Intro to Backend Development

Lecture 5 · Containerization

Jessica Sylvester
Shungo Najima

Announcements

Announcements

- PA 3 grades released
- PA 4 is due **this Wednesday (10/26)**
- Office Hours are on Ed/syllabus

Course Overview

1. Routes
2. Databases
3. Relational Databases
4. Abstractions
5. Containerization & DevOps
6. Deployment & Services
7. Hack Challenge
8. Images

Course Overview

1. Routes
2. Databases
3. Relational Databases
4. Abstractions
5. Containerization & DevOps
6. Deployment & Services
7. Hack Challenge
8. Images

Review

ORMs

- Uses objects/classes to interact with the database
- Class ⟷ Table
- Translates SQL to Python
- Automatically handles enforcing reference conditions
- Automatically handles manipulating join tables

Deployment Process

Dev vs Prod Environments

Dev

- Short for “development”
- Environment that engineers write code in
- Print statements, other debugging logs, testing

Prod

- Short for “production”
- Environment that your users interact with
- Clean, error-free code

Deployment Process

1. Compress code into a production environment
2. Prepare production environment to be run-ready
3. Spin up server(s)
4. Download and run prepared production environment on server(s)

How do we run apps on other machines?

Why do such a thing?

- In production, we do not want apps to be running off our local machines
- Instead, we'll be running them on other machines (i.e. servers)!

What is in an environment?

- An operating system
- Your code
- Libraries
- Environment variables

What is in an environment?

- An operating system - `MacOS, Windows, Ubuntu`
- Your code - `app.py, db.py, etc.`
- Libraries - everything inside `requirements.txt, Python`
- Environment variables - 😎

Environment Variables

- We do not want sensitive information in source code
  - Username & password combinations
  - API keys
- Define variables in a local `.env` file
- SSH into server and create the same `.env` file

Environment Variables

```
conn = psycopg2.connect(
    host="localhost",
    database="ithaca-transit",
    user="sn685",
    password="supersecurepassword"
)
```

Environment Variables

```
conn = psycopg2.connect(
    host="localhost",
    database="ithaca-transit",
    user="sn685",
    password="supersecurepassword"
)
```

.env

```
export DATABASE="ithaca-transit"
export USERNAME="sn685"
export PASSWORD="supersecurepassword"
```

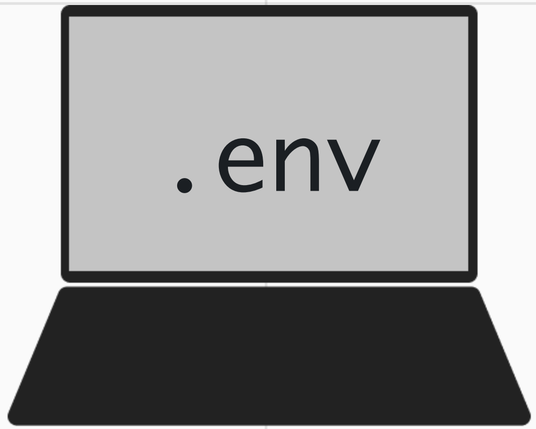
Environment Variables

```python
conn = psycopg2.connect(
    host="localhost",
    database=os.environ["DATABASE"],
    user=os.environ["USERNAME"],
    password=os.environ["PASSWORD"]
)
```

.env

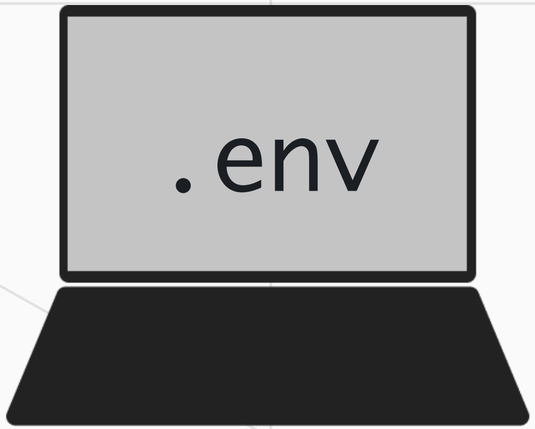
```
export DATABASE="ithaca-transit"
export USERNAME="sn685"
export PASSWORD="supersecurepassword"
```

