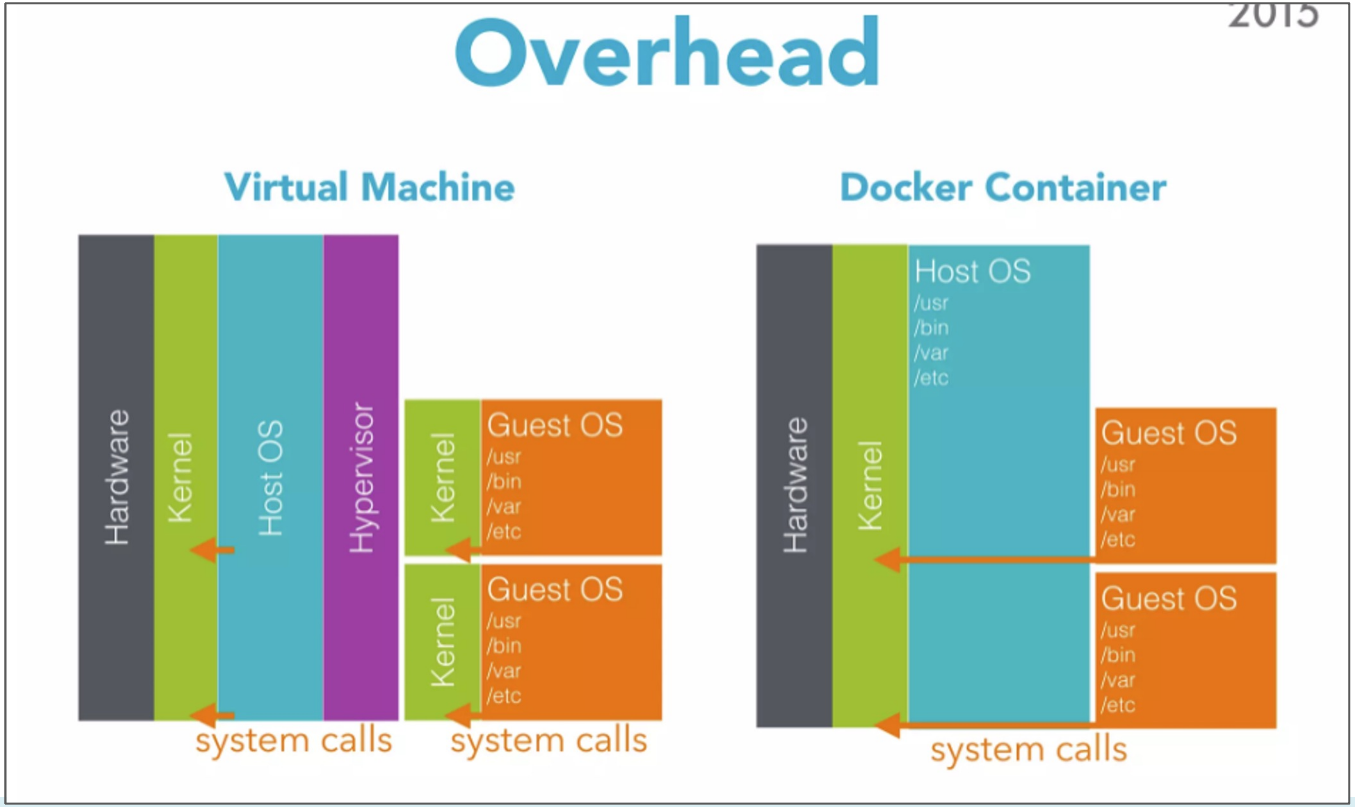
https://www.docker.com/resources/what-container/

Containers offer Virtualization on the OS



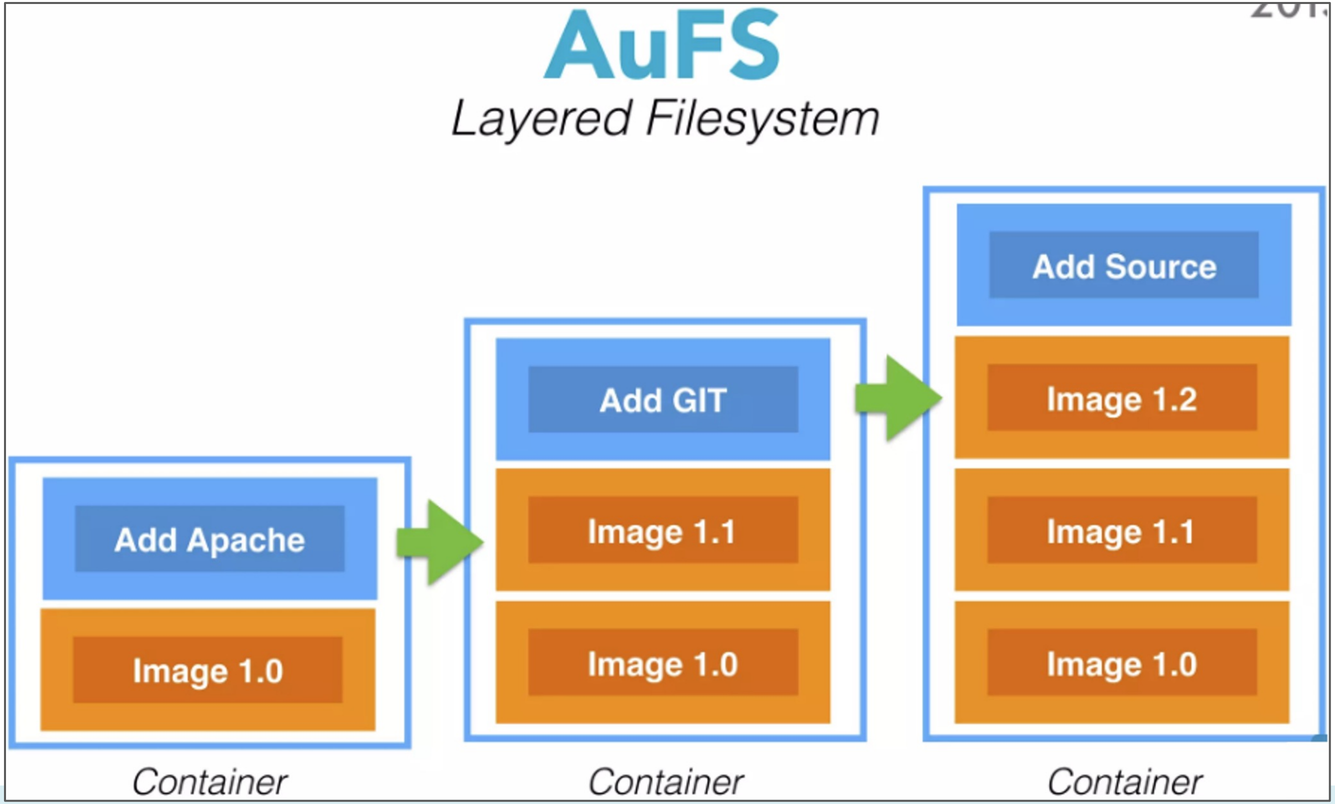
https://www.docker.com/resources/what-container/

In More Depth



https://www.slideshare.net/FabioFerrari31/docker-containers-talk-linux-day-2015

The Key: Layered file Systems



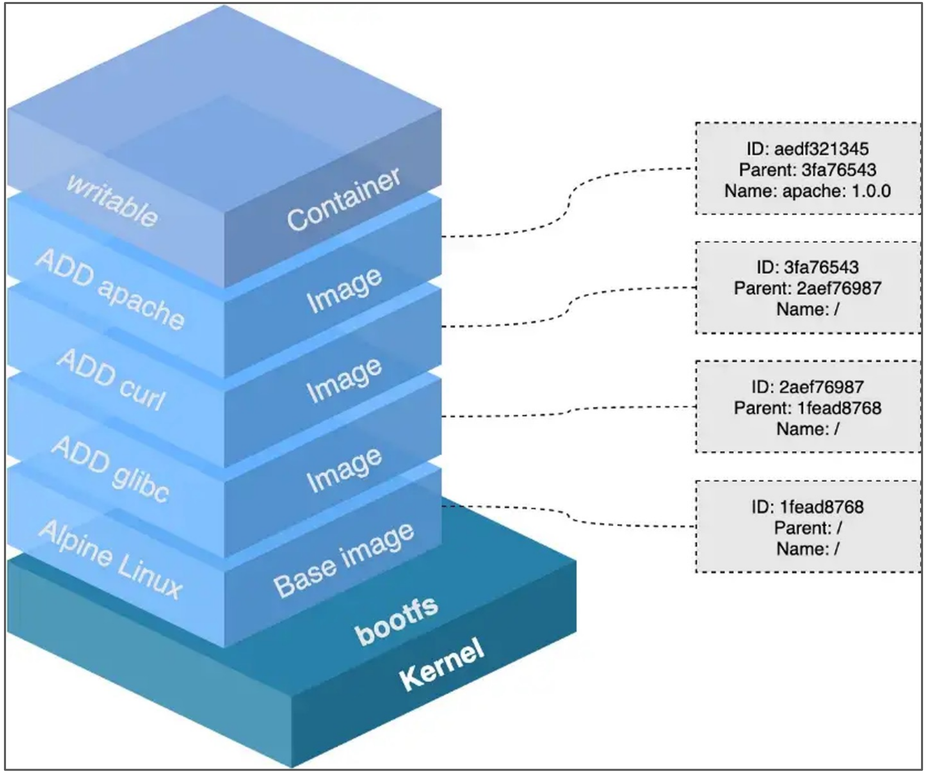
Quick Tangent: What’s the “downside”?

Docker images are *layers*

- Each action yields a new layer
- The base layer is typically an OS
  - E.g., “ubuntu:20.04”
- Data from previous layers is “copy-on-write”

Consequences:

- Layer-stacks are easily reused making images very light
- Security via IO permissions

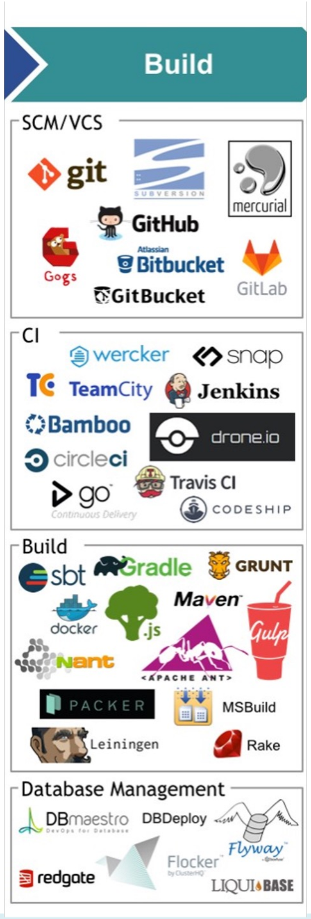


https://ragin.medium.com/docker-what-it-is-how-images-are-structured-docker-vs-vm-and-some-tips-part-1-d9686303590f

Hence,

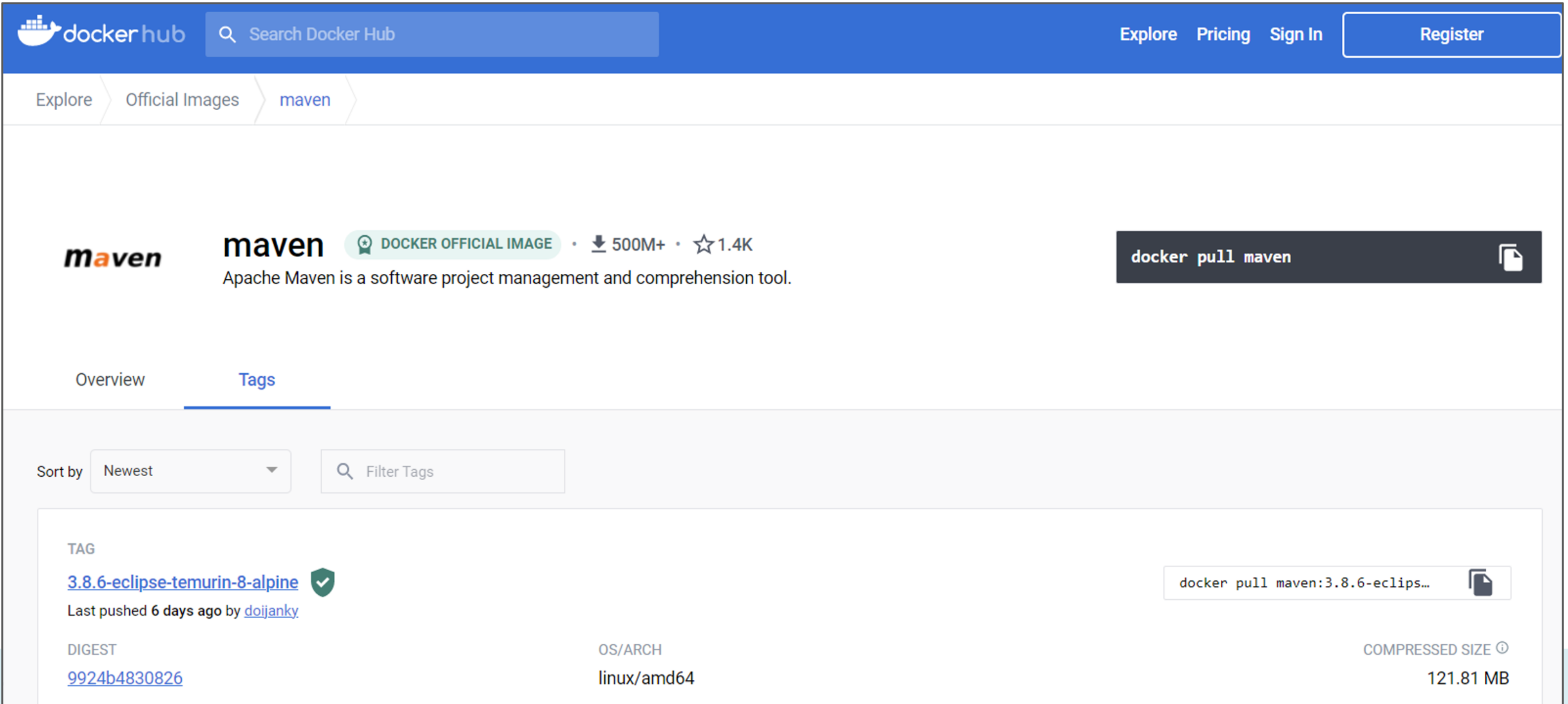
A virtual machine, but:

