Check-in Code

for

Essence of Backend Engineering: Dock...

docker!

ACM at UCSD

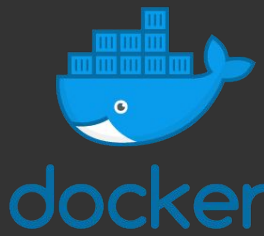
members.acmucsd.com

Check-In Code: docker!

Essence of Backend Engineering: Docker

Taught by Nikhil

Slides: https://acmurl.com/docker-slides



Check-In Code: docker!

ACM at UCSD

docker!

Check-In
members.acmucsd.com

Check-In Code: docker!

ACM at UCSD

Docker Installation

Windows: Docker Desktop for Windows

- Note: Windows 10 Home users must have WSL 2 (Windows Subsystem for Linux 2) before installing Docker Desktop
- WSL 2: WSL Install and WSL 2 Upgrade

Mac: Docker Desktop for Mac

Linux: Ubuntu Installation

- (instructions for other distros can be found on the left side of the website)

Also create a free account on Docker Hub (hub.docker.com)

Check-In Code: docker!

ACM at UCSD

today's agenda

1. **Docker Basic Concepts**
   Containers, VMs, images, and how Docker fits in
2. **Docker CLI**
   All of Docker at your fingertips
3. **Dockerfiles**
   Creating our own application image
4. **Publishing Images to Docker Hub**
   Pushing our image to Docker's container registry

Check-In Code: docker!

ACM at UCSD

Docker Basic Concepts

Containers, VMs, and how Docker fits in

Check-In Code: docker!

ACM at UCSD

What is Docker?

From the docs: "Docker is an open platform for developing, shipping, and running applications"

Docker essentially makes developing, shipping, and running applications easy by letting you:

- Pull **images** of *any* OS or platform needed (e.g. Ubuntu, Node.js, MySQL, etc)
- Build your own application into an image
- Run images as isolated **containers** that serve the application, *and that can run on any machine with Docker installed*

Check-In Code: docker!

ACM at UCSD

Before the days of Docker

Back before Docker (and any containerization technology) existed, all we had were physical servers

- If you wanted to host your application on a server, you'd have to:
  - Find a physical computer
  - Configure it to run that application

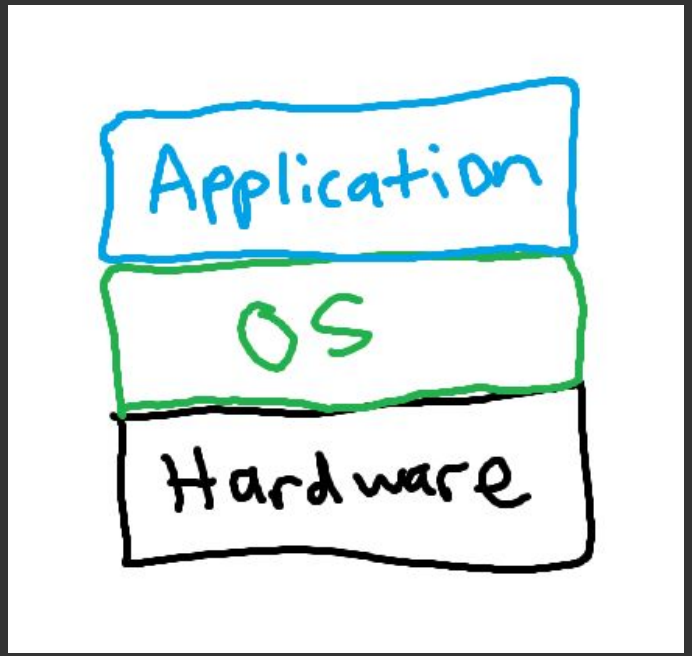
Each physical server has only one of its own:

- OS
- CPU
- RAM
- Hard drive

Check-In Code: docker!

ACM at UCSD

Hardware Layer Visualization



Check-In Code: docker!

ACM at UCSD

So what's the problem with hardware?

Having a dedicated physical server sounds good, but what's the issue?

The Problem: what if we want to run another server?

- e.g. we want another application with its own dedicated OS, memory, filesystem, etc.
- We would need to buy another physical server and configure it to run our second application
- This is ridiculously expensive!!!!

Check-In Code: docker!

