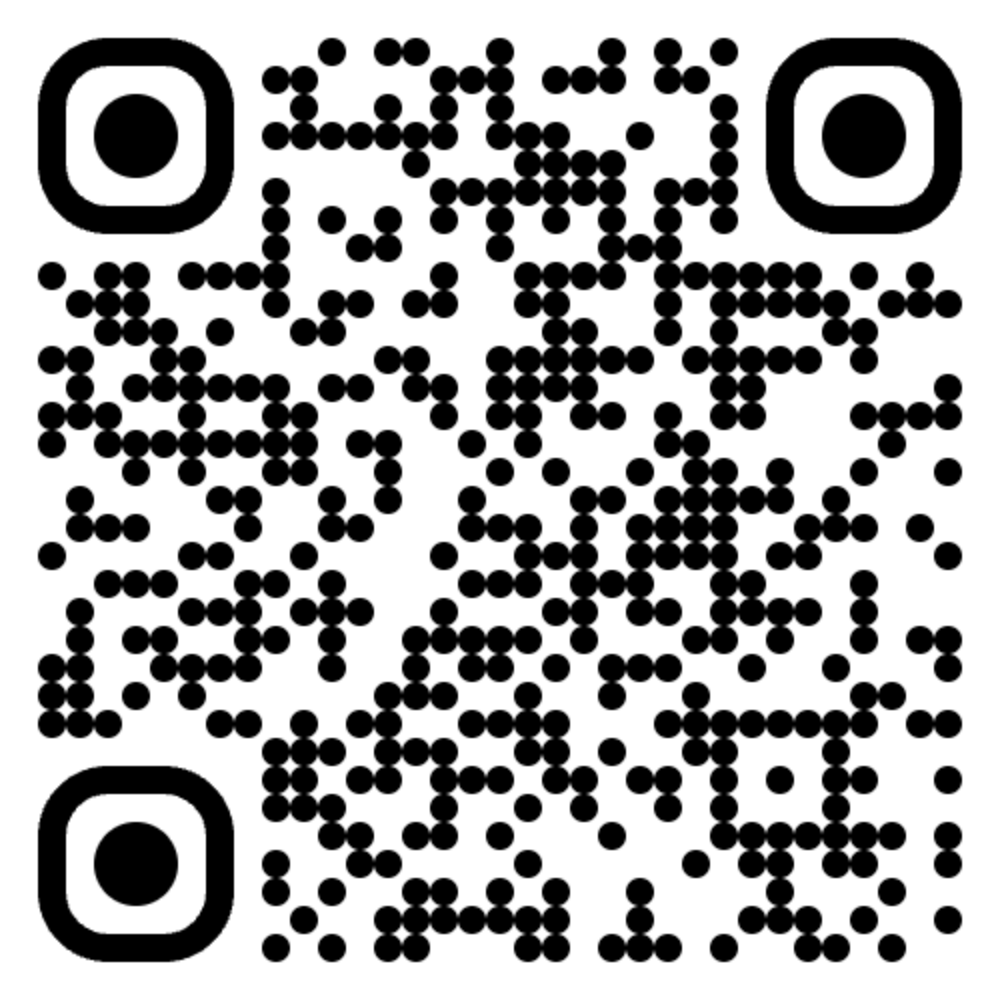
# Containers & Docker

# Admin Stuff

- HW1 due 2/4
- Make sure to join the Slack if you haven't yet (link on Canvas)
  - #general is where all announcements will be made, e.g. HW releases, make sure your notifications are on
  - #help will be for Ed style questions
- You will need Docker installed for today's lab



https://forms.gle/nRcDvXGMNdeEiVrv7

# Agenda

- Why do we need Docker?
- What is Docker?
- How do we use Docker?
- Configurations & Secrets

# Scenario: Deploying Code

- Let's say you have a server that works locally, e.g.
  `uv run fastapi dev main.py`
- How do you actually get this running on another machine?
  - What do we need to do as setup for a new copy of your computer?
  - What if the machine is completely different?
- What we need is *portability*, the ability to run code in different environments easily.
  - Package managers ensure we have the right versions of things, but what about Python itself?
  - How can we ensure that system level things are set/installed, e.g. environment variables or `curl`?