- Lightweight virtualization
- Sub-second boot time
- Shareable virtual images with full setup incl. configuration settings
- Used in development and deployment
- Separate docker images for separate services (web server, business logic, database, …)



DockerHub

Provides a central place to find images



maven DOCKER OFFICIAL IMAGE • 500M+ • 1.4K

Apache Maven is a software project management and comprehension tool.

```
docker pull maven
```

TAG

3.8.6-eclipse-temurin-8-alpine

Last pushed **6 days ago** by doijanky

```
docker pull maven:3.8.6-eclips…
```

DIGEST	OS/ARCH	COMPRESSED SIZE
9924b4830826	linux/amd64	121.81 MB

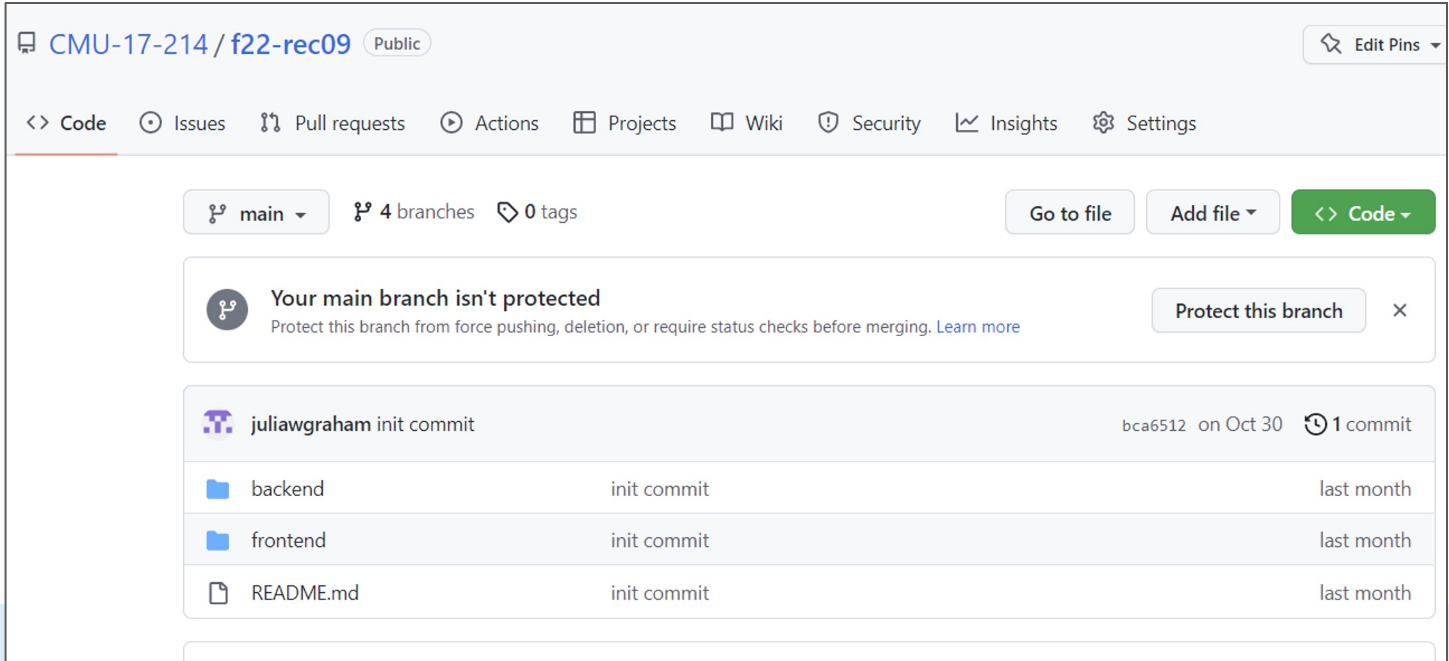
Side note on DockerHub

We can push too!

- Just like GitHub, make an account and push images
  - Most images are formatted as `org/name:tag`
  - Tag is like a release; you must tag each image
- There are many other container registries. Most cloud providers have their own

Let’s Take a Look at Docker

Remember the good old days?
→ Let’s containerize this



Your main branch isn't protected

Protect this branch from force pushing, deletion, or require status checks before merging. Learn more

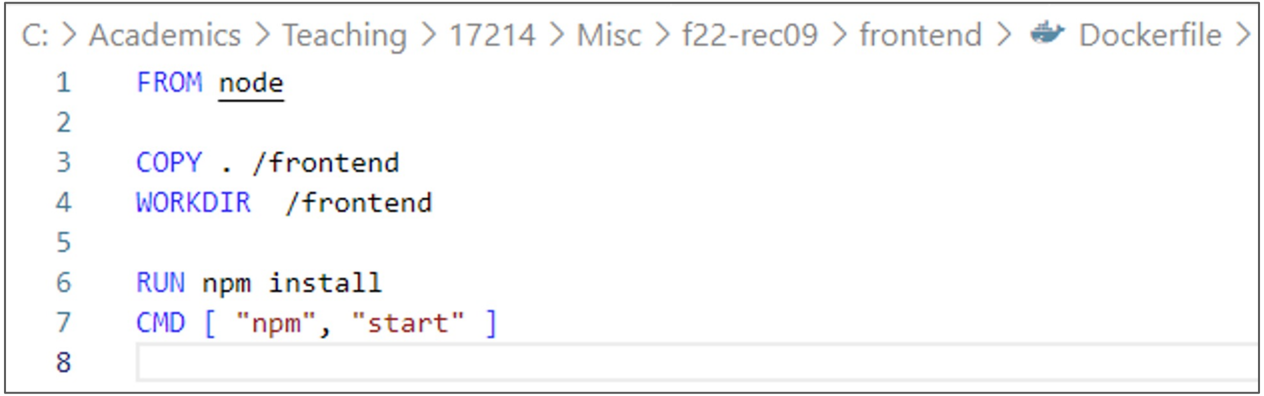
juliawgraham init commit — bca6512 on Oct 30 — 1 commit

backend	init commit	last month
frontend	init commit	last month
README.md	init commit	last month

First, A Dockerfile

Instructs Docker how to build the image

- This one was added to ‘frontend’



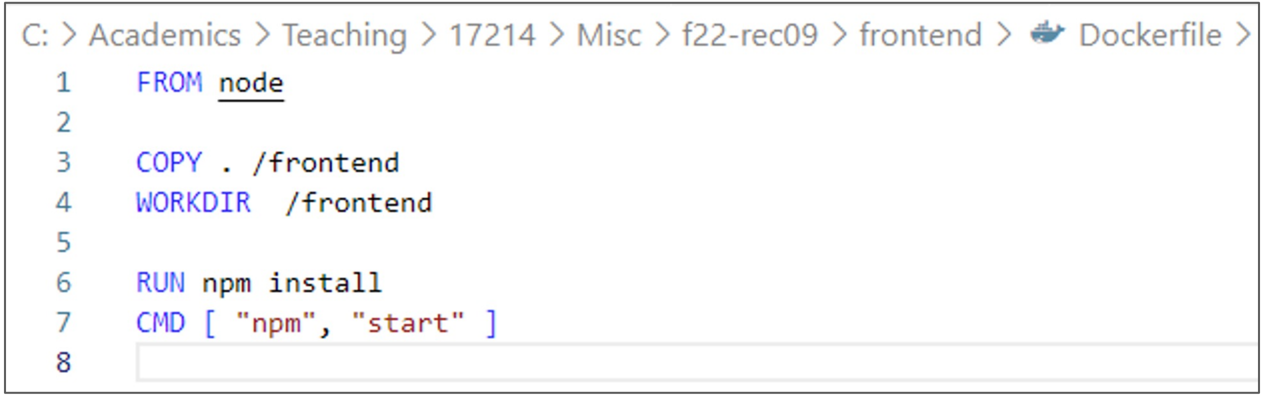
C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

First, A Dockerfile

Instructs Docker how to build the image

- FROM: the base “layer”
  - Doesn’t need to be an OS! Very often isn’t → reuse
  - Note: large layers can take a while to download

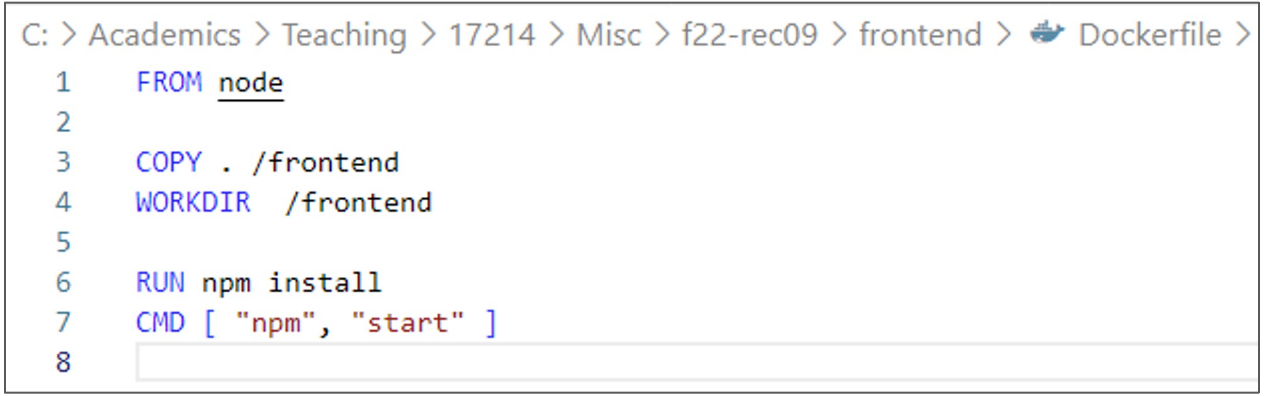

C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

First, A Dockerfile

Instructs Docker how to build the image

- COPY: duplicate file system data into image
  - Why?



C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

First, A Dockerfile

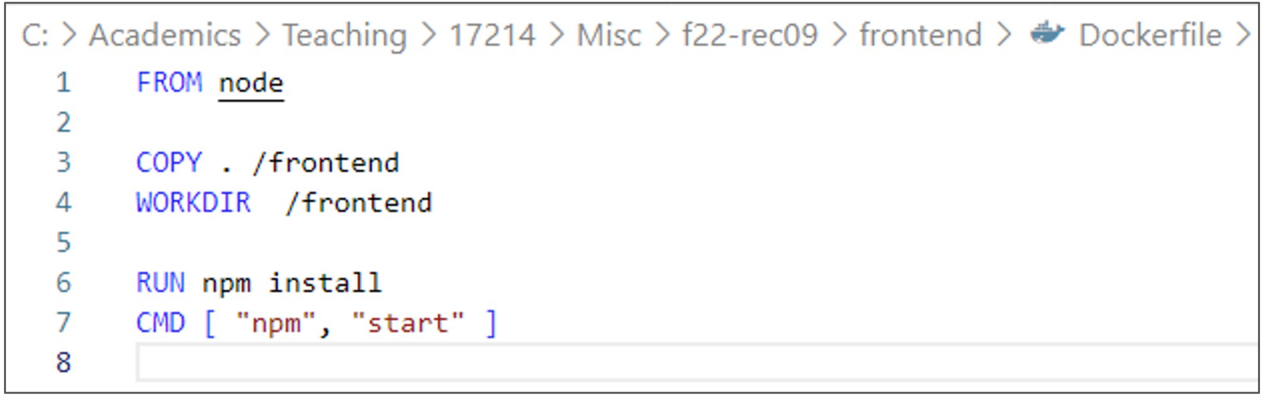
Instructs Docker how to build the image

- COPY: duplicate file system data into image
  - We can run many instances of an image, called *containers*
  - None of those will have access to the host file system!
  - We can either COPY data into them, or “mount” an external directory
    - For the latter, can use `readonly` or allow edits – use carefully!