Environment Variables



Your computer

Internet



Server

Small-scale Approach

- Install `python3`
- Install `requirements.txt`
- Run `app.py`
- Works fine for simple applications

Larger Applications

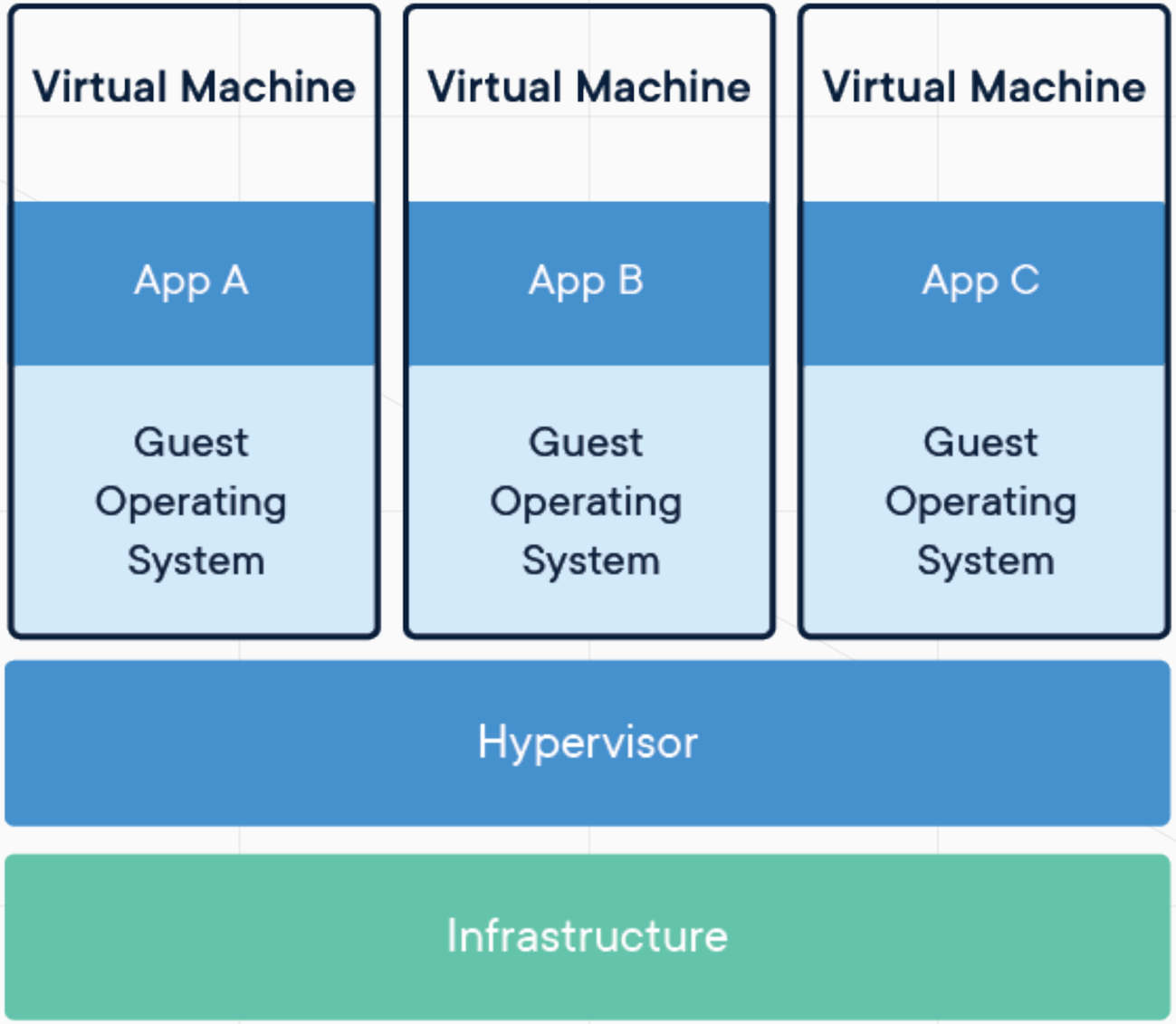
- Include a database
- More custom package installations
- Manually isolate environments
- **TL;DR Setup becomes cumbersome**
- We want a modular and automated solution