*Code works fine on local machine.

The code in the the dev server


ProgrammerHumor.io

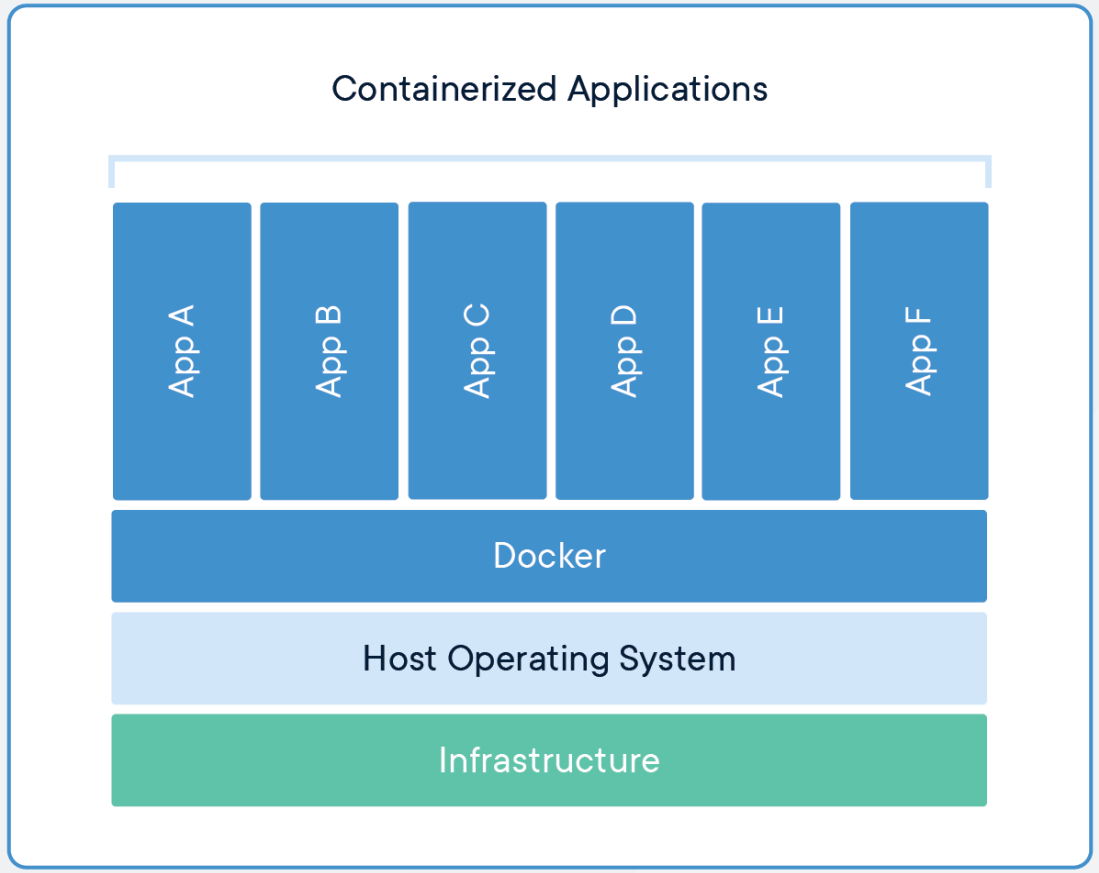
# What if I just used an exact copy of my machine to deploy to production?

