1 ☑ **Docker Mastery**
bretfisher.com/dockermastery
twitter.com/bretfisher

2 ☑

3 ☐ **Assignment Reminder**

4 ☐ **Why Docker? Why Now?**
- Speed. Speed. Speed.
- Making dev->test->prod easier and faster
- "make my code run the same way on two different machines"
- Reduce complexity of developing code for distributed systems
- Reduce complexity of deploying code to the cloud
- Reduce complexity of updating code in the cloud
- No one explains the "why" better then Docker's founder and CTO Solomon Hykes
  - "Why we build Docker" from dotScale conference in 2013

5 ☐ **Docker Editions**
- Learn about the various (dozen+) Editions of Docker
- Learn which to use for this course
- Learn Docker CE vs. EE
- Learn Stable vs. Edge releases

6 ☐ **Docker Editions at store.docker.com**
- Docker is no longer just a "container runtime"
- Docker moves fast, it matters how you install it
- Docker CE (Community Edition)
- Three major types of installs: Direct, Mac/Win, Cloud
- Linux (different per distro) (don't use default package)
- Docker for Windows (or legacy Docker Toolbox)
- Docker for Mac (or legacy Docker Toolbox) (don't use brew)
- Docker for AWS/Azure/Google

7 ☐ **CE vs. EE, Stable vs. Edge**
- Docker CE (free) vs. Docker EE (paid)
- EE = Enterprise Edition
- EE = Support + extra products
- EE = Certified on specific platforms

- docker.com/pricing

8 **CE vs. EE, Stable vs. Edge Cont.**

- Edge (beta) released monthly, Stable quarterly
- Edge gets new features first, but only supported for a month
- Stable rolls in three months of Edge features, EE supported longer

9 **Docker on Windows**

- Learn the two types of containers Windows can run
- Learn which Docker Edition to install on your Windows version
- Learn differences between Windows 10 and Windows Server 2016

10 **Docker on Windows Overview**

- Two Types of Containers: Linux Containers and Windows Containers
- Linux Containers still default, when I say "containers" I mean Linux
- Best experience: Docker for Windows, but Win10 Pro/Ent only
- Win7/8/8.1 or Win10 Home should use Docker Toolbox
- Windows Server 2016 also supports Windows Containers
- Getting better all the time
  - e.g. Native Linux containers coming soon

11 **Docker on Windows 10 Pro/Ent**

- Use Docker for Windows, from store.docker.com
- More features then just a Linux VM
- Uses Hyper-V with tiny Linux VM for Linux Containers
- PowerShell native

12 **Docker for Mac**

- Use Docker for Mac, from store.docker.com
- More features then just a Linux VM
- Uses "xhyve" with tiny Linux VM for Linux Containers
- Terminal/iTerm native
-
- PowerShell native

13 **Docker on Windows 7/8 and 10 Home**

- Use Docker Toolbox, not as fancy as Docker for Windows
- Like others, download at store.docker.com
- Runs a tiny Linux VM in VirtualBox via docker-machine
- Uses a bash shell to make it more like Linux/Mac options

- Does not support Windows Containers

14 **Docker on Windows Server 2016**

- Windows Server 2016 supports native Windows Containers
- ”Docker for Windows” runs on Win 2016 but not required
  - Only do this for when you run Win 2016 locally for dev/test. NOT for prod
- No options for previous Windows Server versions
- Hyper-V can still run Linux VM's (that can run Docker) just fine

15 **Docker for Windows: Setup**

- Install Docker
- Tweak Docker for Windows settings
- Clone my GitHub repo
- Get a code editor
- Tweak your terminal and shell (optional)

16 **Docker for Windows Tips**

- PowerShell Command Completion: posh-docker
  - https://docs.docker.com/docker-for-windows/#set-up-tab-completion-in-powershell
- Code paths enabled for Bind Mounts (C:\Users by default)
- Bind Mounts work for code (but often not databases)
- Backup option: use docker-machine create --driver hyperv
  - https://docs.docker.com/machine/drivers/hyper-v/
- Great Dockerfile/Compose file editor: Visual Studio Code
  - https://code.visualstudio.com/
- Great Terminal UI replacement: cmder
  - http://cmder.net/
- Great info and troubleshooting/FAQ
  - https://docs.docker.com/docker-for-windows/

