This Section

- All about images, the building blocks of containers
- What's in an image (and what isn't)
- Using Docker Hub registry
- Managing our local image cache
- Building our own images
- VOLUME and mounting host data
- Images for different CPU architectures
- Windows images and how they differ

What's In An Image (And What Isn't)

- App binaries and dependencies
- Metadata about the image data and how to run the image
- Official definition: "An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime."
- Not a complete OS. No kernel, kernel modules (e.g. drivers)
- Small as one file (your app binary) like a golang static binary
- Big as a Ubuntu distro with apt, and Apache, PHP, and more installed

Image Creation and Storage

- Created Using a `Dockerfile`
- Or committing a containers changes back to an image
- Stored in Docker Engine image cache
- Move images in/out of cache via:
  - local filesystem via tarballs
  - push/pull to a remote "image registry" (e.g. Docker Hub)
- Images aren't ideal for persistent data
  - Mount a host file system path into container
  - Use docker volume to create storage for unique/persistent data

Image Highlights

- Images are made up of app binaries, dependencies, and metadata
- Don't contain a full OS
- Usually use a Dockerfile recipe to create them
- Stored in your Docker Engine image cache
- Permanent Storage should be in a Image Registry
- Image's don't usually store persistent data

This Lecture

- Basics of Docker Hub (hub.docker.com)
- Find Official and other good public images
- Download images and basics of image tags

This Lecture: Review

- Docker Hub, "the apt package system for Containers"
- Official images and how to use them
- How to discern "good" public images
- Using different base images like Debian or Alpine
- The recommended tagging scheme used by Official images

This Lecture

- Image layers
- Union file system
- `history` and `inspect` commands
- Copy on write

Image and Their Layers: Review

- Images are made up of file system changes and metadata
- Each layer is uniquely identified and only stored once on a host
- This saves storage space on host and transfer time on push/pull
- A container is just a single read/write layer on top of image
- `docker image history` and `inspect` commands can teach us

This Lecture: Requirements

- Know what container and images are
- Understand image layer basics
- Understand Docker Hub basics

This Lecture

- All about image tags
- How to upload to Docker Hub
- Image ID vs. Tag

This Lecture: Review

- Properly tagging images
- Tagging images for upload to Docker Hub
- How tagging is related to image ID
- The Latest Tag
- Logging into Docker Hub from docker cli
- How to create private Docker Hub images

Building Images: Requirements

- Understand container and image basics from previous lectures
- Cloned the class repository from Section 1
- Starting in the dockerfile-sample-1 directory

Building Images: The Dockerfile Basics

- Dockerfile basics
- FROM (base image)
- ENV (environment variable)
- RUN (any arbitrary shell command)
- EXPOSE (open port from container to virtual network)
- CMD (command to run when container starts)
- docker image build (create image from Dockerfile)

Assignment: Build Your Own Image

- Dockerfiles are part process workflow and part art
- Take existing Node.js app and Dockerize it
- Make `Dockerfile`. Build it. Test it. Push it. (rm it). Run it.
- Expect this to be iterative. Rarely do I get it right the first time.
- Details in `dockerfile-assignment-1/Dockerfile`
- Use the Alpine version of the official '`node`' 6.x image
- Expected result is web site at http://localhost
- Tag and push to your Docker Hub account (free)
- Remove your image from local cache, run again from Hub