ACM at UCSD

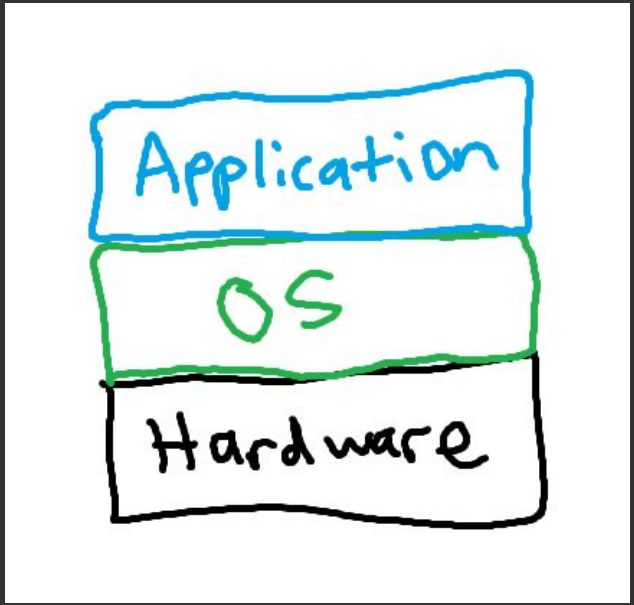
The Solution: Virtualization & Virtual Machines

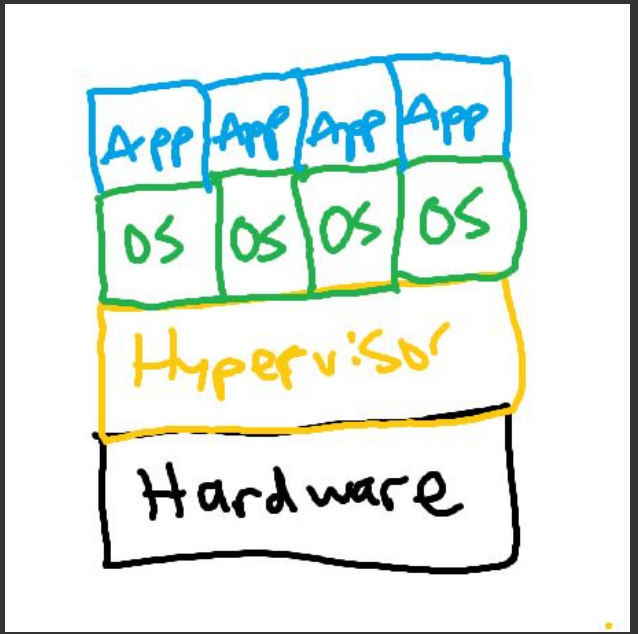
Virtualization: running an instance of an operating system on another operating system

- The OS's are separated from the hardware by a layer called a **hypervisor**
- The hypervisor is in charge of splitting up resources (e.g. CPUs, RAM, Hard drive space) to all OS's (or **virtual machines**)
  - e.g. VMWare is a popular product for helping spin up new VM's
- Virtual machines allow multiple applications to be ran with their own dedicated resources

Check-In Code: docker!

Virtualization Layer Visualization





Check-In Code: docker!

ACM at UCSD

So what's the problem with virtualization?

Having multiple VM's on one OS sounds good, but what's the issue?

The Problem: Running multiple operating systems is ***really*** slow

- How long does it take to boot up your computer?
- It would take AT LEAST as long for each VM to boot up, but most certainly longer since they'd have divided resources

A good solution would be something that supports running multiple applications in isolation without having to house multiple OS's

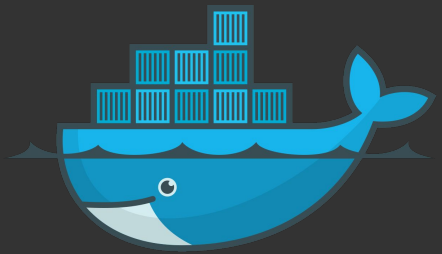
Check-In Code: docker!

ACM at UCSD

The Solution: Containers!

Container: a light-weight, isolated execution environment for an application

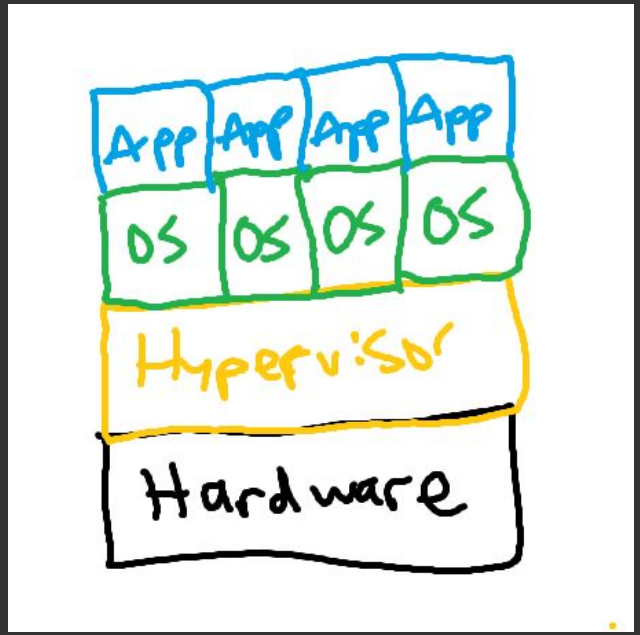
- Containers are essentially lightweight VMs
- Docker provides a virtualization layer over the OS rather than the hardware
- This means the **kernel** (the interface between hardware and software) does not need to be duplicated for every container
- This saves tons of disk space and boot-up time