17 **Docker Toolbox on Windows: Setup**

- Install Docker
- Clone my GitHub repo
- Start the Docker Quickstart Terminal
- Tweak Docker VM settings
- Get a code editor
- Tweak your terminal and shell (optional)

18 **Docker Toolbox on Windows: Tips**

- Use the Docker Quickstart Terminal to start with
  - In background it auto-creates and auto-starts VM
  - Defaults to bash shell
- Code paths enabled for Bind Mounts work in C:\Users only
- Bind Mounts work for code (but often not databases)
- Re-create Linux VM or create more with docker-machine
- Great Dockerfile/Compose file editor: Visual Studio Code
  - https://code.visualstudio.com/
- Great Terminal UI replacement: cmder
  - http://cmder.net/

19 **Docker on macOS Overview**

- Docker for Mac
  - Requires Yosemite 10.10.3 (2014 release)
  - Yosemite works with 2007-2008 Mac's and newer
- Docker Toolbox
  - For Snow Leopard, Lion, Mountain Lion (10.6-10.8)
- Docker in a Linux VM
- Docker in a Windows VM
  - Not usually possible, only works with Vmware Fusion
- Don't use homebrew (brew install docker), it's docker CLI only

20 **Docker for Mac: Setup**

- Install Docker
- Tweak Docker for Mac settings
- Clone my GitHub repo
- Get a code editor
- Tweak your terminal and shell (optional)

21 **Docker for Mac Tips**

- Bash Command Completion
  - https://docs.docker.com/docker-for-mac/#installing-bash-completion
- Code paths enabled for Bind Mounts (/Users by default)
- Bind Mounts work for code and (usually) databases
- Run more nodes: docker-machine create --driver
  - Fusion, VirtualBox, Parallels, etc.

https://docs.docker.com/machine/drivers/
- Great Dockerfile/Compose file editor: Visual Studio Code
  - https://code.visualstudio.com/
- Great Terminal replacement: iTerm2
  - https://www.iterm2.com/
- Great info and troubleshooting/FAQ
  - https://docs.docker.com/docker-for-mac/
- My Shell Setup (iTerm2 + oh-my-zsh + much more)
  - http://www.bretfisher.com/shell

22 ☐ **Docker on Linux**
- Easiest install/setup, best native experience
- Three main ways to install: script, store, or docker-machine
- get.docker.com script (latest Edge release)
  - curl -sSL https://get.docker.com/ | sh
- store.docker.com has instructions for each distro
- RHEL officially only supports Docker EE (paid), but CentOS will work
- Installing in a VM, Cloud Instance, all are the same process
- May not work for unlisted distros (Amazon Linux, Linode Linux, etc.)
- Don't use pre-installed setups (Digital Ocean, Linode, etc.)

23 ☐ **Docker for Linux: Setup**
- Install Docker
- Add your user to docker group
- Clone my GitHub repo
- Get a code editor
- Tweak your terminal and shell (optional)

24 ☐ **Docker for Linux Tips**
- After installing Docker, also get docker-compose and docker-machine
  - https://docs.docker.com/machine/install-machine/
  - https://docs.docker.com/compose/install/
- Run more nodes: docker-machine create --driver
  - VirtualBox, AWS, etc. https://docs.docker.com/machine/drivers/
- Great Dockerfile/Compose file editor: Visual Studio Code
  - https://code.visualstudio.com/
- Bash command completion just works

25 ☐ **Section 2**

26 ☐ **Container VS Virtual Machine**

- Containers aren't Mini-VM's
- They are just processes
- Limited to what resources they can access (file paths, network devices, running processes)
- Exit when process stops
- docker run --name mongo -d mongo
- docker top mongo
- ps aux
- docker stop mongo
- ps aux

27 ☐ **This Section**

- Check versions of our docker cli and engine
- Create a Nginx container
- Learn common container commands
-
- Requirements: Have docker installed