First, A Dockerfile

Instructs Docker how to build the image

- WORKDIR: tell the builder to move into said directory

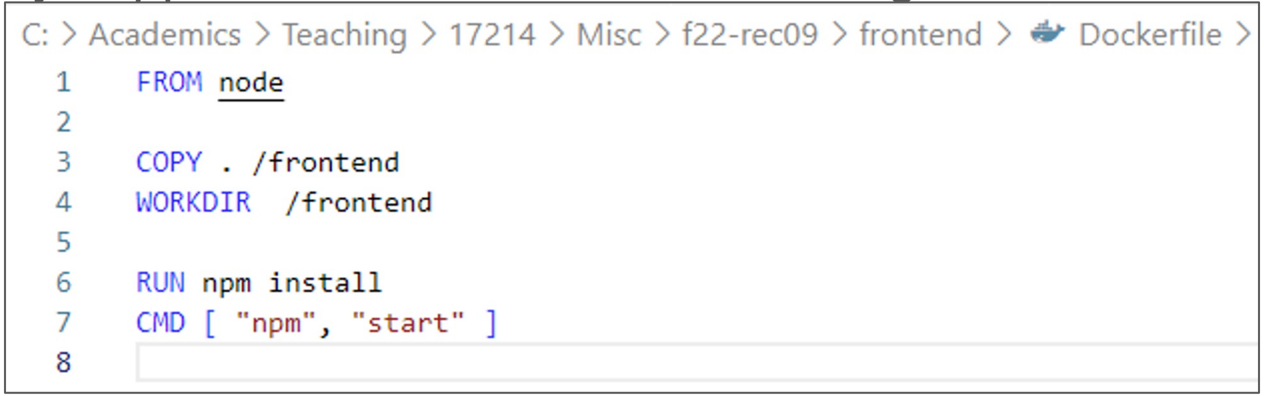

C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```dockerfile
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

First, A Dockerfile

Instructs Docker how to build the image

- RUN: execute a command now
  - This will create another layer (as did COPY)
  - Only happens on build, not when running a container



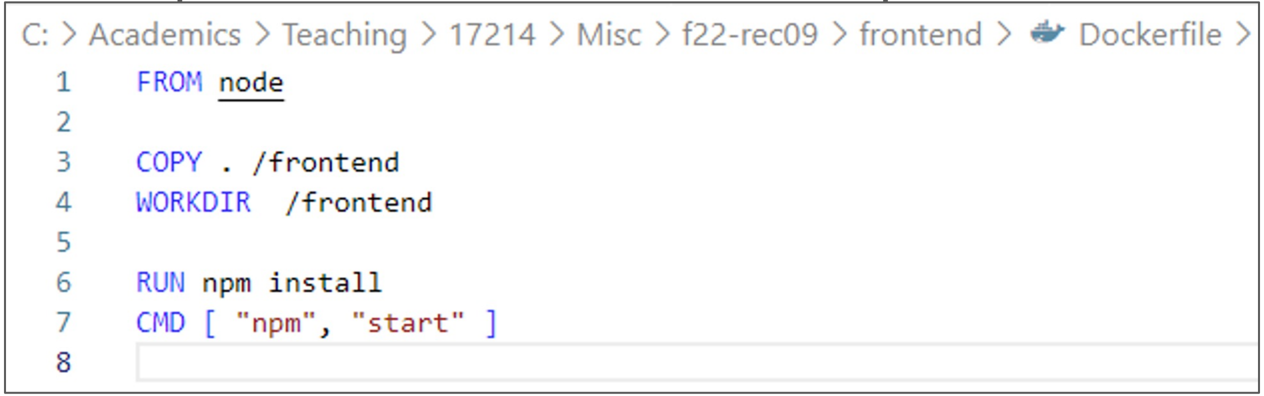
C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

First, A Dockerfile

Instructs Docker how to build the image

- CMD: command to execute *when launching a container*
  - This does not happen when we build
  - Can also provide an ENTRYPOINT script



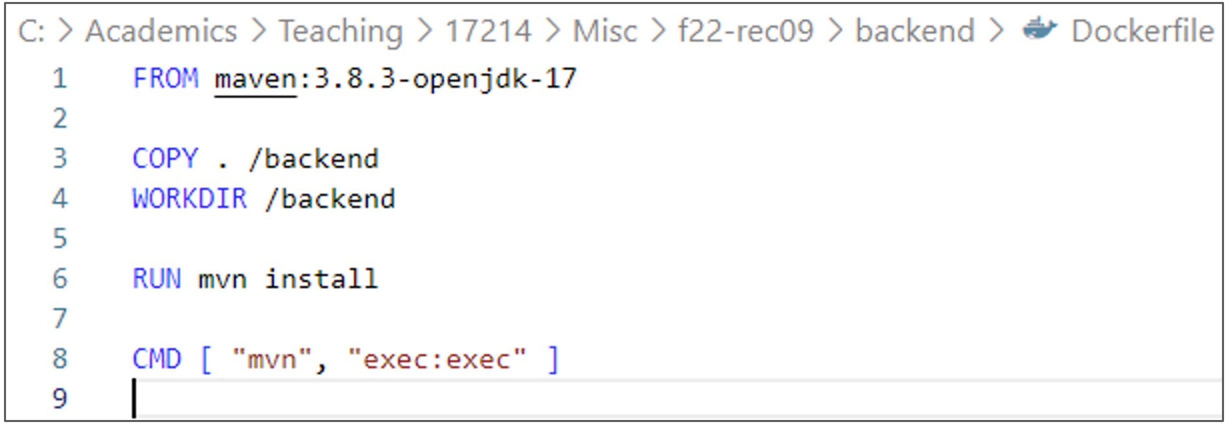
C: > Academics > Teaching > 17214 > Misc > f22-rec09 > frontend > Dockerfile >

```
FROM node

COPY . /frontend
WORKDIR  /frontend

RUN npm install
CMD [ "npm", "start" ]

```

Same for the Backend

Note how the FROM image can have detailed *tags*

- These come from Dockerhub.



C: > Academics > Teaching > 17214 > Misc > f22-rec09 > backend > Dockerfile

```
FROM maven:3.8.3-openjdk-17

COPY . /backend
WORKDIR /backend

RUN mvn install

CMD [ "mvn", "exec:exec" ]

```

Docker Demo

- Build and start front-end
- Build and start back-end
- See layers
- Try to connect to front-end from browser

What Now?

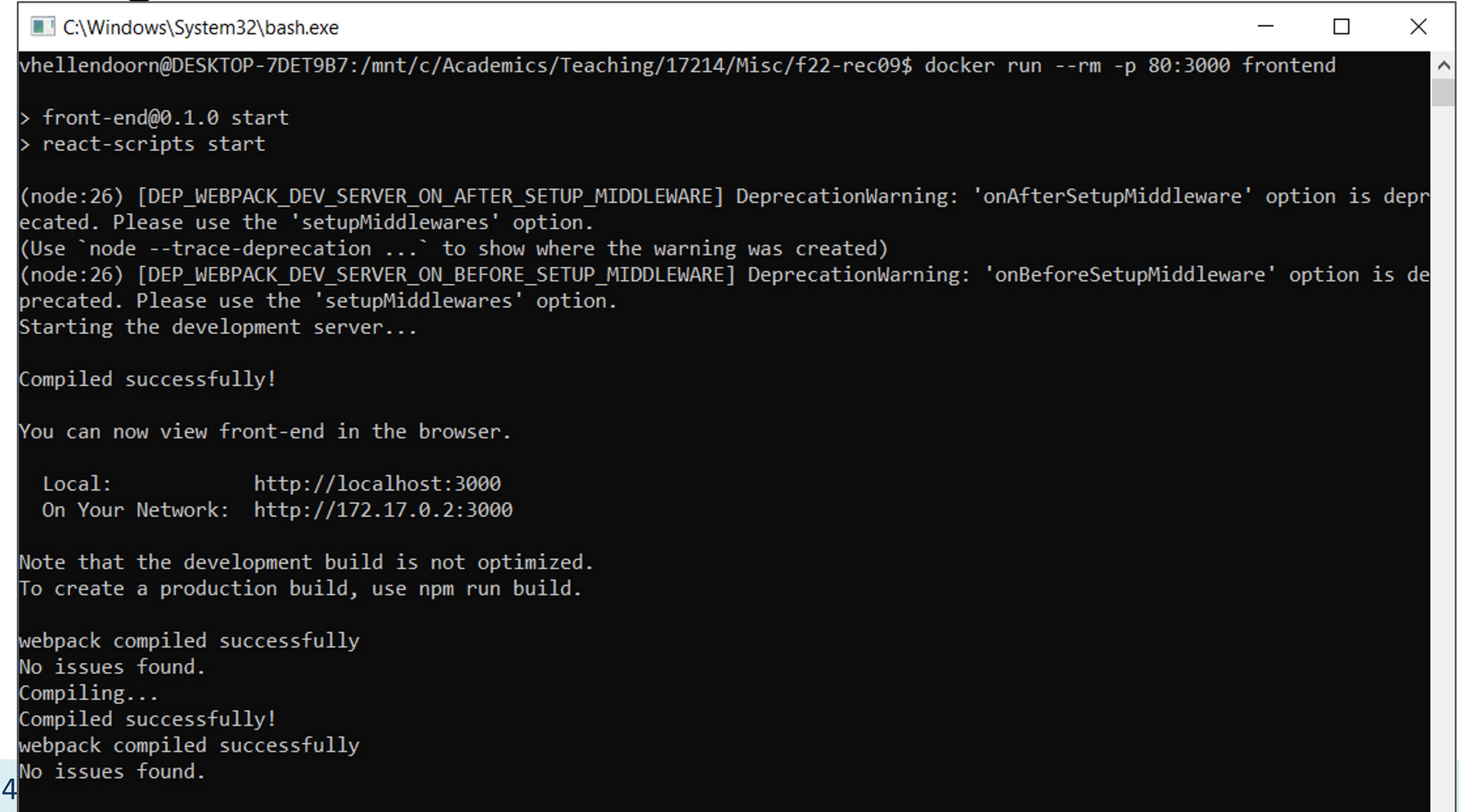
We’ve packaged frontend and backend as separate images

- These are in the same repo -- why separate containers?

How do we talk to them?

- Not quite obvious: containers isolate *everything*

Running Docker Containers



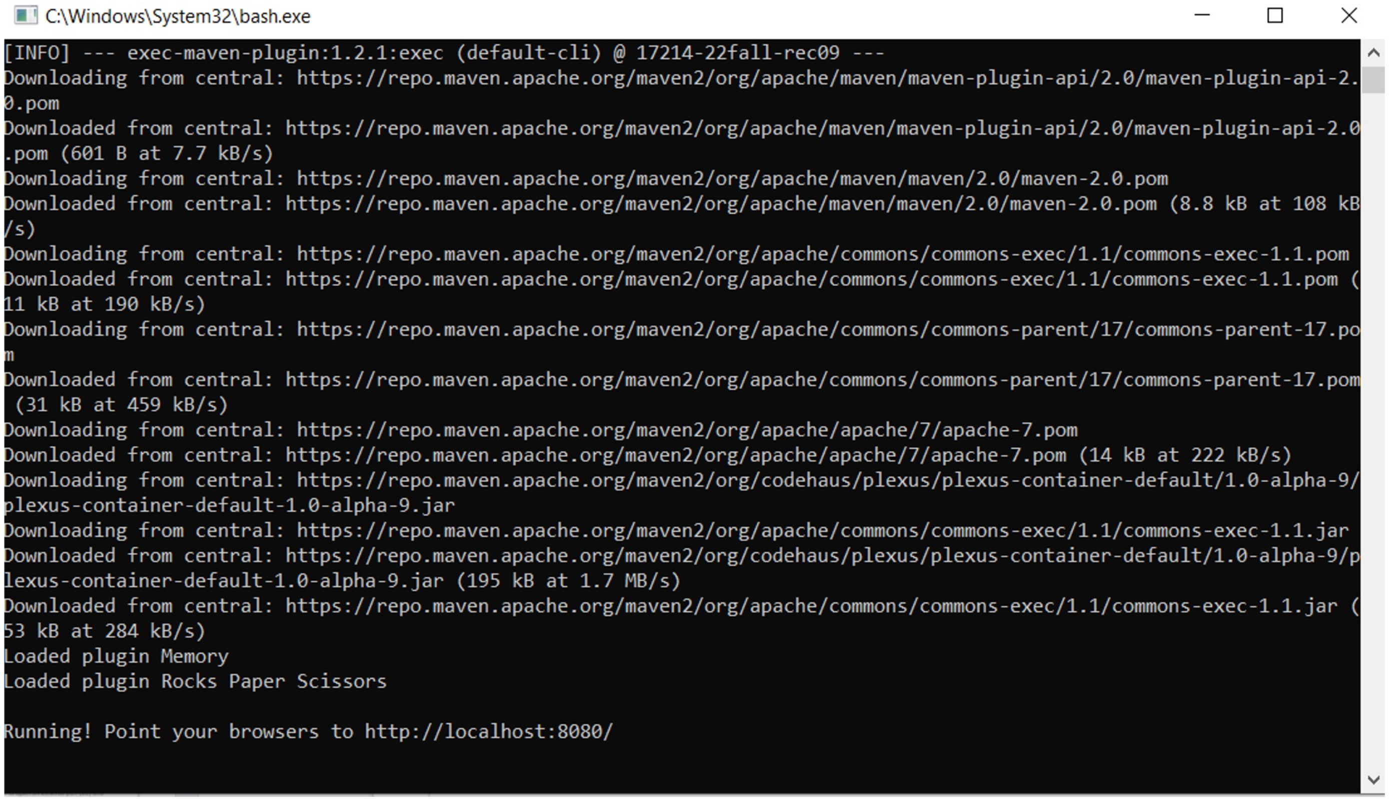
```
vhellendoorn@DESKTOP-7DET9B7:/mnt/c/Academics/Teaching/17214/Misc/f22-rec09$ docker run --rm -p 80:3000 frontend

> front-end@0.1.0 start
> react-scripts start

(node:26) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is depr
ecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:26) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is de
precated. Please use the 'setupMiddlewares' option.
Starting the development server...

Compiled successfully!

You can now view front-end in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.17.0.2:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
No issues found.
Compiling...
Compiled successfully!
webpack compiled successfully
No issues found.
```