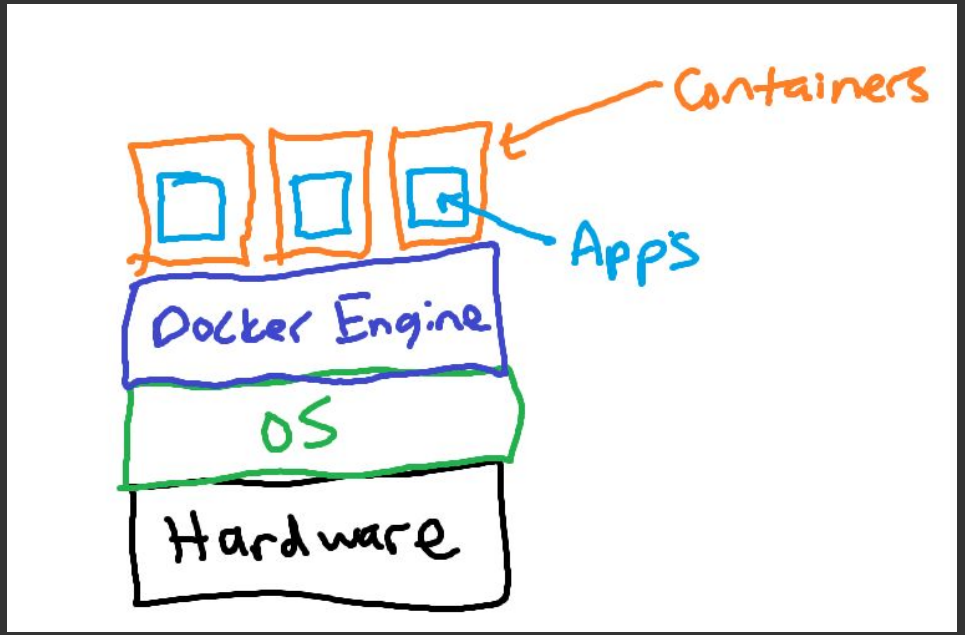


Check-In Code: docker!

ACM at UCSD

Docker Container Layer Visualization





Check-In Code: docker!

ACM at UCSD

Docker Images

Docker Image: a collection of config and execution parameters for running a Docker container

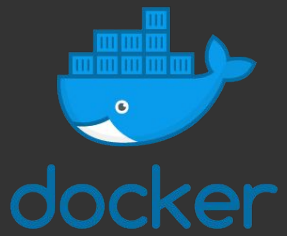
- Images are ran using Docker Engine to spin up containers of an app
- Free images exist on online registries (like Docker Hub) such as:
  - Node.js
  - PostgreSQL
  - Java
  - Ubuntu
  - ... and thousands of others

ACM at UCSD

Check-In Code: docker!

Docker Conceptual Recap

- Docker consists of:
  - **Images**: a collection of configs for running a Docker container
  - **Containers**: an isolated, lightweight environment executing the application
- Docker solves the problems of:
  - High costs of hardware servers
  - Time inefficiency of VMs
  - The "it works on my machine" dilemma



Check-In Code: docker!

ACM at UCSD

Docker CLI

All of Docker at our fingertips

Check-In Code: docker!

ACM at UCSD

Quick Pause! Do you have Docker installed?

- Windows: Docker Desktop for Windows
  - Note: Windows 10 Home users must have WSL 2 (Windows Subsystem for Linux 2) before installing Docker Desktop
  - WSL 2: WSL Install and WSL 2 Upgrade
- Mac: Docker Desktop for Mac
- Linux: Ubuntu Installation
  - (instructions for other distros can be found on the left side of the website)

Check-In Code: docker!

ACM at UCSD

Docker CLI

A command-line interface for creating, pulling, and running Docker images and containers

Key commands:

- `docker pull <image>` → pull an image from Docker Hub
- `docker run --name <name> <image>` → run a local image
  - Useful flags: -d (run in background), -t (creates terminal in container), -p (specify ports)
- `docker ps` → view all running containers
- `docker images` → view all local images
- `docker exec <image>` → execute a command inside the container

Check-In Code: docker!