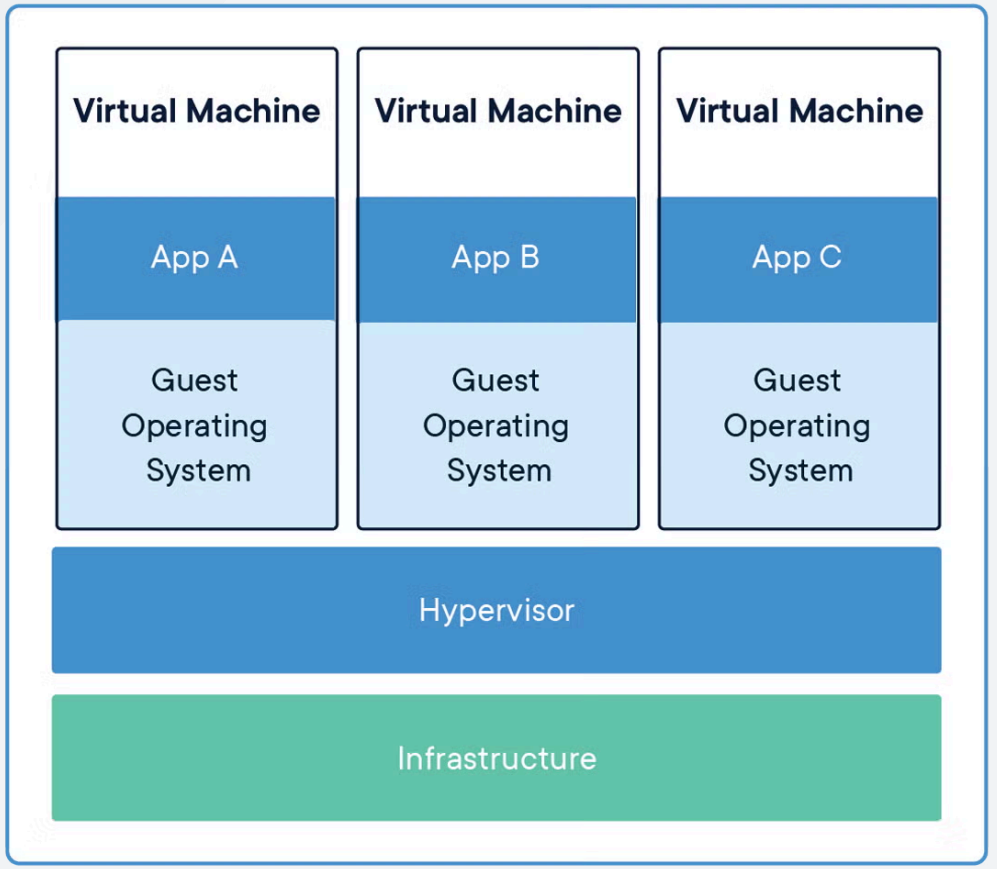
(why does this suck?)

# Containerization

- Intuition: what if we could use something like a virtual machine, but at the application level?
- We define environment setup *as code*, making an **image**
- Anyone can run an image to spawn a **container**
  - Think of images as a blueprint and containers as the object created from that blueprint
  - This allows us to create self-contained and portable execution environments