Virtualization

(the old way)

Virtual Machines

- Running a computer within a computer
- Allows us to set an operation system to run
- Sandboxed from our host computer’s files
- Can run multiple simultaneously on the same computer
- Servers use a “hypervisor” to manage the VMs



Source: Docker

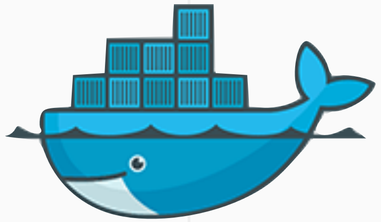
Containerization

Containerization

1. Compress code into a production environment
2. Prepare production environment to be run-ready
3. Spin up server(s)
4. Download and run prepared production environment on server(s)

1. Compress Code Into a Production Environment

Docker



- Containerization = neatly packaging
- Allows developers to package code into a **standardized unit of software**
- Build code into images
- Run an image as a container

Docker Images

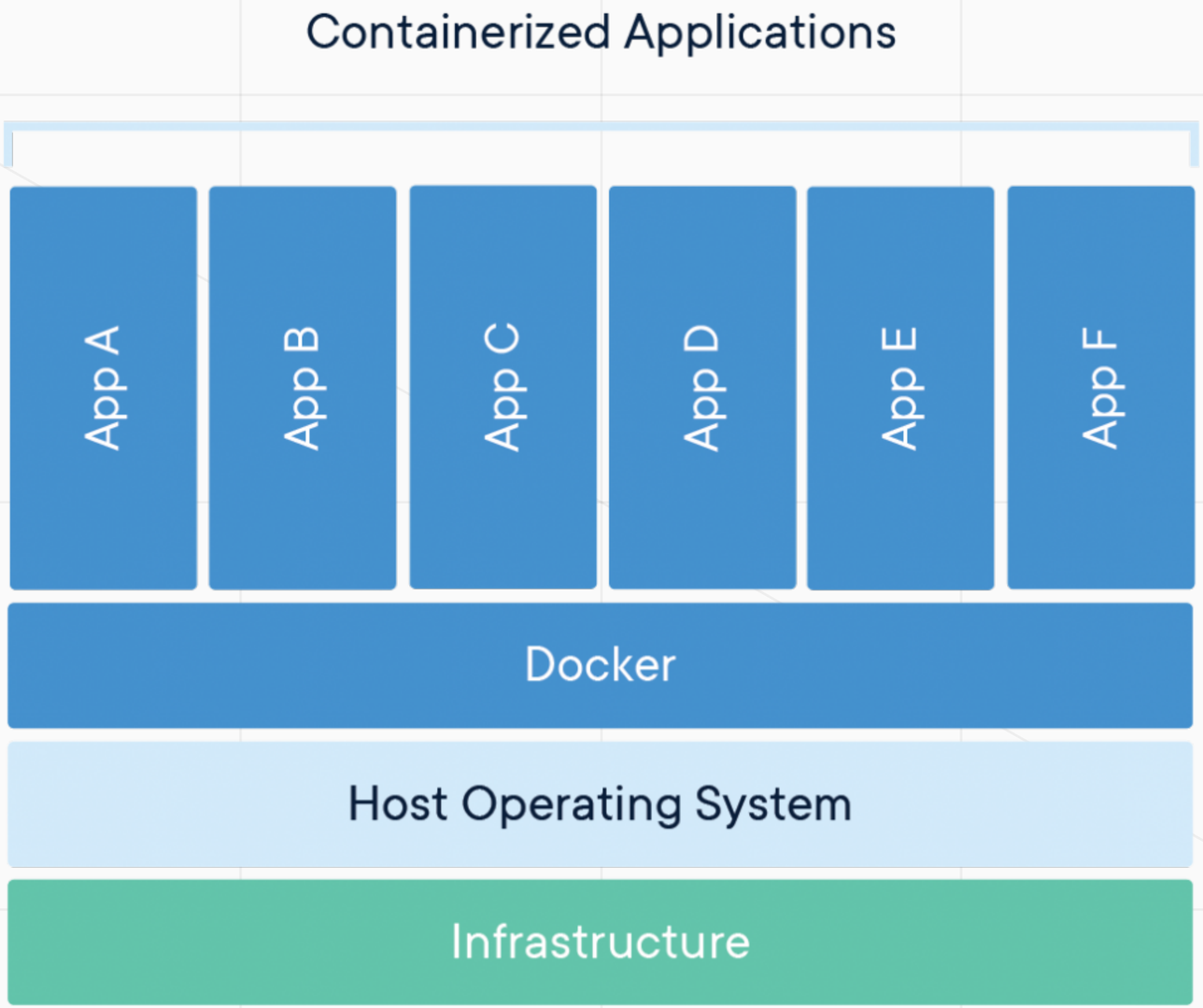
- Build images from source code
- Serve as setup instructions for the environment
- Define with a **Dockerfile**
- To build image - `docker build .`
- To see all images - `docker images`

