Intro to Containers (Docker)

With Riley Flynn

My Background

- Cloud Architect @ CoLab
- On year 9 (and counting) of my degree at MUN
- Been using Docker for 8 years
- These days, running an overkill home server setup using containers



Quick notes before starting

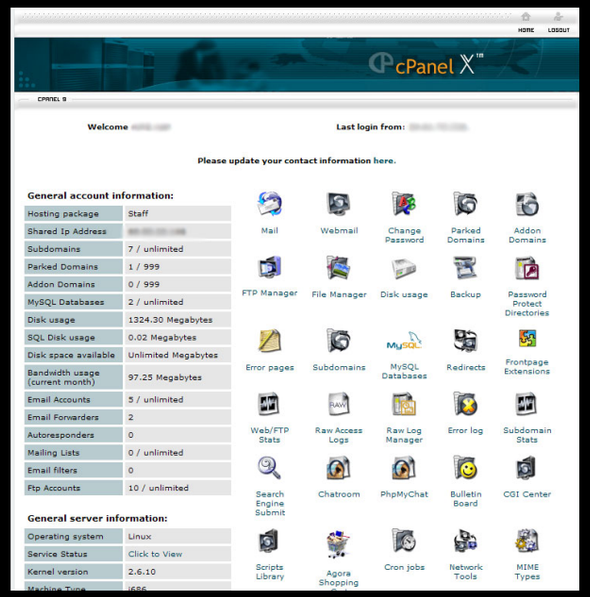
- There will be a demo at the end, but there is no participation component
- If anyone has questions at any point, feel free to raise your hand
  - I will also take questions at the end

How we got here

- Story starts in 2005
- Largely have 2 options for hosting software

Shared Web Hosting

- Cheap
- Accessible
- Limited in terms of support



- The kind of web hosting where you get a control panel, some credentials to upload files, and *maybe* a way to one-click install apps
- Doesn’t fit well for any non-website use
- Restricted to whatever runtime your host has. Have no control over versions
- Companies like GoDaddy, 1&1, Hostgator, etc.

Virtual / Dedicated Server

- More expensive
- Less approachable
- Significantly more flexible



- Requires you to take care of maintaining servers
- Can run any software you want, so no worries about runtime versions or workloads being well-suited to running on a web server
- Providers like AWS

Zimki

- Launched in 2006 by Fotango
- Allowed devs to write simple JS functions, which would be ran on-demand on Fotango's servers
- First instance of what's now referred to as platform-as-a-service

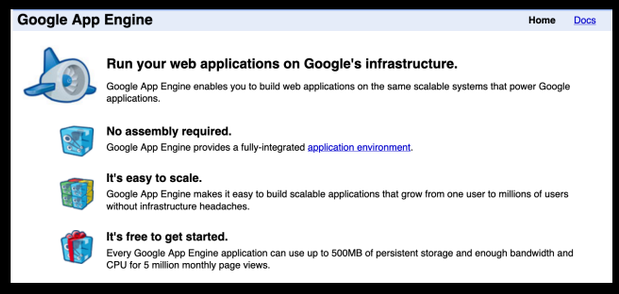
```
function hello() {
    return "HELLO WORLD";
}
zimki.publishPath("/", hello);
```

- Fotango was an image sharing site owned by Canon
- Platform-as-a-Service: an in between of the two types of earlier hosting. Host provides a platform to run your code on and handles maintaining the underlying hardware and software - you just write your code and run it
- Also first instance of what people these days refer to as Serverless

PaaS explosion begins

- Many platform-as-a-service companies launch, each targeting a specific language
- Examples:
  - Heroku: Ruby
  - Google App Engine: Python





- Many other examples, but they were the largest and will be the names followed for this story

DotCloud launches

- Advertises itself as the first “multi-language PaaS”
- Supports a significantly wider variety of languages & databases than their competitors



- Call out some examples
    - Python, Ruby, Java, PHP, Node.js
    - MySQL, SQLite, PostgreSQL

“We support major application servers, databases and message buses. If we don't support it, ask.”

DotCloud homepage (January 2011)

- Unlike other providers, DotCloud makes it a point of calling for customers to ask them for anything they don’t support
- This isn’t just a case of them supporting a couple extra languages than their competitors

Other PaaS platforms adapt

- Heroku releases a new platform version, Celadon Cedar, completely rearchitecting how the product worked on the backend and adding support for Node.js
- Google App Engine releases from beta, supporting 2 additional languages

- Despite this, DotCloud *always* stays ahead of them in language support

“Cedar has many new features and updates, but most importantly introduces a new way of thinking about web application architecture and scaling apps on Heroku: the process model. Check it out: …

- Official support for Node.js”

Heroku Celadon Cedar announcement (May 2011)

- Heroku wouldn’t go on release another platform version until the release of Fir earlier this year