Running Docker Containers

Run: `docker run --rm -p 80:3000 frontend`

- `--rm`: removes the container after shutdown
  - Important! Docker keeps machines around indefinitely otherwise
  - Containers can hold quite a bit of data
- `-p 80:3000`: instruct Docker to open an external port (80) and forward requests there to the internal one (3000)

Start the Backend too, go to localhost:80, and…

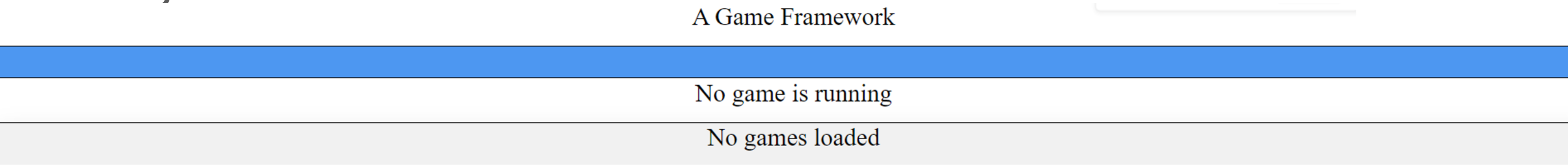


```
[INFO] --- exec-maven-plugin:1.2.1:exec (default-cli) @ 17214-22fall-rec09 ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-api/2.0/maven-plugin-api-2.
0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-api/2.0/maven-plugin-api-2.0
.pom (601 B at 7.7 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven/2.0/maven-2.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven/2.0/maven-2.0.pom (8.8 kB at 108 kB
/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.pom (
11 kB at 190 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-parent/17/commons-parent-17.po
m
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-parent/17/commons-parent-17.pom
 (31 kB at 459 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/7/apache-7.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/7/apache-7.pom (14 kB at 222 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-9/
plexus-container-default-1.0-alpha-9.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-9/p
lexus-container-default-1.0-alpha-9.jar (195 kB at 1.7 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.jar (
53 kB at 284 kB/s)
Loaded plugin Memory
Loaded plugin Rocks Paper Scissors

Running! Point your browsers to http://localhost:8080/
```

It doesn’t work!?

The frontend loads, but can’t talk to the backend

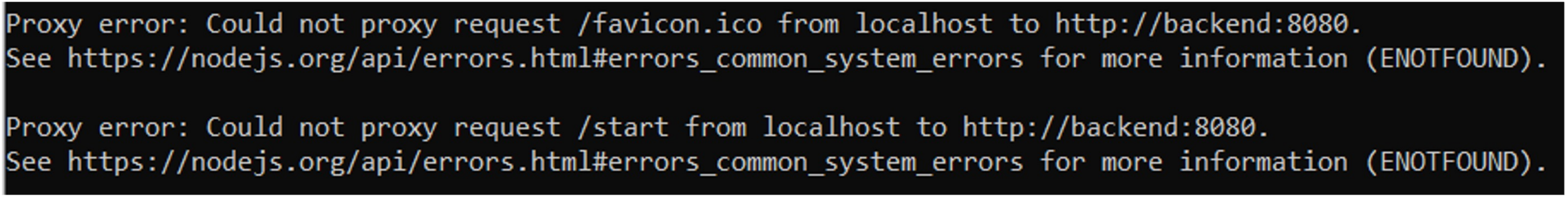
Why not?



A Game Framework

No game is running

No games loaded



```
Proxy error: Could not proxy request /favicon.ico from localhost to http://backend:8080.
See https://nodejs.org/api/errors.html#errors_common_system_errors for more information (ENOTFOUND).

Proxy error: Could not proxy request /start from localhost to http://backend:8080.
See https://nodejs.org/api/errors.html#errors_common_system_errors for more information (ENOTFOUND).
```

Remember: containers means isolation

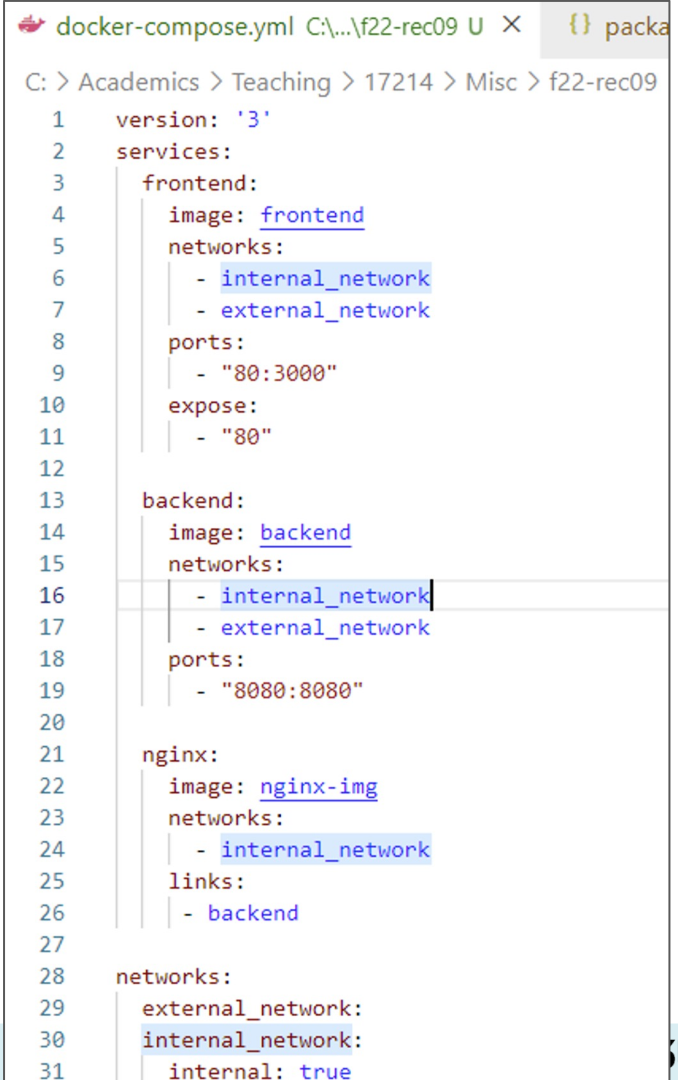
Networks are also virtual

- Each container subscribes to ‘bridge’ by default
- Containers are assigned unique IPs within each network
- We *could* make this work by (a) starting backend, (b) finding its IP on ‘bridge’, (c) rebuilding frontend with that IP hard-coded in package.json, and (d) launching frontend
- Not great; imagine running a website that way

Docker Compose

We (and Bogdan’s suntan!) need container management tools

- Lowest level: docker compose
  - Specify images, networks & ports, links, etc.
  - Can launch many copies of each image

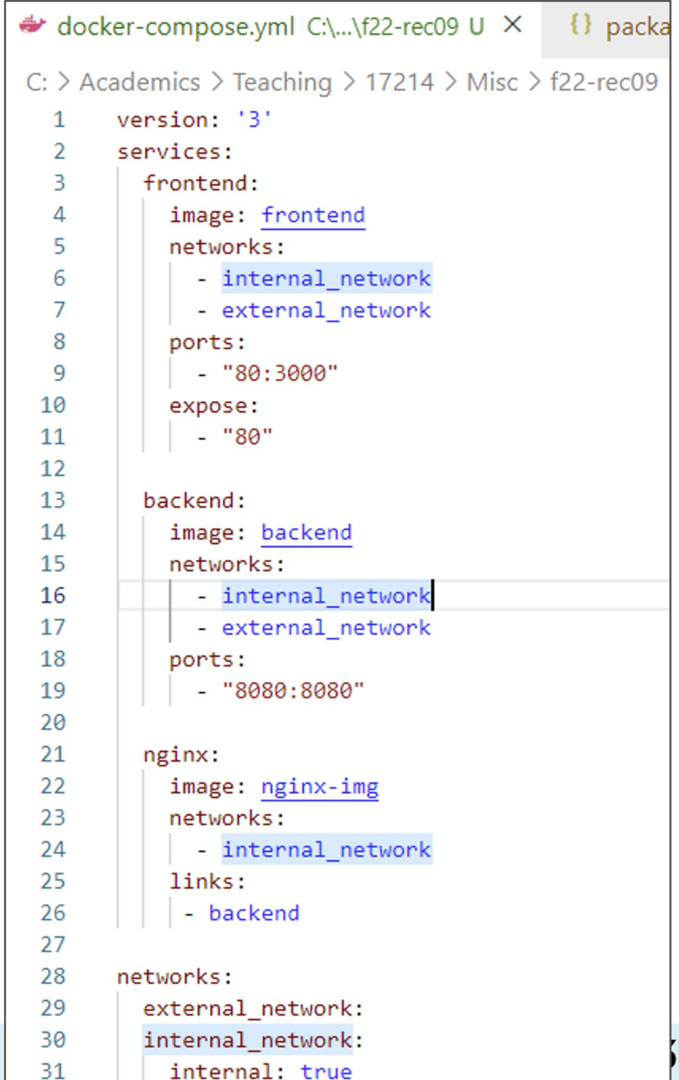

docker-compose.yml C:\...\f22-rec09 U ×
{} packa
C: > Academics > Teaching > 17214 > Misc > f22-rec09

```yaml
version: '3'
services:
  frontend:
    image: frontend
    networks:
      - internal_network
      - external_network
    ports:
      - "80:3000"
    expose:
      - "80"

  backend:
    image: backend
    networks:
      - internal_network
      - external_network
    ports:
      - "8080:8080"

  nginx:
    image: nginx-img
    networks:
      - internal_network
    links:
      - backend

networks:
  external_network:
  internal_network:
    internal: true
```