# Containers vs VMs



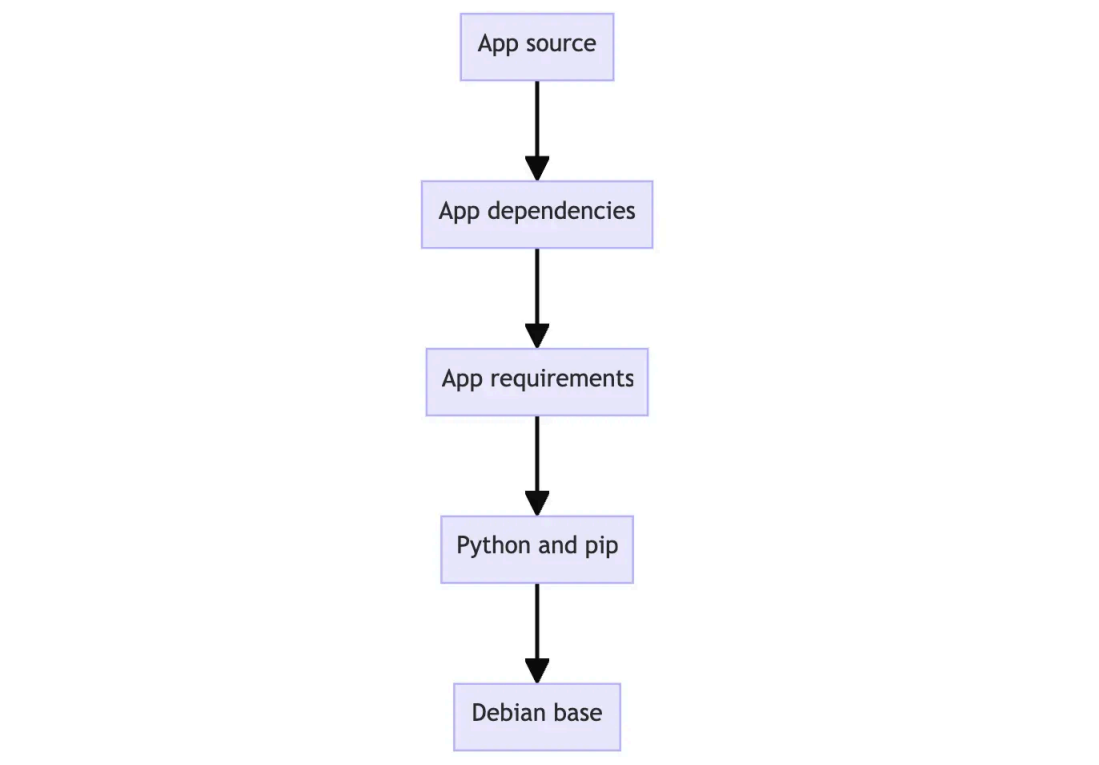

https://www.docker.com/resources/what-container/

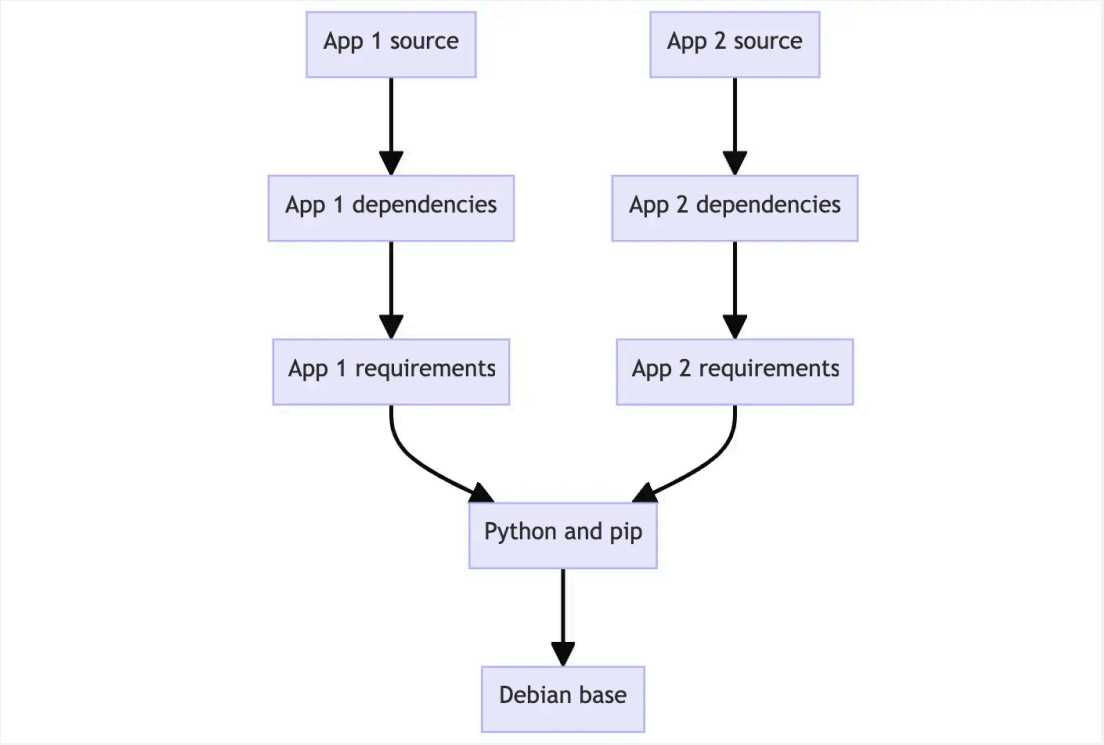

- "Build once, run anywhere"
- Industry-standard tool to package applications and their dependencies into lightweight, portable containers
- Fast and efficient - implemented on top of host OS kernel as a software process, unlike VMs
- Rich ecosystem - Docker Hub registry with millions of pre-built images