“Your application can run in one of three runtime environments: the Go environment, the Java environment, and the Python environment”

Post-release Google App Engine docs (September 2011)

PyCon 2013, Day 1

- Solomon Hykes presents a 5 minute, unscheduled lightning talk
- Reveals to the world the tech DotCloud had built to allow them to keep ahead of their competitors on supporting new languages



- PyCon is the yearly Python conference
- Lightning talks occur at the end of every day
- Each talk is 5 minutes and are not scheduled ahead of time
- People wishing to give a lightning talk can enter their idea at lunch time of each given day, and the organizers will pick people to speak that evening

Introducing: Docker

- The announcement is linked on the resources site that I'll show at the end

Docker

- Open source tool for running containers on Linux
- Follows a client/server model
  - Exposes an API on your machine
  - All tools interact with Docker via this API

- Linux detail is key - if you’re running Docker on macOS or Windows, there is a Virtual Machine running Linux somewhere under the hood
- API by default is just exposed as a file on your machine

Containers

- A collection of one or more processes, running isolated from the rest of the system
- They are:
  - Ephemeral
  - Self-contained

- Ephemeral: no persistent state
- Self-contained: cannot interact with any files on the host unless you explicitly permit it - therefore will contain all dependencies needed except the Linux kernel

Why would you use it?

- Simplifies running & distributing complicated software
- Provides a *reasonable* level of isolation between a program and the rest of your system

- Previously, you'd have to either put in a bunch of effort packaging software for distributions, write scripts a mile long, or write documentation a mile long
    - Now can just chuck all dependencies into a container, and it should run anywhere
    - Especially with tools like Docker Compose, greatly simplifies running things with many components
- Doesn't provide comprehensive security, but for many purposes it's fine

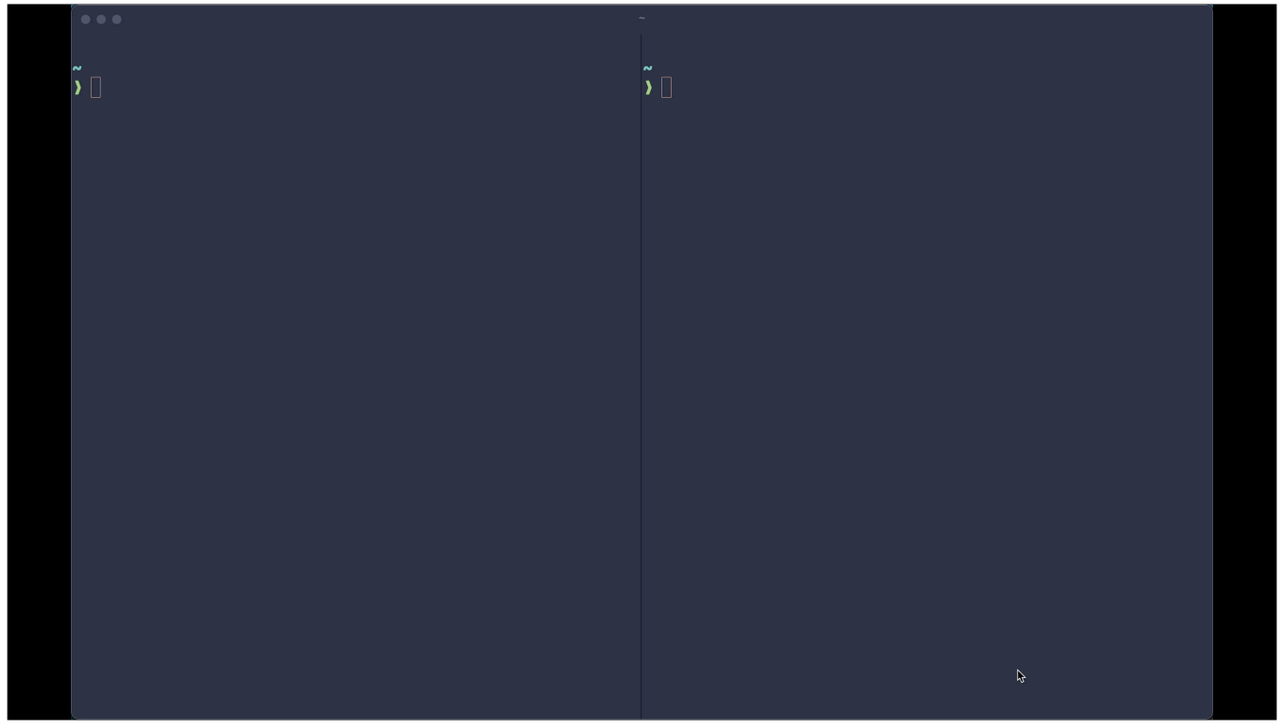
Examples

- Quickly test some commands on an Ubuntu-like environment, without running a full Ubuntu VM
  - docker run --rm -it ubuntu:24.04
- Run a Postgres server, without needing to install Postgres on your system
  - docker run --rm -p 5432:5432 -e POSTGRES_PASSWORD=techni postgres:17