Docker Containers

- Run containers from an image
- A live instance of your application
- Define running instructions with additional flags
- To run a container - `docker run`
- To see all containers - `docker ps`

Images : Containers ::

- Classes : Objects
- Blueprints : Houses
- Recipes : Cakes
- Instructions : Productions of these instructions



Source: Docker

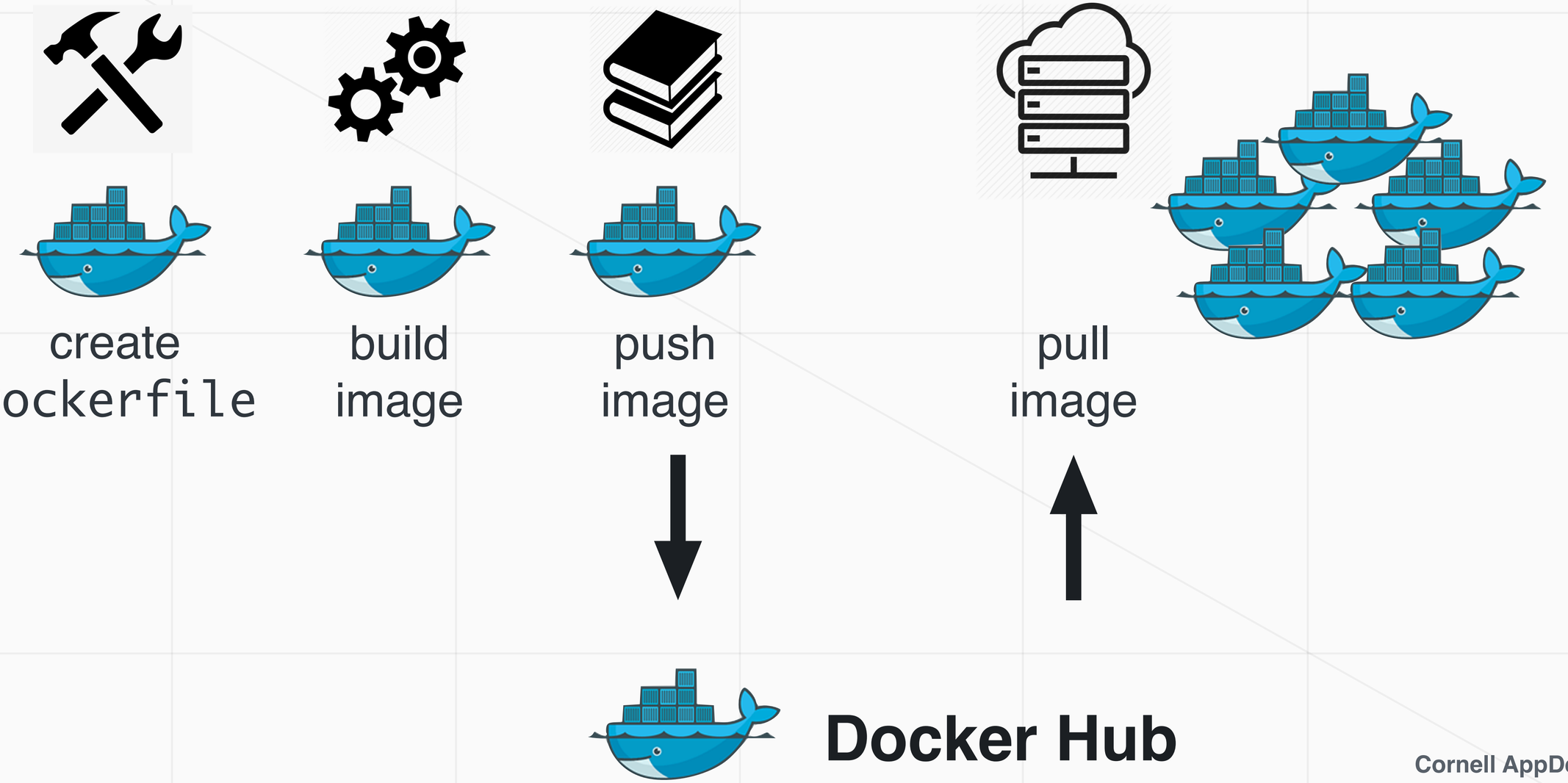
Benefits

- Better application management system
- Requires fewer resources
- Boot up applications faster
- Simplification & Modularity

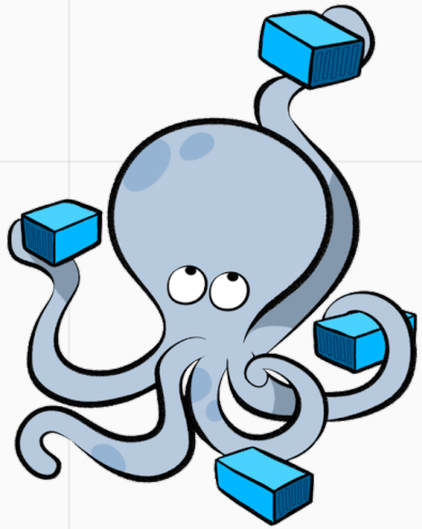
2. Prepare Production Environment to be Run-Ready

Docker Hub

- A cloud-based service
- Stores images in private & public repositories
- code : GitHub :: image : Docker Hub
- Necessary medium for remote servers to access pre-built images