# How does Docker work?

- We define *images* through `Dockerfiles`, which express requirements, configs, binaries, etc as code
- Images are created by *building* a Dockerfile
- Images are composed of immutable *layers*, which are used for caching/reuse between images
- We can then spawn a *container* by running an image

# Docker Image Layers




https://docs.docker.com/get-started/docker-concepts/building-images/understanding-image-layers

```dockerfile
# FROM is used to pick a base image to start from
# Alpine is a lightweight linux distro
FROM alpine:3.22

# Install uv
# COPY can be used to copy files from an existing image on the internet (from a "hub")
COPY --from=ghcr.io/astral-sh/uv:latest /uv /uvx /bin/

# Set work directory, this effectively sets the root path for any subsequent commands
WORKDIR /app

# COPY can also copy files from our local machine to the image
COPY services/auth/. ./

# RUN is used to run a command directly on the image
RUN uv sync --no-cache --link-mode=copy

# CMD is used to set a command when running a container spawned from this image
# This is not required and can be set at runtime
CMD ["uv", "run", "fastapi", "dev", "main.py", "-p", "8000"]
```

# Container Networking

- Say our Dockerfile is running, but how do we access it?
- By default, containers run in their own **isolated network namespace**
  - Think of it like each container living in its own private apartment building
- Two concepts you need to understand:
  - **Port exposure**: creating a doorway from host to container
  - **Host binding**: making sure your app listens at that doorway

# Port Exposure

- Without explicit port mapping, requests to `localhost:8000` can't reach your container
- Use `-p` to map host ports to container ports:

```
# Map host port 8000 → container port 8000
docker run -p 8000:8000 my-image

# Map to a different host port
docker run -p 3000:8000 my-image
```

- Syntax: `-p host_port:container_port`

# Host Binding

- Port exposure gets traffic *to* your container, but your app also needs to *listen* for it
- `127.0.0.1` inside the container refers to the **container's** loopback, not the host's
- Solution: bind to `0.0.0.0` to accept connections from any network interface

```
# This won't work for requests from outside the container
fastapi dev main.py --host 127.0.0.1

# This will work
fastapi dev main.py --host 0.0.0.0
```

# Docker for Development

- Docker is also a great way to create and manage development environments
- Pain point: onboarding to a new codebase sucks, we have to install tools, fight with errors, add things to our shell, etc.
- We can eliminate all of these things by developing *within* a container!
- We won't explicitly do this in class, but it's good to be aware of
- Cons: more resource intensive, persistent storage is harder, not great for GUI related development

# Configurations & Secrets

# Configuration

- Imagine we have Jarvis running in 3 different environments: local, dev, and production
- How might these environments differ?
- How would this affect our code?
- How do we remedy this?

```python
# services/auth/app/config.py

class Settings:
    database_url: str = "sqlite:///../../jarvis.db"
    jwt_secret_key: str = "super-secret-key-dont-use-this-in-production"
    jwt_algorithm: str = "HS256"
    access_token_expire_minutes: int = 30
    refresh_token_expire_days: int = 30
    cookie_secure: bool = True
    cookie_samesite: str = "None"

settings = Settings()
```

Why does this suck?

# Configuration

- We need a way to deal with two problems:
  - Configurations need to differ across environments
  - Some configuration values need to be kept secret
- Idea: read these values from the environment at runtime, or fallback to an environment file
  - For environments like local and development, we probably don't care as much about keeping things secret, so using a file is better to know explicitly where values are coming from
  - For production, we will need a more robust setup (encryption and key management!). We will build upon this groundwork later

# Lab: Configs & Docker