28 ☐ **What We Covered**

- docker version
  - verified it's working
- docker info
  - most config values
- docker command line structure
  - docker <command> <sub-command> (options)

29 ☐ **This Lecture**

- image vs. container
- run/stop/remove containers
- check container logs and processes

30 ☐ **Image vs. Container**

- An Image is the application we want to run
- A Container is an instance of that image running as a process
- You can have many containers running off the same image
- In this lecture our image will be the Nginx web server

- Docker's default image "registry" is called Docker Hub (hub.docker.com)

31 **docker container run --publish 80:80 nginx**

1. Downloaded image 'nginx' from Docker Hub
2. Started a new container from that image
3. Opened port 80 on the host IP
4. Routes that traffic to the container IP, port 80

32 **What happens in 'docker container run'**

1. Looks for that image locally in image cache, doesn't find anything
2. Then looks in remote image repository (defaults to Docker Hub)
3. Downloads the latest version (nginx:latest by default)
4. Creates new container based on that image and prepares to start
5. Gives it a virtual IP on a private network inside docker engine
6. Opens up port 80 on host and forwards to port 80 in container
7. Starts container by using the CMD in the image Dockerfile

33 **Example Of Changing The Defaults**

docker container run --publish 8080:80 --name webhost -d nginx:1.11 nginx -T

34 **Assignment: Manage Multiple Containers**

- docs.docker.com and --help are your friend
- Run a nginx, a mysql, and a httpd (apache) server
- Run all of them --detached (or -d), name them with --name
- nginx should listen on 80:80, httpd on 8080:80, mysql on 3306:3306
- When running mysql, use the --environment option (or -e) to pass in MYSQL_RANDOM_ROOT_PASSWORD=yes
- Use docker container logs on mysql to find the random password it created on startup
- Clean it all up with docker container stop and docker container rm (both can accept multiple names or ID's)
- Use docker container ls to ensure everything is correct before and after cleanup

35 **What's Going On In Containers**

- docker container top - process list in one container
- docker container inspect - details of one container config
- docker container stats - performance stats for all containers

36 **Getting a Shell Inside Containers**

- docker container run -it - start new container interactively
- docker container exec -it - run additional command in existing container
- Different Linux distros in containers

37 **Docker Networks: Concepts**

- Review of docker container run -p
- For local dev/testing, networks usually "just work"
- Quick port check with docker container port <container>
- Learn concepts of Docker Networking
- Understand how network packets move around Docker

38 **Docker Networks Defaults**

- Each container connected to a private virtual network "bridge"
- Each virtual network routes through NAT firewall on host IP
- All containers on a virtual network can talk to each other without -p
- Best practice is to create a new virtual network for each app:
  - network "my_web_app" for mysql and php/apache containers
  - network "my_api" for mongo and nodejs containers

39 **Docker Networks Cont.**

- "Batteries Included, But Removable"
  - Defaults work well in many cases, but easy to swap out parts to customize it
- Make new virtual networks
- Attach containers to more then one virtual network (or none)
- Skip virtual networks and use host IP (--net=host)
- Use different Docker network drivers to gain new abilities
- and much more…

40 **Docker Networks: CLI Management**

- Show networks docker network ls
- Inspect a network docker network inspect
- Create a network docker network create --driver
- Attach a network to container docker network connect
- Detach a network from container docker network disconnect

41 **Docker Networks: Default Security**

- Create your apps so frontend/backend sit on same Docker network
- Their inter-communication never leaves host
- All externally exposed ports closed by default
- You must manually expose via -p, which is better default security!
- This gets even better later with Swarm and Overlay networks

42 ☐ **Docker Networks: DNS**

- Understand how DNS is the key to easy inter-container comms
- See how it works by default with custom networks
- Learn how to use --link to enable DNS on default bridge network

43 ☐ **Docker Networks: DNS**

- Containers shouldn't rely on IP's for inter-communication
- DNS for friendly names is built-in if you use custom networks
- You're using custom networks right?
- This gets way easier with Docker Compose in future Section