Docker Compose Demo

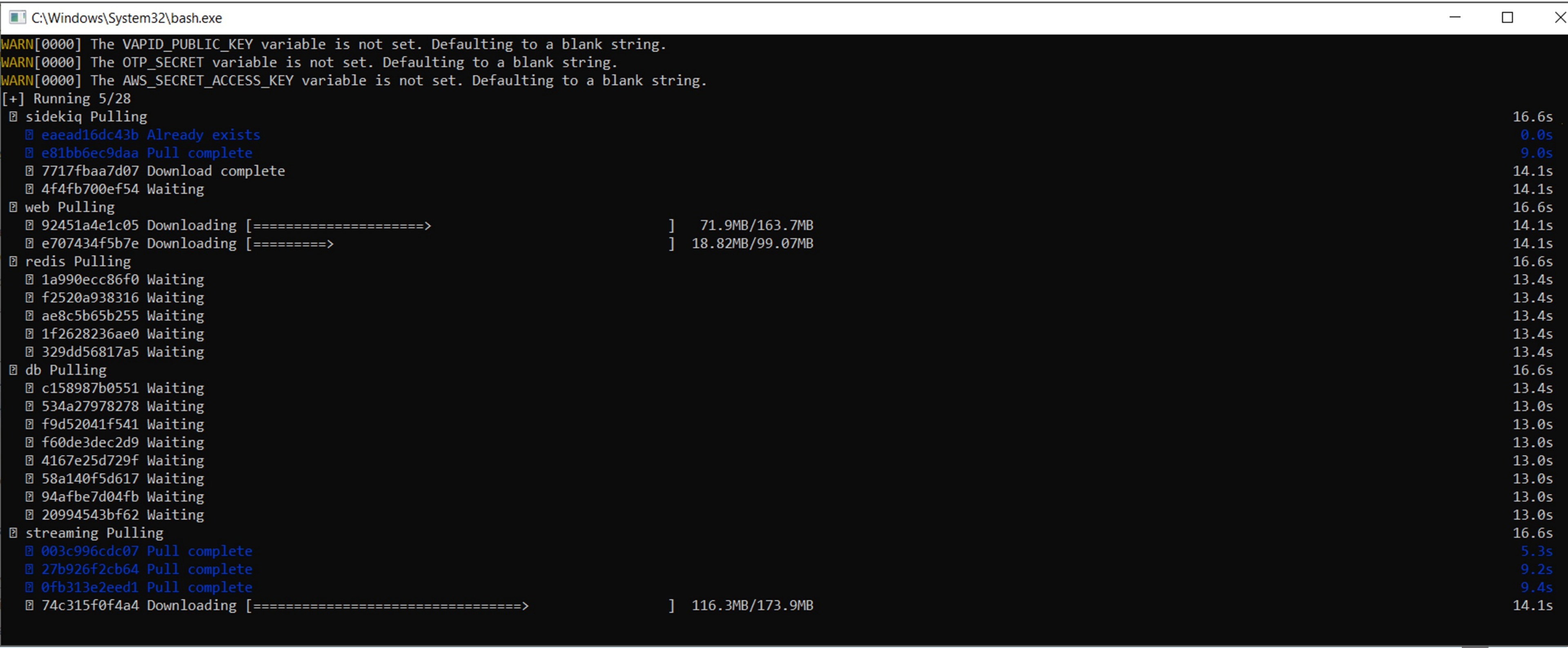
- Let Docker Compose:
  - Start front-end
  - Start back-end
  - Configure all the permissions
- Try to connect to front-end

- We had to change the front-end’s package.json proxy statement from localhost to backend **then** rebuild the front-end container
- Why?


docker-compose.yml C:\...\f22-rec09 U ×
{} packa
C: > Academics > Teaching > 17214 > Misc > f22-rec09

```yaml
version: '3'
services:
  frontend:
    image: frontend
    networks:
      - internal_network
      - external_network
    ports:
      - "80:3000"
    expose:
      - "80"

  backend:
    image: backend
    networks:
      - internal_network
      - external_network
    ports:
      - "8080:8080"

  nginx:
    image: nginx-img
    networks:
      - internal_network
    links:
     - backend

networks:
  external_network:
  internal_network:
    internal: true
```

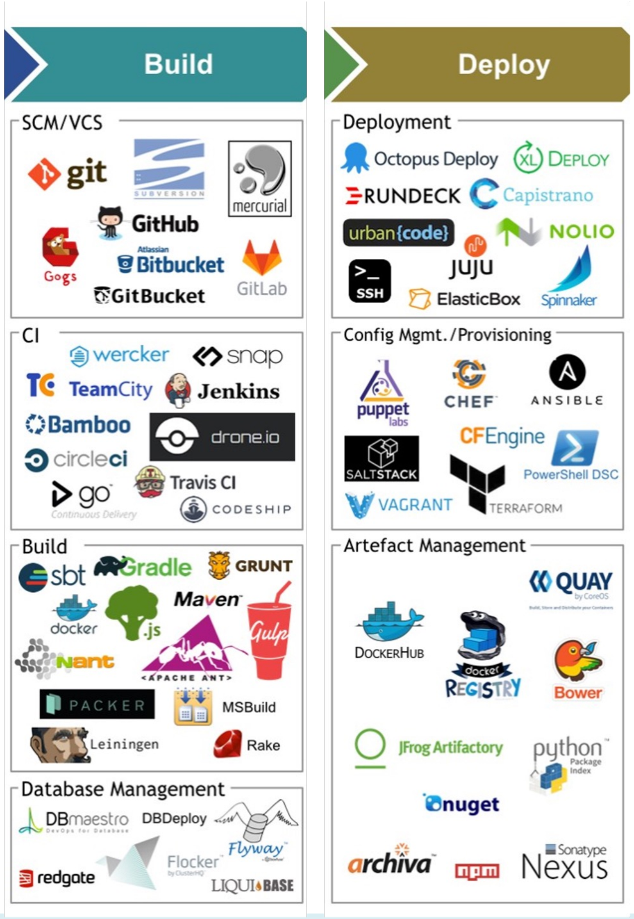
Many apps can be deployed this way (Mastodon)



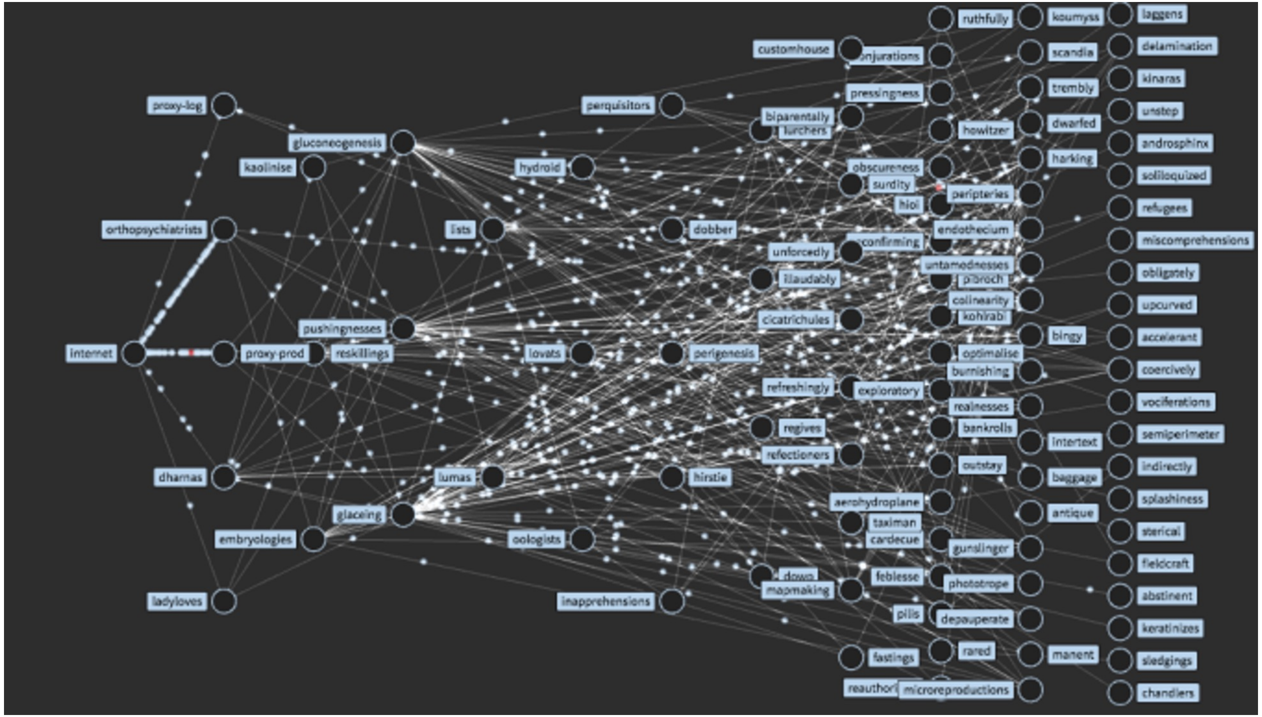
```
WARN[0000] The VAPID_PUBLIC_KEY variable is not set. Defaulting to a blank string.
WARN[0000] The OTP_SECRET variable is not set. Defaulting to a blank string.
WARN[0000] The AWS_SECRET_ACCESS_KEY variable is not set. Defaulting to a blank string.
[+] Running 5/28
 ⍰ sidekiq Pulling                                                                                                                                                                        16.6s
   ⍰ eaead16dc43b Already exists                                                                                                                                                           0.0s
   ⍰ e81bb6ec9daa Pull complete                                                                                                                                                            9.0s
   ⍰ 7717fbaa7d07 Download complete                                                                                                                                                       14.1s
   ⍰ 4f4fb700ef54 Waiting                                                                                                                                                                 14.1s
 ⍰ web Pulling                                                                                                                                                                            16.6s
   ⍰ 92451a4e1c05 Downloading [=====================>                             ]   71.9MB/163.7MB                                                                                      14.1s
   ⍰ e707434f5b7e Downloading [=========>                                         ]  18.82MB/99.07MB                                                                                      14.1s
 ⍰ redis Pulling                                                                                                                                                                          16.6s
   ⍰ 1a990ecc86f0 Waiting                                                                                                                                                                 13.4s
   ⍰ f2520a938316 Waiting                                                                                                                                                                 13.4s
   ⍰ ae8c5b65b255 Waiting                                                                                                                                                                 13.4s
   ⍰ 1f2628236ae0 Waiting                                                                                                                                                                 13.4s
   ⍰ 329dd56817a5 Waiting                                                                                                                                                                 13.4s
 ⍰ db Pulling                                                                                                                                                                             16.6s
   ⍰ c158987b0551 Waiting                                                                                                                                                                 13.4s
   ⍰ 534a27978278 Waiting                                                                                                                                                                 13.0s
   ⍰ f9d52041f541 Waiting                                                                                                                                                                 13.0s
   ⍰ f60de3dec2d9 Waiting                                                                                                                                                                 13.0s
   ⍰ 4167e25d729f Waiting                                                                                                                                                                 13.0s
   ⍰ 58a140f5d617 Waiting                                                                                                                                                                 13.0s
   ⍰ 94afbe7d04fb Waiting                                                                                                                                                                 13.0s
   ⍰ 20994543bf62 Waiting                                                                                                                                                                 13.0s
 ⍰ streaming Pulling                                                                                                                                                                      16.6s
   ⍰ 003c996cdc07 Pull complete                                                                                                                                                            5.3s
   ⍰ 27b926f2cb64 Pull complete                                                                                                                                                            9.2s
   ⍰ 0fb313e2eed1 Pull complete                                                                                                                                                            9.4s
   ⍰ 74c315f0f4a4 Downloading [=================================>                 ]  116.3MB/173.9MB                                                                                      14.1s
```

Where are we now?

- We’ve discussed:
  - Docker as a build tool
  - DockerHub for deployment
  - Docker Compose for orchestration
- Something is off about our app
  - What’s missing?



Remember this?



http://christophermeiklejohn.com/filibuster/2021/10/14/filibuster-4.html

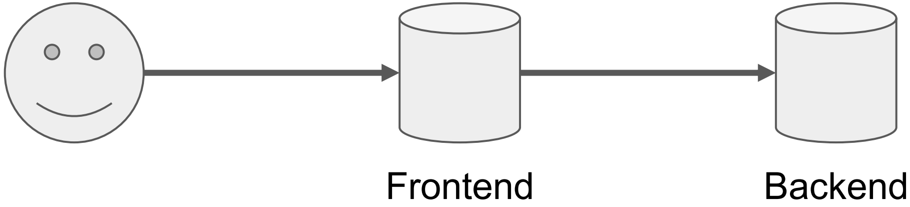
Towards Distributed Systems

- Docker compose helps us set up local systems
  - The result could be microservice or a larger app
  - Often very useful: enables modular development with all the ease of docker images for deployment
- But in our case, backend and frontend are both microservices
  - Why might we not want just one of each, hard-coded to talk to each other?

Towards Distributed Systems

Let’s start with:

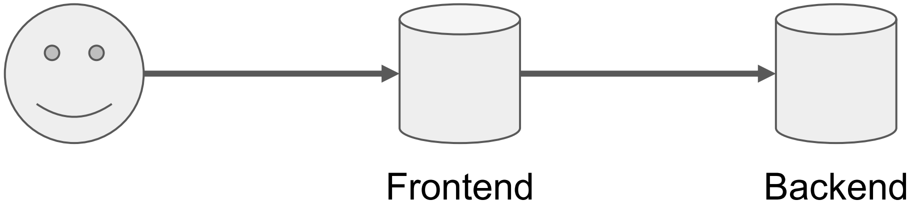
- Put up two VMs in the cloud, deploy one image on each
- Tell ‘frontend’ where to find ‘backend’ by IP



Towards Distributed Systems

How about:

- Put up two VMs in the cloud, deploy one image on each
- Tell ‘frontend’ where to find ‘backend’ by IP
- Problems?