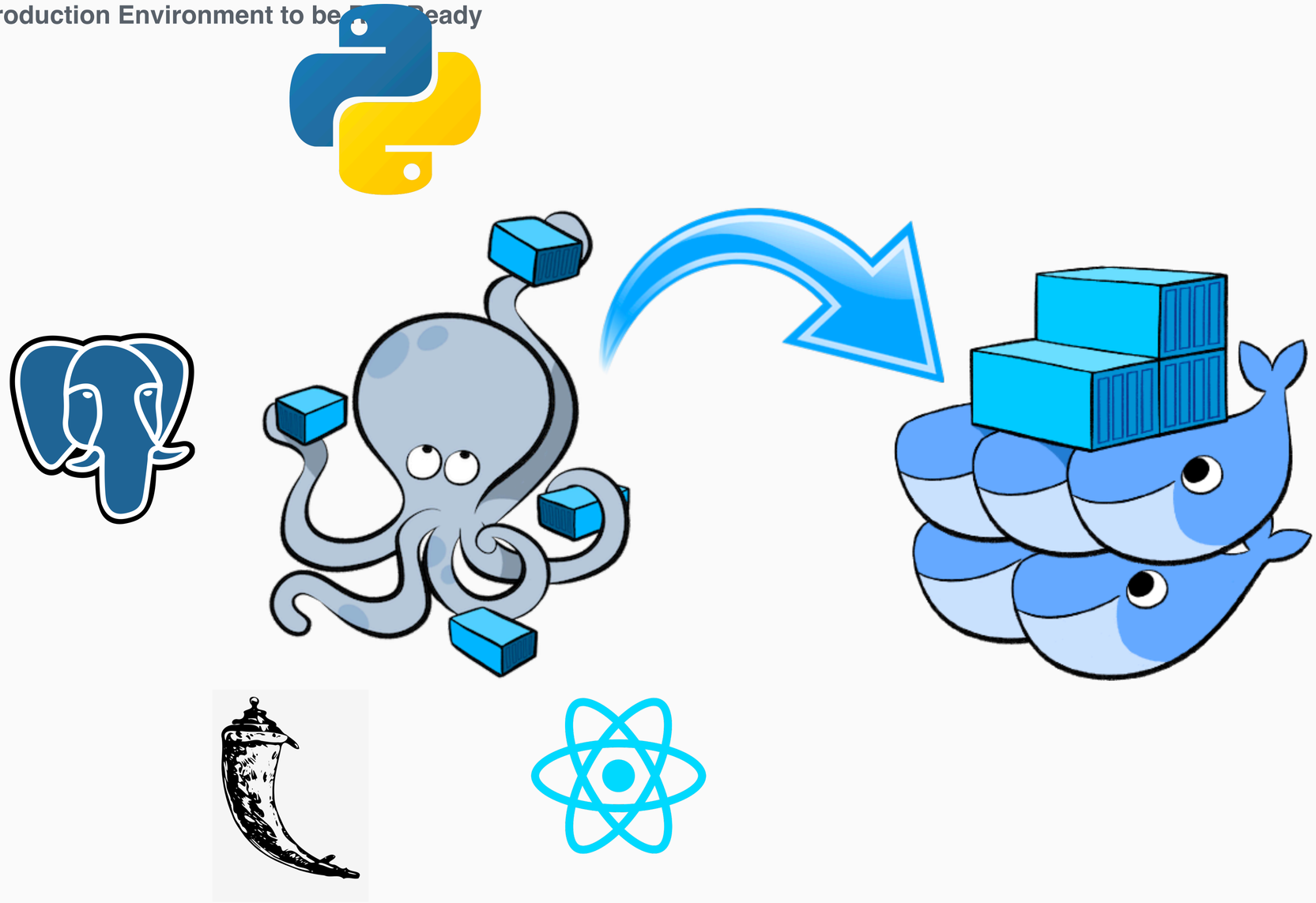

create
Dockerfile
build
image
push
image
pull
image
Docker Hub

Docker Compose



- A tool for defining and running multi-container Docker apps
- Create a `docker-compose.yml` to define application's services
- This course works with just one container