44 ☐ **Assignment Requirements: CLI App Testing**

- Know how to use -it to get shell in container
- Understand basics of what a Linux distribution is like Ubuntu and CentOS
- Know how to run a container 😬

45 ☐ **Assignment: CLI App Testing**

- Use different Linux distro containers to check curl cli tool version
- Use two different terminal windows to start bash in both centos:7 and ubuntu:14.04, using -it
- Learn the docker container --rm option so you can save cleanup
- Ensure curl is installed and on latest version for that distro
  - ubuntu: apt-get update && apt-get install curl
  - centos: yum update curl
- Check curl --version

46 ☐ **Assignment Requirements: DNS RR Test**

- Know how to use -it to get shell in container
- Understand basics of what a Linux distribution is like Ubuntu and CentOS
- Know how to run a container 😬
- Understand basics of DNS records

47 ☐ **Assignment: DNS Round Robin Test**

- Ever since Docker Engine 1.11, we can have multiple containers on a created network respond to the same DNS address
- Create a new virtual network (default bridge driver)
- Create two containers from elasticsearch:2 image
- Research and use --net-alias search when creating them to give them an additional DNS name to respond to
- Run alpine nslookup search with --net to see the two containers list for the same DNS name
- Run centos curl -s search:9200 with --net multiple times until you see both "name" fields show

48 ☐ **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

49 ☐ **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

50 ☐ **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

51 **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

52 **This Lecture**

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

53 **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

54 **This Lecture**

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

55 **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

56 **This Lecture: Requirements**

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

- Understand Docker Hub basics

57 ☐ **This Lecture**

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

58 ☐ **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

59 ☐ **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

60 ☐ **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)

61 ☐ **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

62 **Section Overview**

- Defining the problem of persistent data
- Key concepts with containers: immutable, ephemeral
- Learning and using Data Volumes
- Learning and using Bind Mounts
- Assignments

63 **Container Lifetime & Persistent Data**

- Containers are usually immutable and ephemeral
- "immutable infrastructure": only re-deploy containers, never change
- This is the ideal scenario, but what about databases, or unique data?
- Docker gives us features to ensure these "separation of concerns"
- This is known as "persistent data"
- Two ways: Volumes and Bind Mounts
- Volumes: make special location outside of container UFS
- Bind Mounts: link container path to host path

64 **Persistent Data: Volumes**

- VOLUME command in Dockerfile
- Also override with docker run -v /path/in/container
- Bypasses Union File System and stores in alt location on host
- Includes it's own management commands under docker volume
- Connect to none, one, or multiple containers at once
- Not subject to commit, save, or export commands
- By default they only have a unique ID, but you can assign name
- Then it's a "named volume"

65 **Persistent Data: Bind Mounting**

- Maps a host file or directory to a container file or directory
- Basically just two locations pointing to the same file(s)
- Again, skips UFS, and host files overwrite any in container
- Can't use in Dockerfile, must be at container run
- … run -v /Users/bret/stuff:/path/container (mac/linux)
- … run -v //c/Users/bret/stuff:/path/container (windows)

66 **Assignment: Named Volumes**

- Database upgrade with containers
- Create a postgres container with named volume psql-data using version 9.6.1

- Use Docker Hub to learn VOLUME path and versions needed to run it
- Check logs, stop container
- Create a new postgres container with same named volume using 9.6.2
- Check logs to validate
- (this only works with patch versions, most SQL DB's require manual commands to upgrade DB's to major/minor versions, i.e. it's a DB limitation not a container one)

67 ☐ **Assignment: Bind Mounts**

- Use a Jekyll "Static Site Generator" to start a local web server
- Don't have to be web developer: this is example of bridging the gap between local file access and apps running in containers
- source code is in the course repo under bindmount-sample-1
- We edit files with editor on our host using native tools
- Container detects changes with host files and updates web server
- start container with docker run -p 80:4000 -v $(pwd):/site bretfisher/jekyll-serve
- Refresh our browser to see changes
- Change the file in _posts\ and refresh browser to see changes
-