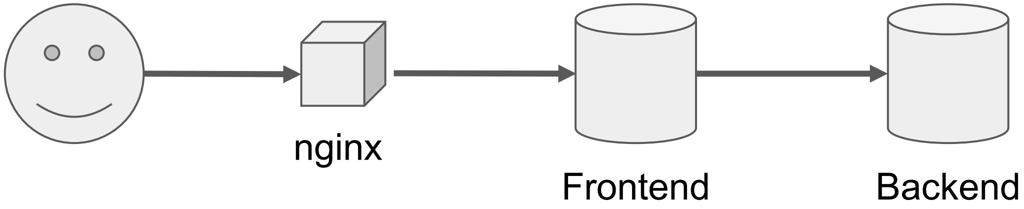
Things to consider in distributed systems

- How will VMs know where other VMs are?
- How will VMs know they can trust incoming messages?
- What parts of your topology may change?
- How will you change the topology without interruptions?
- Where will you need replication?
- How will clients find your application?

nginx

Is a reverse proxy*

- A reverse proxy does for servers what a regular proxy does for users – provide decoupling
  - Good for security, performance, robustness to system changes, ...



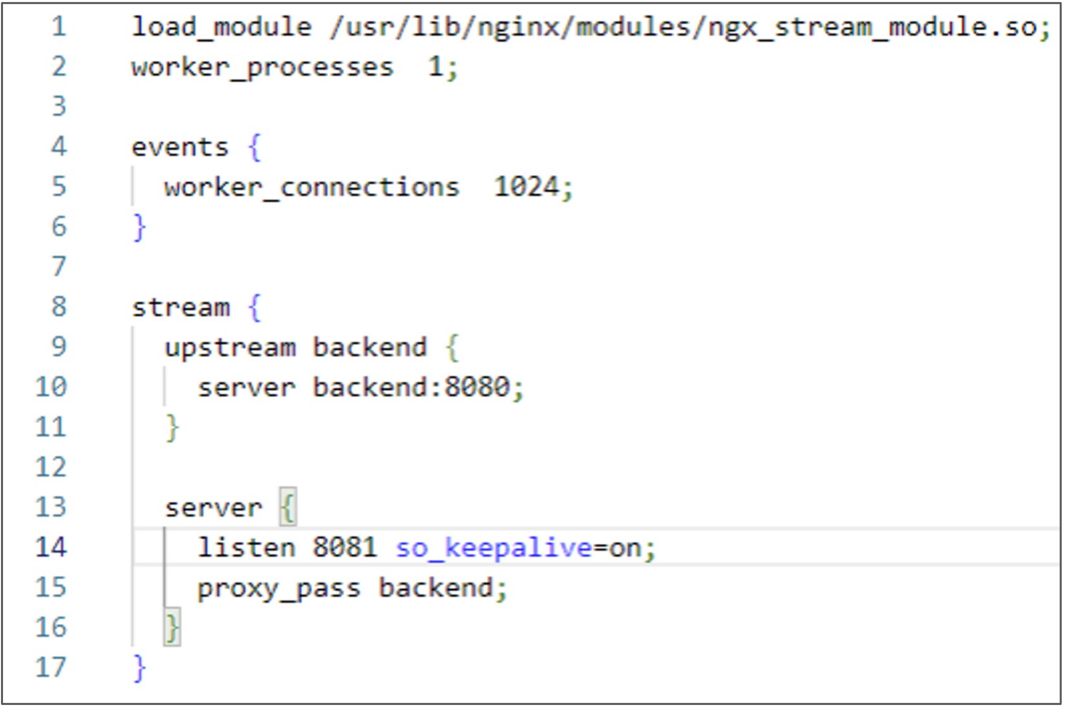
*Technically it’s a web server that is really easy to set up as a reverse proxy server

Nginx Configuration Example

- Handles up to 1024 clients
- ‘upstream’ is the server being proxied for
    - There can be many
- ‘server’ is this proxy server
    - Listens on port, passes messages to upstream

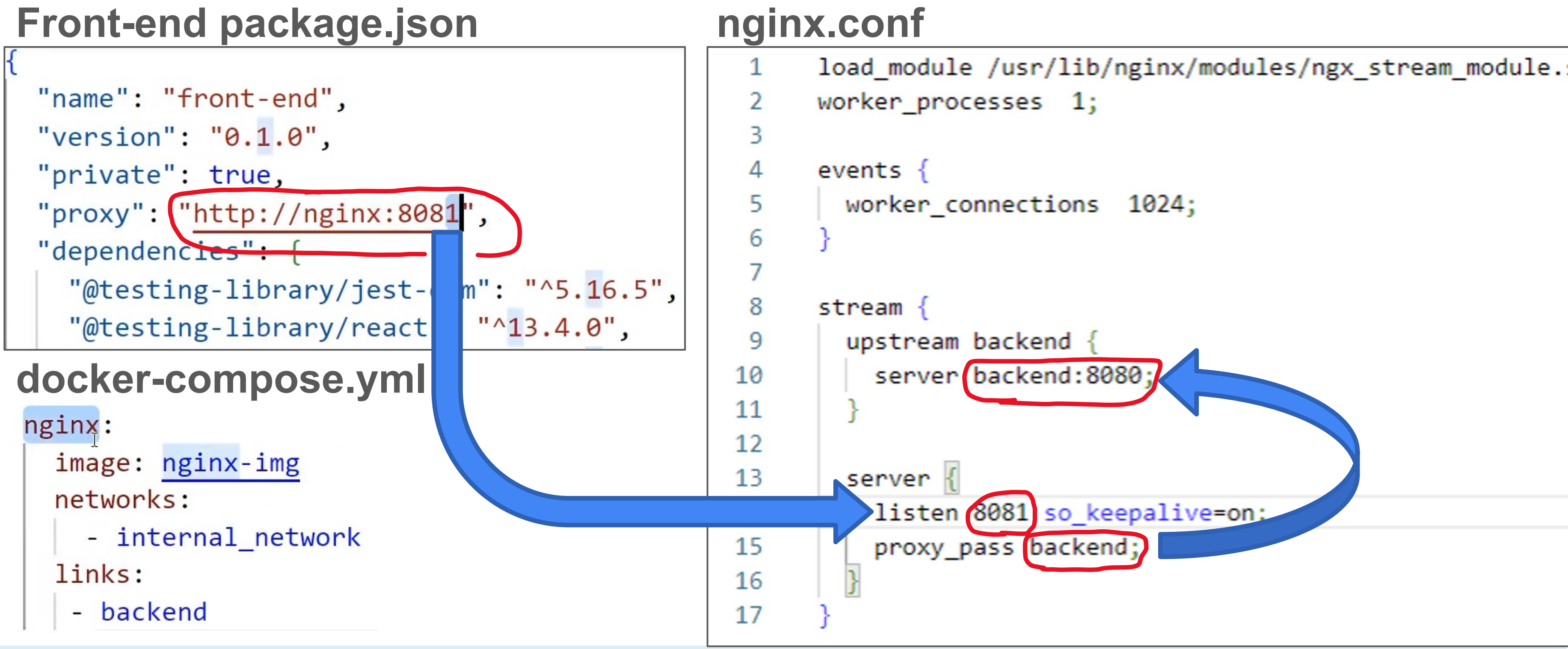
Note: here the proxy is between the frontend and backend

Note: the ‘upstream’ terminology may seem backwards to you … this is because we are using nginx here as a <u>reverse</u> proxy



```
load_module /usr/lib/nginx/modules/ngx_stream_module.so;
worker_processes  1;

events {
  worker_connections  1024;
}

stream {
  upstream backend {
    server backend:8080;
  }

  server {
    listen 8081 so_keepalive=on;
    proxy_pass backend;
  }
}
```

Nginx Configuration Example



Front-end package.json

```json
{
  "name": "front-end",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://nginx:8081",
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
```

docker-compose.yml

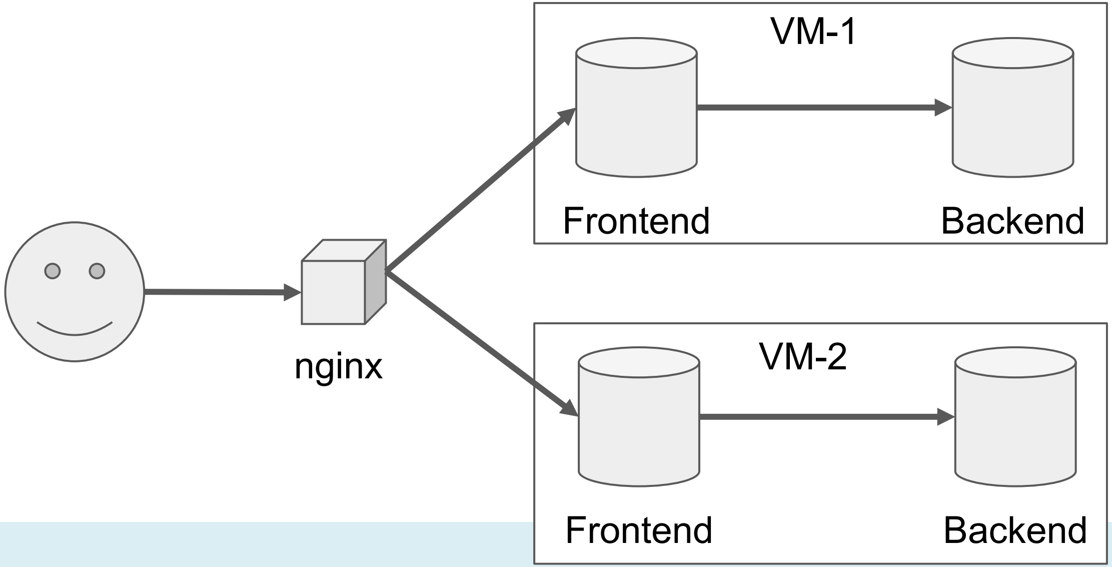
```yaml
nginx:
  image: nginx-img
  networks:
    - internal_network
  links:
    - backend
```

nginx.conf

```
load_module /usr/lib/nginx/modules/ngx_stream_module.
worker_processes  1;

events {
  worker_connections  1024;
}

stream {
  upstream backend {
    server backend:8080;
  }

  server {
    listen 8081 so_keepalive=on;
    proxy_pass backend;
  }
}
```

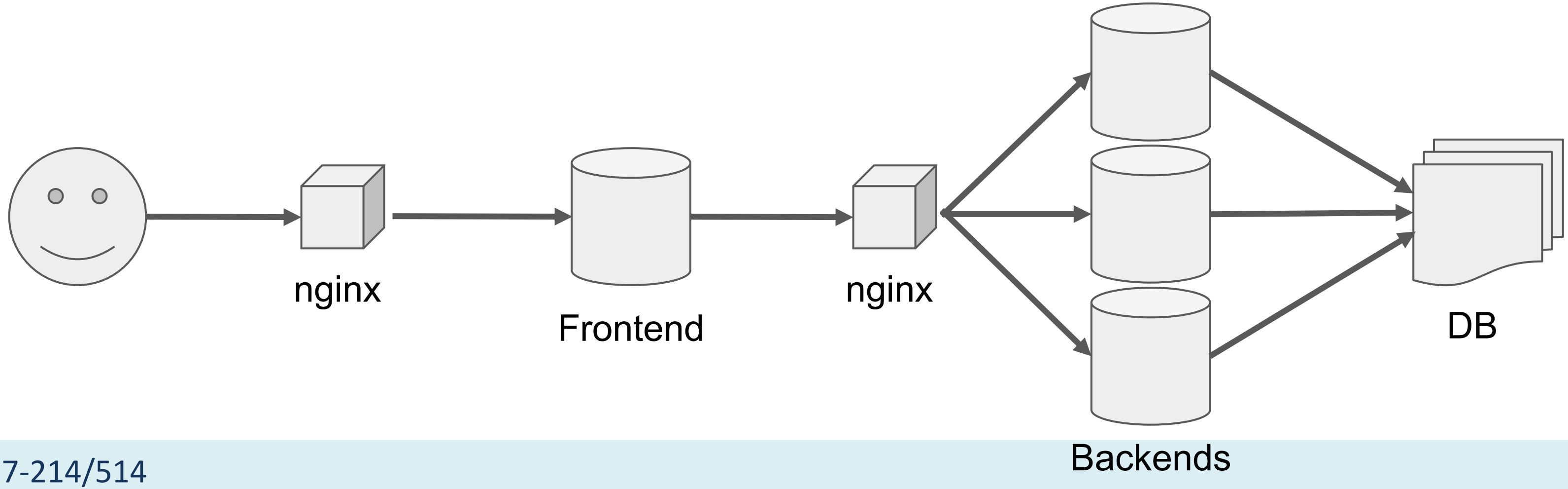
Load Balancing

- Reverse proxies make it easy to divide web traffic
  - ○ Give nginx multiple upstreams
    → nginx will divide traffic using round robin (by default)



Combine Creatively

- Not sufficient, but very helpful for:
  - Performance, through replication
    - Nginx server is often very powerful
  - Robustness, handle failing nodes via indirection



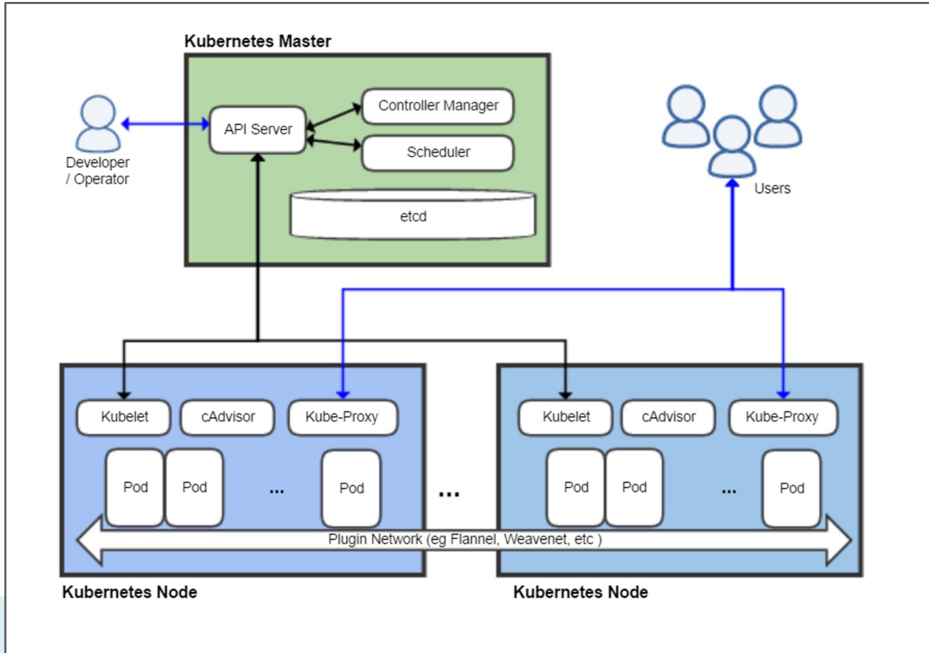
Who tells the proxies what to do?