- First command:
  - Rm: Deletes the container once it exits
  - I: Makes the container interactive (eg. Allows you to interact with it from your local terminal)
  - T: Needed to make the container interactive
  - Ubuntu:24.04: Run Ubuntu 24.04
- Second command:
  - Rm: Deletes the container once it exists
  - P: Makes port 5432 in the container available via port 5432 on the host
  - E: Sets an environment variable (in this case, passes a password for Postgres to use)
  - Postgres:17: Run Postgres 17



- Starting Ubuntu
- uname -a shows I’m on Linux
- /etc/os-release shows I’m on Ubuntu 24.04
- Can run all my apt commands as if I were on an actual Ubuntu system



- Attempt connecting to Postgres on port 5432, get connection rejected
- Start up a Postgres container
- Try again, it works
- Can run queries

Container Images

- The data used to create a container
- Consists of:
  - The files needed to run the container
  - Some basic metadata around how to run the container
- Made up of layers
- Identified by a name and a tag
  - Example: postgres:17

- Metadata example: on start, run command X within the container
- Layers enable deduplication and more efficient caching
- Tag: image name is Postgres, and the tag is 17
- Tags are used for things like allowing addressing individual versions

Container Registries

- Services speaking standardized APIs, responsible for distributing container images
- Examples:
  - hub.docker.com: Docker Hub (default)
  - ghcr.io: GitHub Container Registry

- Many other registries, anyone can make their own

Image Creation

- Created via a file called a Dockerfile
- Consists of a number of fairly simple instructions, such as:
  - FROM: Specifies the image to build from
  - COPY: Copies files from the local system to the container
  - RUN: Runs a command in the container
  - ENTRYPOINT / CMD: Specifies the command to run when launching and the args to pass to it

- Command list is non-exhaustive
- Examples for usage of each:
    - FROM: Saying “base this off the Node.js base image”
    - COPY: Copy all my source code into the image
    - RUN: Run npm install
    - ENTRYPOINT: Run

Deploying With Docker

- Write your app as usual
- Write a Dockerfile to produce a container image for your app
- Publish your image to a container registry
- Pull the image from your container registry and start a container

- How you do the last step will depend on how exactly you're managing your containers
- If you're using a service that expects a Docker image, it is doing that for you behind the scenes - you just point it at the container registry

Demo time!

- For the demo, will be taking an existing Node.js web app, written with TypeScript and containerizing it

Docker is basically just a
wrapper around Linux features

cgroups

- Allows creating process groups for the purposes of resource usage calculation and constraint
- Used to allow:
  - Seeing the resource usage of a given container
  - Configuring resource limits for a container

- Example: constraining a container to a single gig of ram

Namespaces

- Used to partition the different types of resources managed by the Linux kernel (processes, users, networking, storage, etc.)
- This is what makes containers unable to see other things running on the system

- Docker provides tools to tweak how the namespaces are used
- For example, attaching multiple containers to the same network namespace so they can communicate with each other

chroot

- Enables changing the root of the filesystem
- Used to make the root of the filesystem in the container be the container’s image

- Container’s disk will exist somewhere on the host system, but as far as the container can tell, it’s the root

Overlay filesystems

- Creates a transparent overlay on top of underlying files, where any writes are stored in the overlay rather than the underlying file
- Used for writes to files from within the container, prevents them from messing with the underlying image

As a result, starting a container…

- Creates an overlay filesystem in a unique directory, pointing at the original image
- Create dedicated namespaces for all relevant Linux resources
- Create a new cgroup
- chroot into the overlay filesystem's directory
- Launch the container's entry point, attaching the new namespaces and cgroup



- 5 minute long blog post, walks you through basically starting a very primitive container, entirely manual via shell commands
- Link to the post will be in the resources linked at the end, worth a read

Docker’s technologies have been used in unexpected ways

OCI Artifacts

- Spec to use container registries to distribute arbitrary files
- Examples of users:
  - Ollama: used to distribute models
  - Homebrew: used to distribute binary packages

- Internet did what internet does
- Saw the abundance of cheap / free storage offered by container registries and repurposed it for their own needs
- Also used by things like SBOM / attestation providers

bootc

- Software for building and distributing Linux distros as Docker images
- Used by distros such as:
  - Fedora Silverblue
  - Bazzite

- Avid PC gamers may have heard of Bazzite
- The distro is described by a “Containerfile” (a legally-distinct Dockerfile)
- Updating your distro to a newer version is just changing the tag of the image to use and rebuilding the local state

Questions?



- QR code contains links to everything included in this talk, including copies of the talk slides
- Will be heading to Jumping Bean on Elizabeth Avenue after until 9PM for CTS-NL