ACM at UCSD

Docker CLI Demo

Let's pull and play around with some images, namely:

- Node.js → a JavaScript runtime
- PostgreSQL → a relational database management system
  - Notes:
    - ports must be specified so that the database can be reached
    - a password must be specified when the container is ran with -e
    - a username must be specified when running `docker exec` with -U
  - docker pull postgres:alpine
  - docker run --name postgres-0 -e POSTGRES_PASSWORD=password -d -p 5432:5432 postgres:alpine
  - docker exec -it postgres-0 psql -U postgres

Check-In Code: docker!

ACM at UCSD

Dockerfiles

Creating our own application images with Dockerfiles

Check-In Code: docker!

ACM at UCSD

What are Dockerfiles?

Dockerfile: a text-based file that contains script instructions for building Docker images

- Think of it as a cooking recipe for making Docker images!
- Format:

```
# comments start with a '#'

INSTRUCTION arguments
```

- With `docker build -t <image-name> .`, Docker will use the instructions and configurations in the Dockerfile to build the image with a tag name that's associated with the image.

Check-In Code: docker!

ACM at UCSD

Example Dockerfile

```
# builds image from node version 16
FROM node:16

# creates app directory where source code will go
WORKDIR /usr/src/app

# copy package.json & package-lock.json files to the image working directory
COPY package*.json ./

# installs dependencies from package files using npm
RUN npm install

# copies source code to inside docker image
COPY . .

# exposes 8080 port needed for app
EXPOSE 8080

# command for running the app
CMD [ "node", "server.js" ]
```

Tip: We want to make sure we copy the package*.json files over before running npm install and copying the files, so that we cache the installation process!

Check-In Code: docker!

ACM at UCSD

Dockerfiles (cont.)

Here are some other things to note about Dockerfiles:

- The first instruction of a Dockerfile **must** be a FROM instruction
  - Dockerfiles build new images using a base or parent image
- Common instructions include: FROM, RUN, COPY, WORKDIR, CMD
- Only one CMD instruction is allowed, otherwise only the last CMD instruction is used
  - CMD vs. RUN: CMD should be used to tell the container what command/executable should be run when it's started (i.e. node, python, etc.), while RUN is used to execute any other necessary commands like installing dependencies

Check-In Code: docker!

ACM at UCSD

Your Turn: Let's create a Docker image!

We'll be Dockerizing a simple Node.js web app found [at this Github repo]

What you'll be doing:

- Cloning the repo using Git
- Creating a Dockerfile with the following specs:
  - Use node version 18
  - Have the working directory as "/app"
  - Expose port 3000
  - Use caching techniques for npm install
- Run `docker build -t <image-name> .` to build the image
- Run `docker run --name sample-app -p 3000:3000 <image-name>` to see your container in action!

Hint: This is what we mean by caching techniques for npm install!

```
# copy package.json & package-lock.json files
COPY package*.json ./

# installs dependencies from package files using npm
RUN npm install

# copies source code to inside docker image
COPY . .
```

Check-In Code: docker!

ACM at UCSD

Pushing Images to Docker Hub

Check-In Code: docker!

ACM at UCSD

Pushing Images

Pushing an image to Docker Hub allows you to share your image publicly and enables others to download it and run an instance on their local machines.

How to push an image:

- Login to Docker CLI using `docker login`
- Make sure that your Docker image is tagged correctly with the format "your-docker-username/image-name:tag-name"
  - You can tag an image using `docker tag <current name> <new name>`
- You can then push it to Docker Hub using `docker push <tag>`
- Visit hub.docker.com to view your pushed image!

Check-In Code: docker!

ACM at UCSD

Your Turn: Now you try!

Now we'll have you try to push your image onto Docker Hub. Make sure that you have a Docker Hub account set up.

- If you haven't already, make sure that your image's tag matches the format on the previous slide.
- Make sure that you're in the correct directory for your image when running the Docker commands.
- Once it's pushed, try running it on https://labs.play-with-docker.com/!
  - Log in with your Docker Hub credentials and click on "Add New Instance".
  - Run the app with `docker run --name sample-app -p 3000:3000 <image-name>`

Check-In Code: docker!

ACM at UCSD

Extending Docker

- We Dockerized a simple application. Is there more to it?
- Docker Compose
  - Multi-container applications
  - Just a single yaml file
- Kubernetes
  - Container Orchestration
  - Easy to maintain, test, scale

Check-In Code: docker!

ACM at UCSD

Thanks for Coming!

Request for future workshop topics at:

acmurl.com/hack-workshop-requests

Check-In Code: docker!

ACM at UCSD