- Note that Nginx doesn’t solve most of our problems!
    - How will VMs know where other VMs are?
    - How will VMs know they can trust incoming messages?
    - What parts of your topology may change?
    - *How will you change the topology without interruptions?*
    - Where will you need replication?
    - How will clients find your application?

Managing Distributed Topologies is Hard

So don’t do it (yourself)!

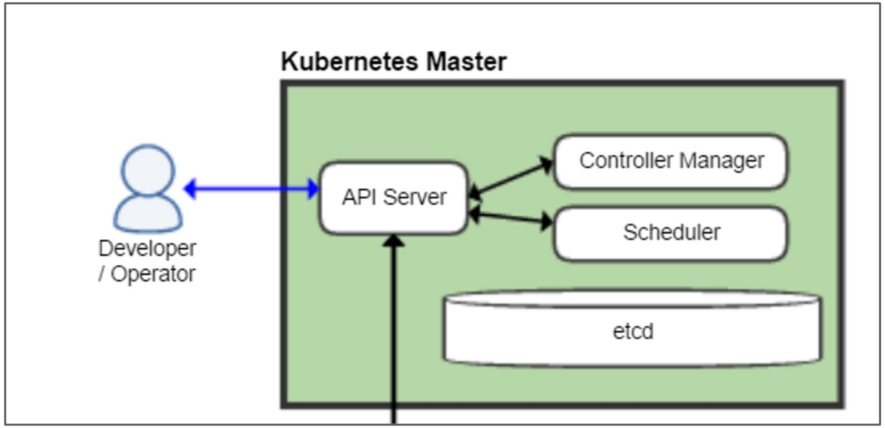
- Kubernetes (k8s), built by Google, manages containers
- Many now-familiar ideas; let’s inspect them



Managing Systems with Kubernetes

The Master:

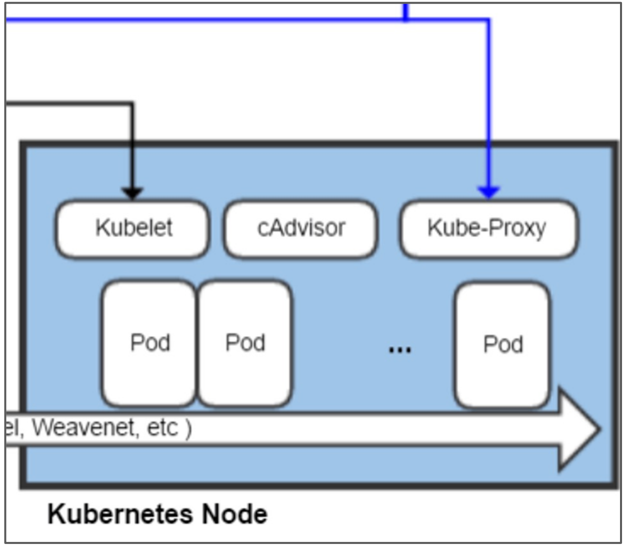
- Tracks global system state in etcd
- Scheduler tracks resource availability, assigns work to hardware
- Controllers plan services to meet demands, goals
- API for monitoring, updating



Managing Systems with Kubernetes

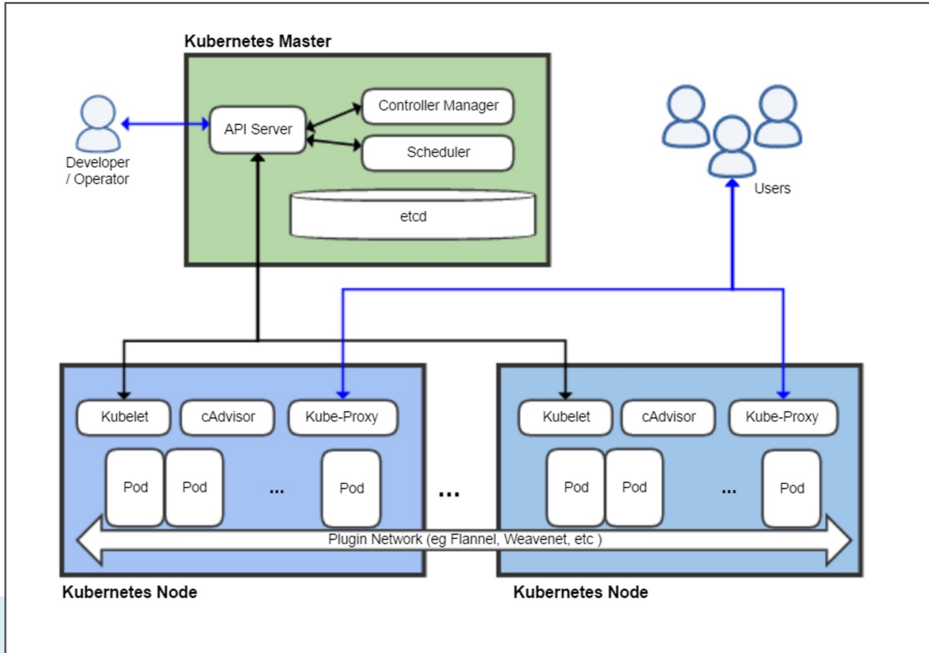
The workers

- Each node is a machine
- Pods consist of connected container(s)
  - Conf., a docker-compose system
  - In fact, containers are usually Docker
- Kubelets monitor the pods, can reprovision
  - Connected to the master
- Kube-proxy provides routing, load balancing
  - Conf., nginx



Managing Systems with Kubernetes

- Note how much this decouples the client from the code
  - In our previous systems, the client talked directly to the frontend
  - Now, to a data center, which talks to a proxy, to a pod, to a container, to code



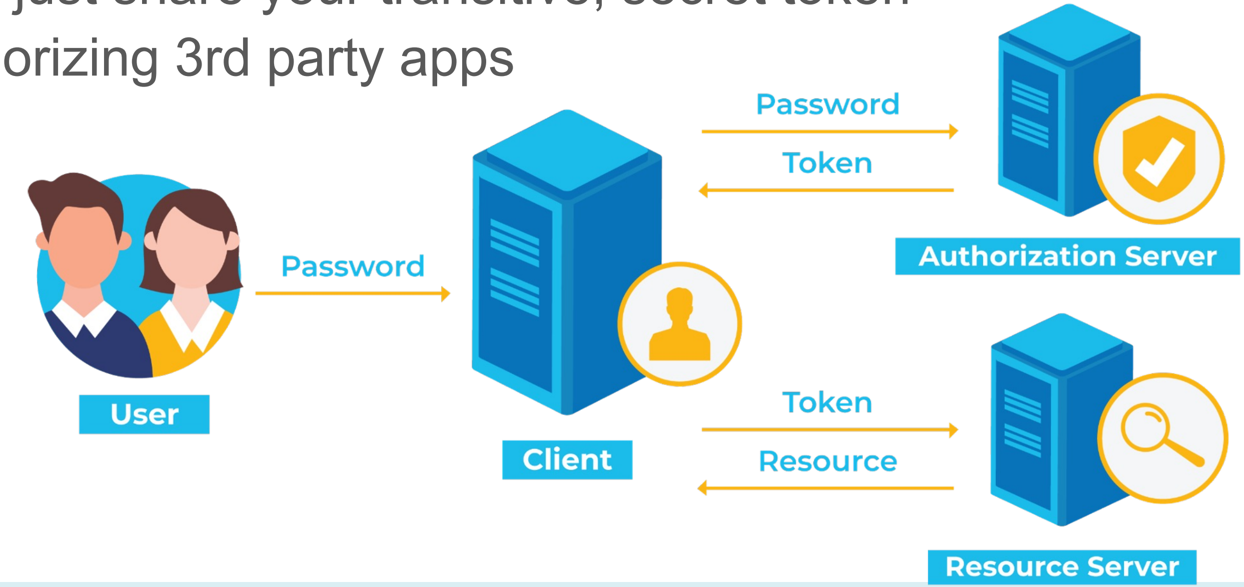
Addresses several questions

- **How will VMs know where other VMs are?**
- *How will VMs know they can trust incoming messages?*
- **What parts of your topology may change?**
- **How will you change the topology without interruptions?**
- Where will you need replication?
- How will clients find your application?

In Brief: Secure Communication

Auth tokens reign supreme these days

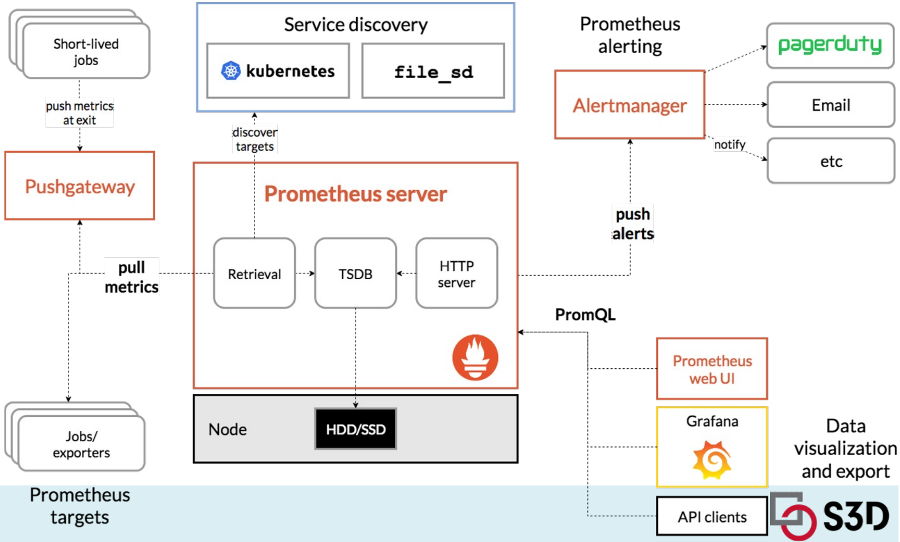
- Single sign on, then just share your transitive, secret token
- Also popular in authorizing 3rd party apps
  – see OAuth(2)