2. Prepare Production Environment to be [illegible] Ready



Summary

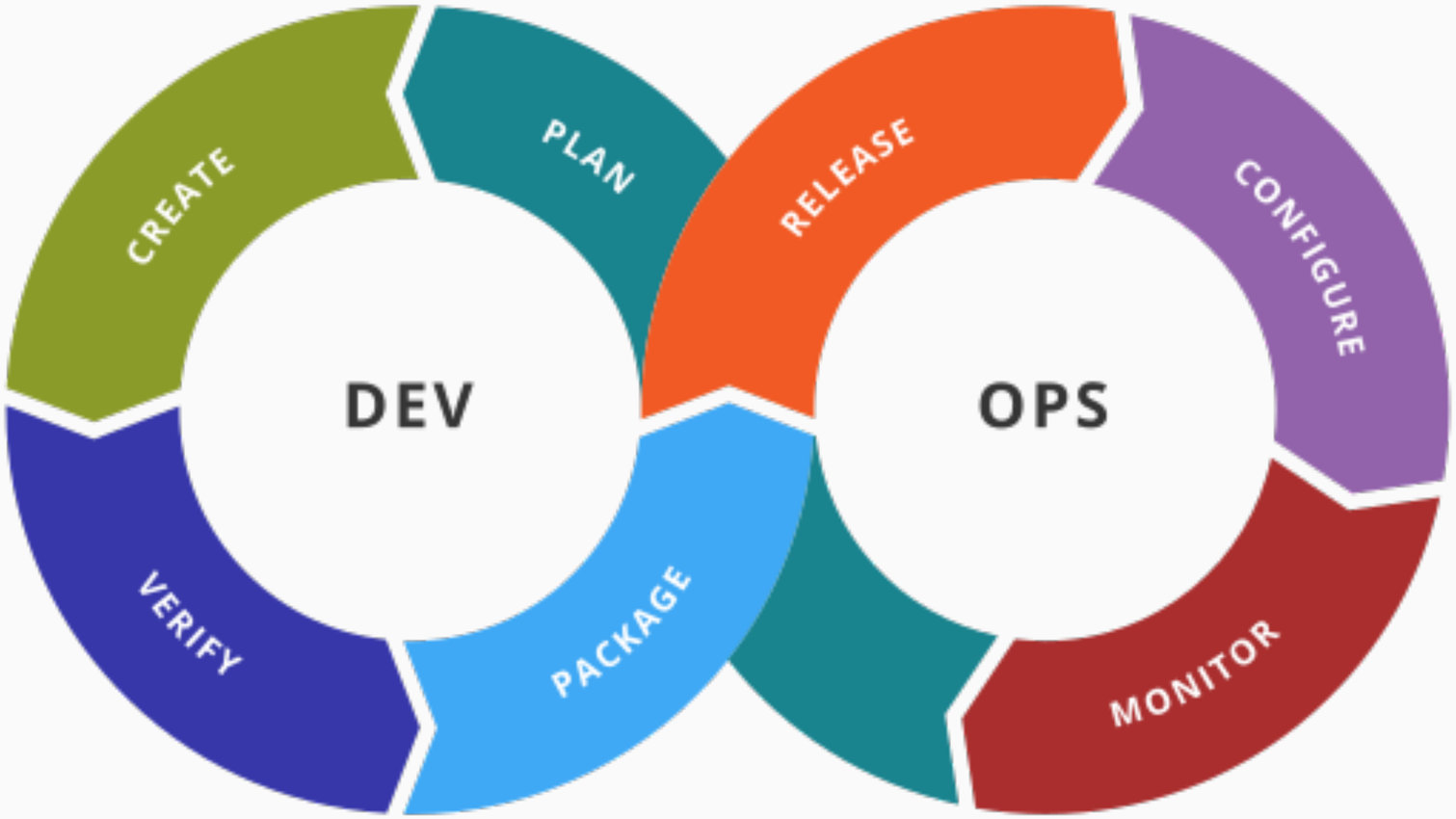
Summary

1. Define a `Dockerfile` to specify app's required environment
2. Build the Docker image
3. Push the image to Docker Hub
4. Define how to run your image(s) in a `docker-compose.yml`
5. Deploy

DevOps

Ship Software Fast

- Development: building software (what we've been doing)
- Operations: testing and releasing software
- DevOps: the boundary where these two meet
- Goal: build and ship software quickly and reliably
- Make development and operations flows seamless



Source: Consultadd

Demo