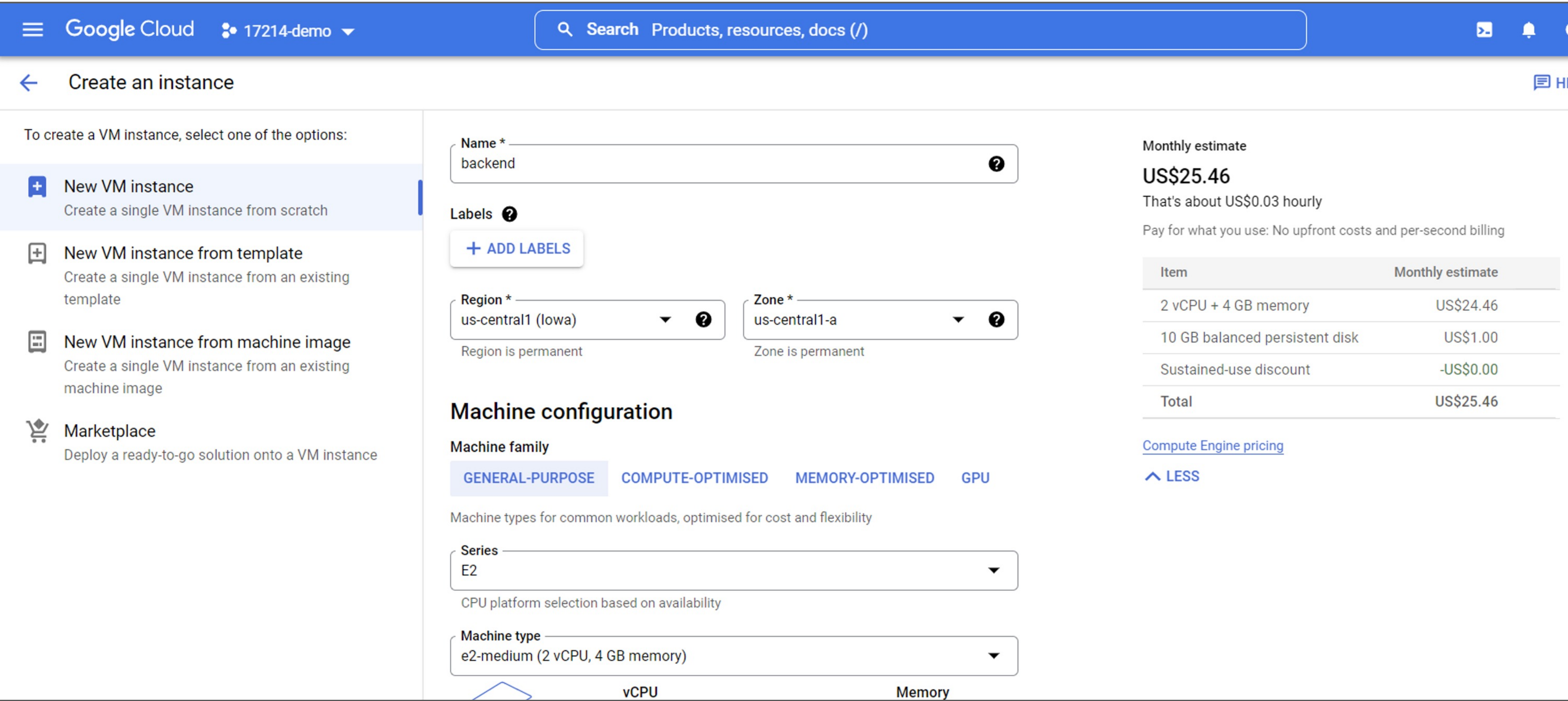
In Brief: Where to Replicate?

Complicated decision, but monitoring helps

- Cloud providers & tools like Kubernetes provide tons of telemetry
- Other tools tap into this to offer insight
- Of course, also financial aspects, legal considerations (geography), forecasting (nothing is ever instant)



This brings us to: Deploying in the Cloud



Create an instance

To create a VM instance, select one of the options:

New VM instance
Create a single VM instance from scratch

New VM instance from template
Create a single VM instance from an existing template

New VM instance from machine image
Create a single VM instance from an existing machine image

Marketplace
Deploy a ready-to-go solution onto a VM instance

Name * backend

Labels

Region * us-central1 (Iowa)
Region is permanent

Zone * us-central1-a
Zone is permanent

Machine configuration

Machine family

Machine types for common workloads, optimised for cost and flexibility

Series E2
CPU platform selection based on availability

Machine type e2-medium (2 vCPU, 4 GB memory)

Monthly estimate

US$25.46

That's about US$0.03 hourly

Pay for what you use: No upfront costs and per-second billing

Item	Monthly estimate
2 vCPU + 4 GB memory	US$24.46
10 GB balanced persistent disk	US$1.00
Sustained-use discount	-US$0.00
Total	US$25.46

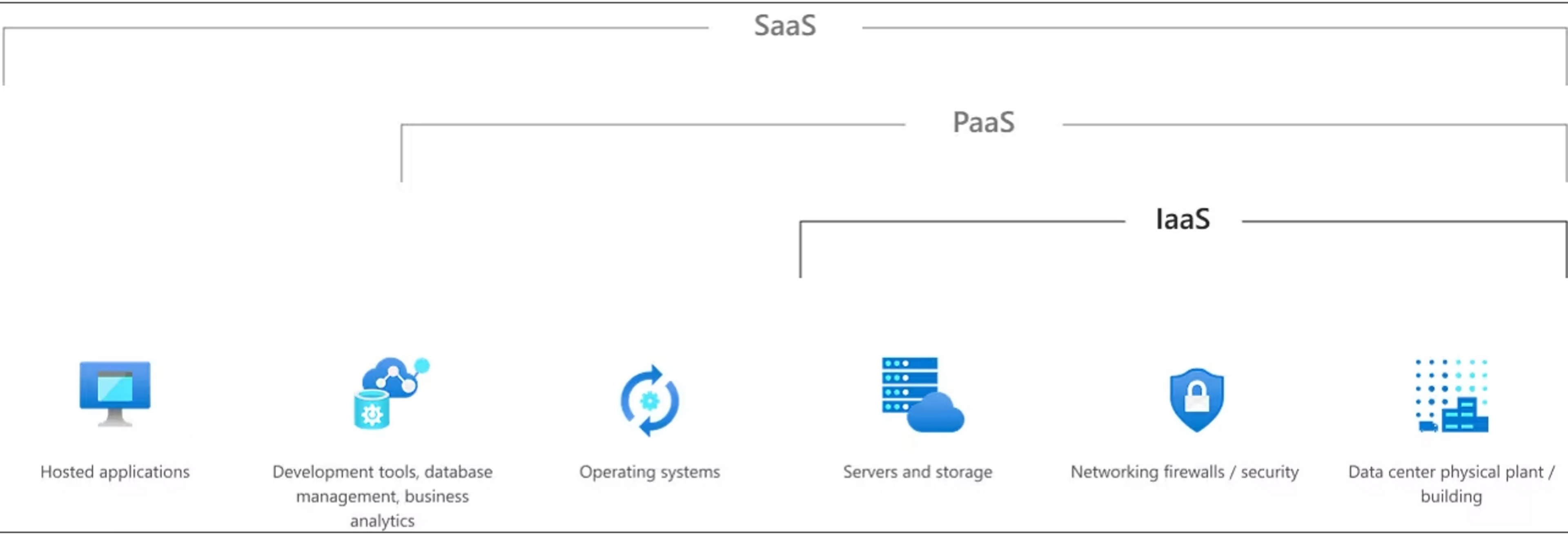
Compute Engine pricing

Deploying in the Cloud

Many types of cloud services are available

- Most natural: Infrastructure as a Service (IaaS)
  - Provision Virtual Machines (VMs) of a given size
    - That’s right, virtualization on top of virtualization
  - Or databases, firewalls, entire clusters – anything that would go in building your own data center

There’s more in the cloud


SaaS
PaaS
IaaS
Hosted applications
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking firewalls / security
Data center physical plant / building

https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/

Note: not everyone thinks of these as nested categories

PaaS: why install your own software?

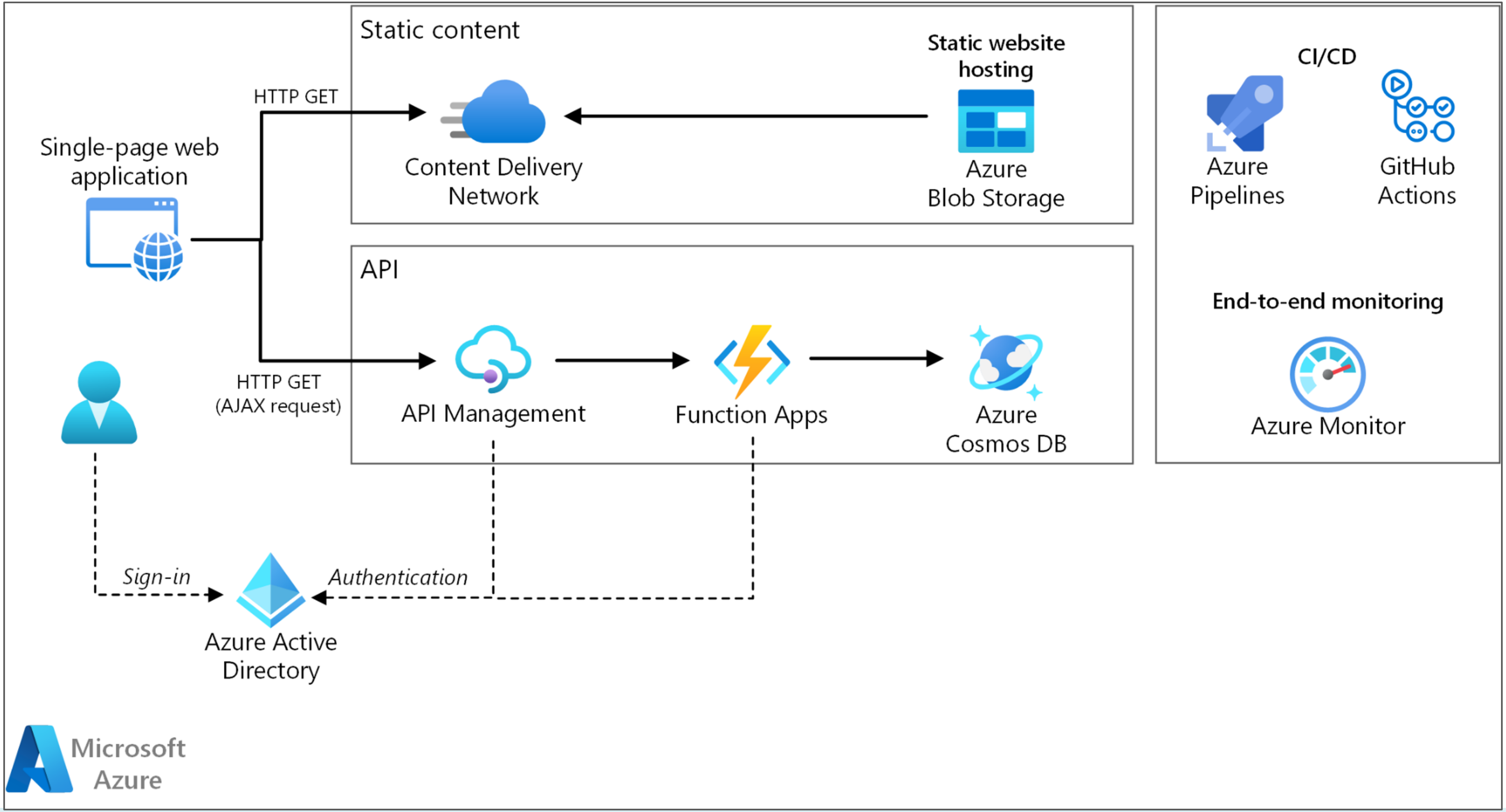
- Don’t just rent machines, rent systems
  - Distributed systems have many common components
    - Like design patterns!
  - Platform as a Service provides preconfigured machines, orchestrators
- Very handy for startups, small teams
  - Managing large distributed systems is <u>hard</u>.

SaaS: why think about machines at all?

- Rent apps, don't think about where they run
  - Common example: email
  - GMail, Google Docs, SalesForce, Colab, etc. are all SaaS
- Very common use-case, major benefits
  - Leaves it to cloud provider to manage infrastructure and deployment. Often a win-win – they benefit from scale.
  - Seriously, don't discount this as an option!
    - Obviously not always applicable, but if you can avoid building your own email client, you should, no matter how easy it seems to develop. A huge chunk of the cost is "hidden" in ops.

Recently Popular: Serverless Computing

- Doesn’t mean “no servers,” just “developers won’t see the servers”
  - Recall PaaS: time not spent managing ops is a big win
- Several instantiations:
  - Functions (e.g., AWS Lambda) – event-driven services that are scaled by the cloud provider (sometimes called **FaaS**)
  - Workflow orchestrators – low/no-code system design
  - Databases – data stores that resize seamlessly (part of **BaaS**)



https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/serverless/web-app

Cloud Computing: Getting to the Point

- We talk a lot about how good design benefits from reuse
  - Of familiar patterns,
  - …of libraries,
  - …of your own code

- This isn’t a distributed systems course
- Take advantage of existing components unless you’re really sure what you are doing

Finally, is the Cloud right for you?

- You're borrowing someone else's computer
  - That comes at a big premium
    - Hosting on-prem can be many times cheaper
    - I recall a thread where a Twitter engineer said their AWS bill would be $100M+/month if they went that way
  - Also fewer guarantees
    - Some VMs are rarely available
    - Allocating large nrs of any kind almost certainly requires discussion
- Still worth it if you:
  - Are a small team, can't spare cycles for system ops
  - Are growing quickly, won't know your computing needs far out

Summary

- Containers provide isolation
  - Lighter than VMs, built with layers
  - Managed hierarchically, via configuration-as-code
- Proxies provide decoupling
  - Good for performance, robustness, security
  - Kubernetes takes this to massive scale
- Think carefully about how you put your app in the cloud
  - Consider tradeoffs between IaaS, PaaS, SaaS, …
  - Also consider cost; cloud bills pile up fast